Prototype-based Classification and Clustering

Habilitationsschrift

zur Erlangung der Venia legendi für

Informatik

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von Dr.-Ing. Christian Borgelt,
geboren am 6. Mai 1967 in Bünde (Westfalen)

Gutachter: Prof. Dr. Rudolf Kruse
Prof. Dr. Hans-Joachim Lenz
Prof. Dr. Olaf Wolkenhauer

Magdeburg, den 2. November 2005

Contents

Abstract **v**

1 Introduction **1**
1.1 Classification and Clustering 2
1.2 Prototype-based Methods 5
1.3 Outline of this Thesis 9
1.4 Software 10

2 Cluster Prototypes **11**
2.1 Distance Measures 12
2.2 Radial Functions 16
2.3 Prototype Properties 20
2.4 Normalization Modes 22
2.5 Classification Schemes 37
2.6 Related Approaches 41

3 Objective Functions **45**
3.1 Least Sum of Squared Distances 46
3.2 Least Sum of Squared Errors 59
3.3 Maximum Likelihood 62
3.4 Maximum Likelihood Ratio 65
3.5 Other Approaches 69

4 Initialization Methods **73**
4.1 Data Independent Methods 74
4.2 Simple Data Dependent Methods 76
4.3 More Sophisticated Methods 81
4.4 Weight Initialization 94

5 Update Methods **99**
5.1 Gradient Methods . . . 100
5.1.1 General Approach . . . 100
5.1.2 Gradient Descent on Sum of Squared Distances . . . 103
5.1.3 Gradient Descent on Sum of Squared Errors . . . 108
5.1.4 Gradient Ascent on Likelihood Function . . . 113
5.1.5 Problems of Gradient Methods . . . 117
5.2 Alternating Optimization . . . 120
5.2.1 General Approach . . . 120
5.2.2 Classical Crisp Clustering . . . 121
5.2.3 Fuzzy Clustering . . . 125
5.2.4 Expectation Maximization . . . 139
5.3 Competitive Learning . . . 152
5.3.1 Classical Learning Vector Quantization . . . 152
5.3.2 Fuzzy Learning Vector Quantization . . . 159
5.3.3 Size and Shape Parameters . . . 162
5.3.4 Maximum Likelihood Ratio . . . 165
5.4 Guided Random Search . . . 171
5.4.1 Simulated Annealing . . . 171
5.4.2 Genetic or Evolutionary Algorithms . . . 173
5.4.3 Application to Classification and Clustering . . . 175

6 Update Modifications **177**
6.1 Robustness . . . 178
6.1.1 Noise Clustering . . . 178
6.1.2 Shape Regularization . . . 183
6.1.3 Size Regularization . . . 186
6.1.4 Weight Regularization . . . 187
6.2 Acceleration . . . 188
6.2.1 Step Expansion . . . 190
6.2.2 Momentum Term . . . 191
6.2.3 Super Self-Adaptive Backpropagation . . . 191
6.2.4 Resilient Backpropagation . . . 192
6.2.5 Quick Backpropagation . . . 193

7 Evaluation Methods **197**
7.1 Assessing the Classification Quality . . . 198
7.1.1 Causes of Classification Errors . . . 198
7.1.2 Cross Validation . . . 200
7.1.3 Evaluation Measures . . . 202

7.2 Assessing the Clustering Quality . . . 209
7.2.1 Internal Evaluation Measures . . . 210
7.2.2 Relative Evaluation Measures . . . 221
7.2.3 Resampling . . . 228

8 Experiments and Applications 233
8.1 Regularization . . . 234
8.2 Acceleration . . . 239
8.3 Clustering Document Collections . . . 242
8.3.1 Preprocessing the Documents . . . 243
8.3.2 Clustering Experiments . . . 248
8.3.3 Conclusions . . . 260

9 Conclusions and Future Work 263

A Mathematical Background 267
A.1 Basic Vector and Matrix Derivatives . . . 267
A.2 Properties of Radial Functions . . . 270
A.2.1 Normalization to Unit Integral . . . 270
A.2.2 Derivatives . . . 274
A.3 Cholesky Decomposition . . . 278
A.4 Eigenvalue Decomposition . . . 280
A.5 Singular Value Decomposition . . . 284
A.6 Multilinear Regression . . . 285
A.7 Matrix Inversion Lemma . . . 287
A.8 Lagrange Theory . . . 288
A.9 Heron's Algorithm . . . 290
A.10 Types of Averages . . . 290
A.11 The χ^2 Measure . . . 291

B List of Symbols 295

Bibliography 303

Index 327

Curriculum Vitae 339

Abstract

Classification and clustering are, without doubt, among the most frequently encountered data analysis tasks. This thesis provides a comprehensive synopsis of the main approaches to solve these tasks that are based on (point) prototypes, possibly enhanced by size and shape information. It studies how prototypes are defined, how they interact, how they can be initialized, and how their parameters can be optimized by three main update methods (gradient methods, alternating optimization, competitive learning), which are applied to four objective functions (sum of squared distances, sum of squared errors, likelihood, likelihood ratio). Besides organizing these methods into such a unified framework, the main contributions of this thesis are extensions of existing approaches that render them more flexible and more robust or accelerate the learning process. Among these are shape and size parameters for (fuzzified) learning vector quantization, shape and size regularization methods, and a transfer of neural network techniques to clustering algorithms. The practical relevance of these extensions is demonstrated by experimental results with an application to document structuring.

Zusammenfassung

Klassifikation und Clustering gehören ohne Zweifel zu den am häufigsten anzutreffenden Datenanalyseaufgaben. Diese Schrift bietet eine umfassende Zusammenschau der Hauptansätze zur Lösung dieser Aufgaben, die auf (Punkt-)Prototypen basieren, möglicherweise angereichert um Größen- und Forminformation. Es wird untersucht, wie Prototypen definiert werden können, wie sie wechselwirken, wie sie initialisiert werden und wie ihre Parameter mit drei Hauptaktualisierungsmethoden (Gradientenverfahren, alternierende Optimierung, Wettbewerbslernen) optimiert werden können, die auf vier Zielfunktionen angewandt werden (Summe der quadratischen Abstände, Summe der quadrierten Fehler, Likelihood, Likelihood-Verhältnis). Neben der Einordnung dieser Methoden in einen solchen einheitlichen Rahmen bestehen die Hauptbeiträge dieser Arbeit in Erweiterungen existierender Ansätze, die sie flexibler und robuster machen oder den Lernprozeß beschleunigen. Dazu gehören etwa Größen- und Formparameter für die (fuzzifizierte) lernende Vektorquantisierung, Methoden für die Form- und Größenregularisierung, sowie eine Übertragung von Methoden, die für neuronale Netze entwickelt wurden, auf Clusteringalgorithmen. Die praktische Relevanz dieser Erweiterungen wird mit experimentellen Ergebnissen aus einer Anwendung zur Dokumentenstrukturierung belegt.

Chapter 1

Introduction

Due to the extremely rapid growth of collected data, which has rendered manual analysis virtually impossible, recent years have seen an intense interest in intelligent computer-aided data analysis methods (see, for example, [Fayyad *et al.* 1996a, Nakhaeizadeh 1998a, Berthold and Hand 1999, Witten and Frank 1999, Hand *et al.* 2001, Berry and Linoff 2004]).

On the list of data analysis tasks frequently occurring in applications, *classification* (understood as the construction of a classifier from labeled data) and *clustering* (that is, the division of a data set into groups of similar cases or objects) occupy very high, if not the highest ranks [Nakhaeizadeh 1998b]. As a consequence a large variety of methods to tackle these tasks has been developed, ranging from decision trees, (naïve) Bayes classifiers, rule induction, and different types of artificial neural networks for classification to (fuzzy) c-means clustering, hierarchical clustering, and learning vector quantization (see below for references for selected methods).

Several of these methods are based on a fundamentally very simple, but nevertheless very effective idea, namely to describe the data under consideration by a set of *prototypes*, which capture characteristics of the data distribution (like location, size, and shape), and to classify or to divide the data set based on the *similarity* of the data points to these prototypes. The approaches relying on this idea differ mainly in the way in which prototypes are described and how they are updated during the model construction step. The goal of this thesis is to develop a unified view of a certain subset of such prototype-based approaches, namely those that are essentially based on point prototypes, and to transfer ideas from one approach to another in order to improve the performance and usability of known algorithms.

1.1 Classification and Clustering

The terms "classification" and "to classify" are ambiguous. In the area of Machine Learning [Mitchell 1997] they usually turn up in connection with classifiers like decision trees, (artificial) neural networks, or (naïve) Bayes classifiers and denote the process of assigning a class from a *predefined set* to an object or case under consideration. Consequently, a *classification problem* is the task to *construct a classifier*—that is, an automatic procedure to assign class labels—from a data set of labeled case descriptions.

In classical statistics [Everitt 1998], however, these terms usually have a different meaning: they describe the process of *dividing a data set* of sample cases *into groups* of similar cases, with the groups *not* predefined, but to be found by the classification algorithm. This process is also called *classification*, because the groups to be found are often called *classes*, thus inviting unpleasant confusion. Classification in the sense of Machine Learning is better known in statistics as *discriminant analysis*, although this is sometimes, but rarely, called *classification* in statistics as well.

In order to avoid the confusion that may result from this ambiguity, the latter process, i.e., dividing a data set into groups of similar cases, is often called *clustering* or *cluster analysis*, replacing the ambiguous term *class* with the less ambiguous *cluster*. An alternative is to speak of *supervised classification* if the assignment of predefined classes is referred to and of *unsupervised classification* if clustering is the topic of interest. In this thesis, however, I adopt the first option and distinguish between *classification* and *clustering*, reserving the former for classifier construction.

To characterize these two tasks more formally, I assume that we are given a *data set* $\mathbf{X} = \{\vec{x}_1, \ldots, \vec{x}_n\}$ of m-dimensional vectors $\vec{x}_j = (x_{j1}, \ldots, x_{jm})$, $1 \leq j \leq n$. Each of these vectors represents an example case or object, which is described by stating the values of m attributes. Alternatively, the data set may be seen as a *data matrix* $\mathbf{X} = (x_{jk})_{1 \leq j \leq n, 1 \leq k \leq m}$, each row of which is a data vector and thus represents an example case. Which way of viewing the data is more convenient depends on the situation and therefore I switch freely between these two very similar representations.

In general, the attributes used to describe the example cases or objects may have nominal, ordinal, or metric scales[1], depending on the property of

[1] An attribute is called *nominal* if its values can only be tested for equality. It is called *ordinal* if there is a natural order, so that a test can be performed which of two values is greater than the other. Finally, an attribute is called *metric* if numeric differences of two values are meaningful. Sometimes metric attributes are further subdivided according to whether ratios of two values are meaningful (ratio scale) or not (interval scale).

the object or case they refer to. Here, however, I confine myself to metric attributes and hence to numeric (real-valued) vectors. The main reason for this restriction is that prototype-based methods rely on a notion of *similarity*, which is usually based on a *distance measure* (details are given in the next chapter). Even worse, the greater part of such methods not only need to *measure* the distance of a data vector from a prototype, but must also be able to *construct* a prototype from a set of data vectors. If the attributes are not metric, this can lead to unsurmountable problems, so that one is forced to *select* a representative, as, for example, in so-called *medoid clustering* [Kaufman and Rousseeuw 1990, Chu *et al.* 2001], in which the most central element of a set of data vectors is chosen as the prototype. However, even for this approach a distance measure is needed in order to determine which of the elements of the group is most central.

Since plausible, generally applicable distance measure are difficult to find for nominal and ordinal attributes, it is usually easiest to transform them into metric attributes in a preprocessing step. For example, a very simple approach is so-called *1-in-n encoding*, which constructs a metric (or actually binary) attribute for each value of a nominal attribute. In a data vector a value of 1 is then assigned to the metric attribute that represents the nominal value the example case has, and a value of 0 to all metric attributes that correspond to other values of the nominal attribute. Formally: let A be a nominal attribute with domain $\mathrm{dom}(A) = \{a_1, \ldots, a_s\}$. To encode an assignment $A = a_i$, we replace A by s metric (binary) attributes $B_1, \ldots, B_s$ and set $B_i = 1$ and $B_k = 0$, $1 \leq k \leq s, k \neq i$.

If the task is **clustering**, we are given only a data set (or data matrix). The goal is to divide the data vectors into groups or *clusters*, with the number of groups either specified by a user or to be found by the clustering algorithm. The division should be such that data vectors from the same group are as similar as possible and data vectors from different groups are as dissimilar as possible. As already mentioned above, the similarity of two data vectors is usually based on a (transformed) distance measure, so that we may also say: such that data vectors from the same group are as close to each other as possible and data vectors from different groups are as far away from each other as possible. Note that these two objectives are complementary: often one can reduce the (average) distance between data vectors from the same group by splitting a group into two. However, this may reduce the (average) distance between vectors from different groups.

Classical clustering approaches are often *crisp* or *hard*, in the sense that the data points are partitioned into disjoint groups. That is, each data point is assigned to exactly one cluster. However, in applications such hard

partitioning approaches are not always appropriate for the task at hand, especially if the groups of data points are not well separated, but rather form more densely populated regions, which are separated by less densely populated ones. In such cases the boundary between clusters can only be drawn with a certain degree of arbitrariness, leading to uncertain assignments of the data points in the less densely populated regions.

To cope with such conditions it is usually better to allow for so-called *degrees of membership*. That is, a data point may belong to several clusters and its membership is quantified by a real number, with 1 meaning full membership and 0 meaning no membership at all. The meaning of degrees of membership between 0 and 1 may differ, depending on the underlying assumptions. They may express intuitively how typical a data point is for a group, or may represent preferences for a hard assignment to a cluster.

If similarity is based on distances, I call the approach *distance-based clustering*. Alternatively, one may try to find proper groups of data points by building a probabilistic model from a user-defined family for each of the groups, trying to maximize the *likelihood* of the data. Such approaches, which very naturally assign degrees of membership to data vectors, I call *probabilistic clustering*. However, in the following chapter we will see that these categories are not strict, since commonly used probabilistic models are based on distance measures as well. This offers the convenient possibility to combine several approaches into a unified scheme.

If the task at hand is **classification**, a data set (or data matrix) is not enough. In addition to the vectors stating the values of descriptive attributes for the example cases, we need a vector $\vec{z} = (z_1, \ldots, z_n)$, which states the classes of the data points. Some of the approaches studied in this thesis, like *radial basis function networks*, allow for values z_j, $1 \leq j \leq n$, that are real numbers, thus turning the task into a numeric prediction problem. Most of the time, however, I confine myself to true classification, where the z_j come from a finite (and usually small) set of class labels, e.g. $z_j \in \{1, \ldots, s\}$.

The goal of classification is to construct or to parameterize a procedure, called a *classifier*, which assigns class labels based on the values of the descriptive attributes (i.e. the elements of the vectors $\vec{x}_j$). Such a classifier may work in a crisp way, yielding a unique class label for each example case, or it may offer an indication of the reliability of the classification by assigning probabilities, activations, or membership degrees to several classes. The latter case corresponds to the introduction of membership degrees into clustering, as it can be achieved by the same means.

To measure the classification quality, different so-called *loss functions* may be used, for example, *0-1 loss*, which simply counts the misclassifi-

cations (on the training data set or on a separate test data set), and the *sum of squared errors*, which is computed on a 1-in-n encoding of the class labels[2] and can take a measure of reliability, for example, probabilities for the different classes, into account (cf. Sections 3.2 and 7.1).

1.2 Prototype-based Methods

Among the variety of methods that have been developed for classification and clustering, this thesis focuses on what may be called *prototype-based approaches*. Prototype-based methods try to describe the data set to classify or to cluster by a (usually small) set of *prototypes*, in particular *point prototypes*, which are simply points in the data space. Each prototype is meant to capture the distribution of a group of data points based on a concept of *similarity* to the prototype or *closeness* to its location, which may be influenced by (prototype-specific) size and shape parameters.

The main advantages of prototype-based methods are that they provide an intuitive summarization of the given data in few prototypical instances and thus lead to plausible and interpretable cluster structures and classification schemes. Such approaches are usually perceived as intuitive, because human beings also look for similar past experiences in order to assess a new situation and because they summarize their experiences in order to avoid having to memorize too many and often irrelevant individual details.

In order to convey a better understanding of what I mean by prototype-based methods and what their characteristics are, the following lists provide arguments how several well-known data analysis methods can be categorized as based or not based on prototypes. The main criteria are whether the data is captured with few(!) representatives and whether clustering or classification relies on the closeness of a data point to these representatives. Nevertheless, this categorization is not strict as there are boundary cases.

Approaches based on prototypes

- **(naïve and full) Bayes classifiers**
  [Good 1965, Duda and Hart 1973, Langley *et al.* 1992]
  A very frequent assumption in Bayesian classification is that numeric attributes are normally distributed and thus a class can be described in a numeric input space by a multivariate normal distribution, either

[2] Note that the sum of squared errors is proportional to the 0-1 loss if the classifier yields crisp predictions (i.e. a unique class for each example case).

axes-parallel (naïve) or in general orientation (full). This may be seen as a description with one prototype per class, located at the mean vector, with a covariance matrix specifying shape and size.

- **radial basis function neural networks**
  [Rojas 1993, Haykin 1994, Zell 1994, Anderson 1995, Bishop 1995, Nauck *et al.* 2003]
  Radial basis function networks rely on center vectors and reference radii to capture the distribution of the data. More general versions employ covariance matrices to describe the shape of the influence region of each prototype (reference vector). However, learning results do not always yield a representative description of the data as the prominent goal is error minimization and not data summarization.

- **(fuzzy) c-means clustering**
  [Ball and Hall 1967, Ruspini 1969, Dunn 1973, Hartigan und Wong 1979, Lloyd 1982, Bezdek *et al.* 1999, Höppner *et al.* 1999]
  Since in c-means approaches clustering is achieved by minimizing the (weighted) sum of (squared) distances to a set of c center vectors, with c usually small, this approach is clearly prototype-based. More sophisticated approaches use covariance matrices to modify the distance measure and thus to model different cluster shapes and sizes. Fuzzy approaches add the possibility of "soft" cluster boundaries.

- **expectation maximization**
  [Dempster *et al.* 1977, Everitt and Hand 1981, Jamshidian and Jennrich 1993, Bilmes 1997]
  Expectation maximization clustering is based on a mixture model of the data generation process and usually assumes multivariate normal distributions as the mixture components. It is thus very similar to the construction of a naïve or full Bayes classifier for unknown class labels, with the goal to maximize the likelihood of the data given the model. This approach is also very closely related to fuzzy clustering.

- **learning vector quantization**
  [Gray 1984, Kohonen 1986, Kohonen 1990, NNRC 2002]
  The reference or codebook vectors of this approach can be seen as prototypes capturing the distribution of the data, based on the distance of the data points to these vectors. It has to be conceded, though, that in learning vector quantization the number of reference vectors is often much higher than the number of clusters in clustering, making the term "prototypes" sound a little presumptuous.

- **fuzzy rule-based systems / neuro-fuzzy systems**
  [Zadeh 1965, Mamdani and Assilian 1975, Takagi and Sugeno 1985, Nauck and Kruse 1997, Nauck *et al.* 1997, Nauck *et al.* 2003]

  Fuzzy rule-based systems and especially some types of neuro-fuzzy systems are closely related to radial basis function neural networks. They differ mainly in certain restrictions placed on the description of the similarity and distance to the prototypes (usually each input dimension is handled independently in order to simplify the structure of the classifier and thus to make it easier to interpret).

Approaches not based on prototypes

- **decision and regression trees**
  [Breiman *et al.* 1984, Quinlan 1986, Quinlan 1993]

  For numeric attributes decision and regression trees divide the input space by axes-parallel hyperplanes, which are described by simple threshold values. Generalizations to so-called oblique decision trees [Murthy *et al.* 1994] allow for hyperplanes in general orientation. However, no prototypes are constructed or selected and no concept of similarity is employed. It should be noted that decision and regression trees are closely related to rule-based approaches, since each path in the tree can be seen as a (classification) rule.

- **classical rule-based systems**
  [Michalski *et al.* 1983, Clark and Niblett 1989, Clark and Boswell 1991, Quinlan 1993, Cohen 1995, Domingos 1996]

  Depending on its antecedent, a classification rule may capture a limited region of the data space. However, this region is not defined by a prototype and a distance or similarity to it. The classification is rather achieved by separating hyperplanes, in a similar way as decision trees and multilayer perceptrons do. It should be noted that rules easily lose their interpretability if general hyperplanes are used that cannot be described by a simple threshold value.

- **multilayer perceptrons**
  [Rojas 1993, Haykin 1994, Zell 1994, Anderson 1995, Bishop 1995, Nauck *et al.* 2003]

  If multilayer perceptrons are used to solve classification problems, they describe piecewise linear decision boundaries between the classes, represented by normal vectors of hyperplanes. They do not capture the distribution of the data points on either side of these boundaries with

(constructed or selected) prototypes. Seen geometrically, they achieve a classification in a similar way as (oblique) decision trees.

Boundary cases

- **hierarchical agglomerative clustering**
  [Sokal and Sneath 1963, Johnson 1967, Bock 1974, Everitt 1981, Jain and Dubes 1988, Mucha 1992]

  With the possible exception of the *centroid method*, hierarchical agglomerative clustering only groups data points into clusters without constructing or selecting a prototypical data point. However, since only the *single-linkage* method can lead to non-compact clusters, which are difficult to capture with prototypes, one may construct prototypes by forming a mean vector for each resulting cluster.

- **support vector machines**
  [Vapnik 1995, Vapnik 1998, Cristianini and Shawe-Taylor 2000, Schölkopf and Smola 2002]

  Support vector machines can mimic radial basis function networks as well as multilayer perceptrons. Hence, whether they may reasonably be seen as prototype-based depends on the *kernel function* used, whether a reduction to *few support vectors* takes place, and on whether the support vectors actually try to capture the distribution of the data (radial basis function networks) or just define the location of the decision boundary (multilayer perceptrons).

- **k-nearest neighbor / case-based reasoning**
  [Duda and Hart 1973, Dasarathy 1990, Aha 1992, Kolodner 1993, Wettschereck 1994]

  Since in k-nearest neighbor classification the closeness or similarity of a new data point to the points in the training data set determine the classification, this approach may seem prototype-based. However, each case of the training set is a reference point and thus there is no reduction to (few) prototypes or representatives.

As a further characterization of prototype-based methods, approaches in this direction may be divided according to whether they construct prototypes or only select them from the given data set, whether they can incorporate size and shape information into the prototypes or not, whether the resulting model can be interpreted probabilistically or not, and whether they rely on an iterative improvement or determine the set of prototypes in a single run (which may take the form of an iterative thinning).

1.3 Outline of this Thesis

Since prototype-based methods all share the same basic idea, it is possible and also very convenient to combine them into a unified scheme. Such a view makes it easier to transfer ideas that have been developed for one model to another. In this thesis I present some such transfers, for example, shape and size parameters for learning vector quantization, as well as extensions of known approaches that endow them with more favorable characteristics, for example, improved robustness or speed. This thesis is structured as follows:

In **Chapter 2** I introduce the basics of (cluster) prototypes. I define the core prototype properties like location, shape, size, and weight, and study how these properties can be described with the help of parameters of distance measures and radial functions. Since classification and clustering methods always rely on several prototypes, I also study normalization methods, which relate the similarities of a data point to different prototypes to each other and modify them in order to obtain certain properties. Finally, I briefly discuss alternative descriptions like neuro-fuzzy systems as well as extensions to non-point prototypes.

Since most prototype-based classification and clustering approaches can be seen as optimizing some quantity, **Chapter 3** studies the two main paradigms for objective functions: least sum of squared deviations and maximum likelihood. While the former is based on either a distance minimization (clustering) or an error minimization (classification), the latter is based on a probabilistic approach and tries to adapt the parameter of the model so that the likelihood or likelihood ratio of the data is maximized.

Chapter 4 is dedicated to the initialization of the clustering or classification process. I briefly review simple, but very commonly used data independent as well as data dependent methods before turning to more sophisticated approaches. Several of the more sophisticated methods can be seen as clustering methods in their own right (or may even have been developed as such). Often they determine an (initial) clustering or classification in a single run (no iterative improvement). The chapter concludes with a review of weight initialization for classification purposes.

Chapter 5 is the most extensive of this thesis and considers the main approaches to an iterative improvement of an initial clustering or classification: gradient descent, alternating optimization, and competitive learning. I try to show that these methods are closely related and often enough one specific approach turns out to be a special or boundary case of another that is based on a different paradigm. As a consequence it becomes possible to transfer improvements that have been made to a basic algorithm

in one domain to another, for example, the introduction of size and shape parameters into learning vector quantization. The chapter concludes with a brief consideration of guided random search methods and their application to clustering, which however, turn out to be inferior to other strategies.

In **Chapter 6** I discuss approaches to modify the basic update schemes in order to endow them with improved properties. These include methods to handle outliers as well as regularization techniques for the size, shape, and weight of a cluster in order to prevent undesired results. Furthermore, I study a transfer of neural network training improvements, like self-adaptive, resilient, or quick backpropagation, to fuzzy clustering and other iterative update schemes in order to accelerate the learning process.

In **Chapter 7** I consider the question how to assess the quality of a classifier or cluster model. Approaches for the former are fairly straightforward and are mainly based on the error rate on a *validation data set*, with *cross validation* being the most prominent strategy. Success criteria for clustering are somewhat less clear and less convincing in their ability to judge the quality of a cluster model appropriately. The main approaches are evaluation measures and resampling methods.

In **Chapter 8** I report experiments that were done with some of the developed methods in order to demonstrate their benefits. As a substantial application I present an example of clustering web page collections with different algorithms, which emphasizes the relevance of, for example, the generalized learning vector quantization approach introduced in Chapter 5 as well as the regularization methods studied in Chapter 6.

The thesis finishes with **Chapter 9**, in which I draw conclusions and point out some directions for future work.

1.4 Software

I implemented several (though not all) of the clustering and classification methods described in this thesis and incorporated some of the suggested improvements and modifications into these programs. Executable programs for Microsoft Windows™ and Linux™ as well as the source code can be found on my WWW page:

`http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html`

All programs are distributed either under the Gnu General Public License or the Gnu Lesser (Library) General Public License (which license applies is stated in the documentation that comes with the program).

Chapter 2

Cluster Prototypes

The clustering and classification methods studied in this thesis are based on finding a set of c *cluster prototypes*, each of which captures a group of data points that are similar to each other. A clustering result consists of these prototypes together with a rule of how to assign data points to these prototypes, either crisp or with degrees of memberships. For classification purposes an additional decision function is needed. This function draws on the assignments to *clusters* to compute an assignment to the given *classes*, again either crisp or with degrees of membership.[1]

Since the standard way of measuring similarity starts from a *distance measure*, Section 2.1 reviews some basics about such functions and their properties. Similarity measures themselves are then based on *radial functions*, the properties of which are studied in Section 2.2.

Section 2.3 introduces the core properties of cluster prototypes—namely *location*, *size*, *shape*, and *weight*—and elaborates the mathematical means to describe these properties as well as how to measure the similarity of a data point to a cluster prototype based on these means.

However, usually the assignment to a cluster is not obtained directly from the raw similarity value, but from a normalization of it over all clusters. Therefore Section 2.4 studies *normalization modes*, which provide (crisp or graded) assignment rules for *clusters*. Afterwards the (crisp or graded) assignment of data points to *classes* is examined in Section 2.5. The chapter concludes with a brief survey of related approaches in Section 2.6.

[1] Note that *cluster* differs from *class*. There may be many more clusters than classes, so that each class comprises several clusters of data points, even though the special case, in which each class consists of one cluster is often worth considering.

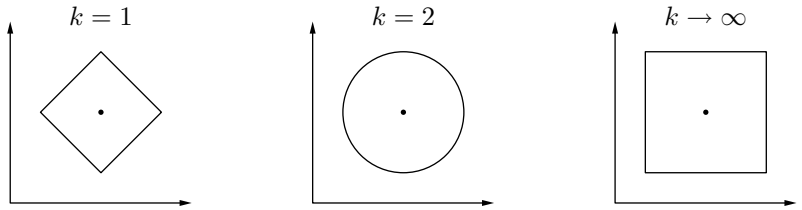

Figure 2.1: Circles for three distance measures from the Minkowski family.

2.1 Distance Measures

Distance measures are the most common way to measure the similarity of data points. Axiomatically, a distance measure is defined as follows:

Definition 2.1 *A function $d : X \times X \to \mathbb{R}_0^+$, where X is an arbitrary set, is called a* **distance measure** *or simply a* **distance** *or* **metric** *(on X) iff it satisfies the following three axioms $\forall x, y, z \in X$:*

- $d(x, y) = 0 \Leftrightarrow x = y$,
- $d(x, y) = d(y, x)$ *(symmetry)*,
- $d(x, z) \leq d(x, y) + d(y, z)$ *(triangle inequality)*.

The best-known examples of distance measures come from the so-called **Minkowski family**, which is defined as

$$d_k(\vec{x}, \vec{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^k \right)^{\frac{1}{k}},$$

where k is a parameter. It contains the following special cases:

$k = 1$: **Manhattan** or **city block distance**,
$k = 2$: **Euclidean distance**,
$k \to \infty$: **maximum distance**, i.e., $d_\infty(\vec{x}, \vec{y}) = \max_{i=1}^{m} |x_i - y_i|$

The properties of distance measures can be illustrated nicely by considering how circles look with them, because a circle is defined as the set of points that have the same given distance (the *radius* of the circle) from a given point (the *center* of the circle). Circles corresponding to the three special cases mentioned above are shown in Figure 2.1.

An important advantage of the best-known and most commonly used distance measure, the Euclidean distance, is that it is invariant w.r.t. orthogonal linear transformations (translation, rotation, reflection), that is, its value stays the same if the vectors are transformed according to

$$\vec{x} \mapsto \mathbf{R}\vec{x} + \vec{o},$$

where $\mathbf{R}$ is an arbitrary orthogonal[2] $m \times m$ matrix and $\vec{o}$ is an arbitrary (but fixed) m-dimensional vector. While all distance measures from the Minkowski family are invariant w.r.t. translation, only the Euclidean distance is invariant w.r.t. rotation and (arbitrary) reflection. However, even the Euclidean distance, as well as any other distance measure from the Minkowski family, is *not* scale-invariant. That is, in general the Euclidean distance changes its value if the data points are mapped according to

$$\vec{x} \mapsto \mathrm{diag}(s_1, \ldots, s_m)\,\vec{x},$$

where the s_i, $1 \leq i \leq m$, which form a diagonal matrix, are the scaling factors for the m axes, at least one of which differs from 1.

A distance measure that is invariant w.r.t. orthogonal linear transformations as well as scale-invariant is the so-called **Mahalanobis distance** [Mahalanobis 1936]. It is frequently used in clustering and defined as

$$d(\vec{x}, \vec{y}; \mathbf{\Sigma}) = \sqrt{(\vec{x} - \vec{y})^\top \mathbf{\Sigma}^{-1} (\vec{x} - \vec{y})},$$

where the $m \times m$ matrix $\mathbf{\Sigma}$ is the covariance matrix of the considered data set $\mathbf{X}$, which (implicitly assuming that all data points are realizations of independent and identically distributed random vectors) is computed as

$$\mathbf{\Sigma} = \frac{1}{n-1} \sum_{j=1}^{n} (\vec{x}_j - \vec{\mu})(\vec{x}_j - \vec{\mu})^\top \qquad \text{with} \qquad \vec{\mu} = \frac{1}{n} \sum_{j=1}^{n} \vec{x}_j.$$

From these formulae it is easily seen that the Mahalanobis distance is actually scale-invariant, because scaling the data points also scales the covariance matrix and thus computing the distance based on the inverse of the covariance matrix removes the scaling again. However, this also makes the distance measure dependent on the data set: in general the Mahalanobis distance with the covariance matrix computed from one data set does *not* have the stated invariance characteristics for another data set.

[2] A square matrix is called *orthogonal* if its columns are pairwise orthogonal and have length 1. As a consequence the transpose of such a matrix equals its inverse.

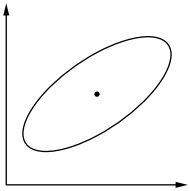

Figure 2.2: A circle for the Mahalanobis distance. The directions of the axes of the ellipse are the eigenvectors of the covariance matrix $\mathbf{\Sigma}$.

It should be noted that the Mahalanobis distance, which is actually a family of distance measures parameterized by a covariance matrix, contains the Euclidean distance as a special case, namely for $\mathbf{\Sigma} = \mathbf{1}$:

$$d(\vec{x}, \vec{y}; \mathbf{1}) = \sqrt{(\vec{x} - \vec{y})^\top (\vec{x} - \vec{y})} = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}.$$

This specific covariance matrix results for uncorrelated data, which are normalized to standard deviation 1 in each dimension.

The insight that the Euclidean distance is a special case of the Mahalanobis distance already provides us with an intuition how circles look with this distance: they should be distorted Euclidean circles. And indeed, circles are ellipses in general orientation, as shown in Figure 2.2, with the axes ratio and the rotation angle depending on the covariance matrix $\mathbf{\Sigma}$.

Mathematically, this can be nicely demonstrated by a Cholesky decomposition or eigenvalue decomposition of the covariance matrix, techniques that are reviewed in some detail in Section A.3 and Section A.4, respectively, in the appendix. Eigenvalue decomposition, for example, makes it possible to write an $m \times m$ covariance matrix $\mathbf{\Sigma}$ as

$$\mathbf{\Sigma} = \mathbf{T}\mathbf{T}^\top \qquad \text{with} \qquad \mathbf{T} = \mathbf{R}\,\mathrm{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_m}),$$

where $\mathbf{R}$ is an $m \times m$ orthogonal matrix (or more specifically: a rotation matrix), the columns of which are the eigenvectors of $\mathbf{\Sigma}$ (normalized to length 1), and the λ_i, $1 \leq i \leq m$, are the eigenvalues of $\mathbf{\Sigma}$. As a consequence the inverse $\mathbf{\Sigma}^{-1}$ of $\mathbf{\Sigma}$ can be written as

$$\mathbf{\Sigma}^{-1} = \mathbf{U}^\top \mathbf{U} \qquad \text{with} \qquad \mathbf{U} = \mathrm{diag}\left(\lambda_1^{-\frac{1}{2}}, \ldots, \lambda_m^{-\frac{1}{2}}\right) \mathbf{R}^\top$$

(cf. page 283 in Section A.4). The matrix $\mathbf{U}$ describes a mapping of an ellipse that is a circle w.r.t. the Mahalanobis distance to a circle w.r.t. the Euclidean distance by rotating (with $\mathbf{R}^\top$) the coordinate system to

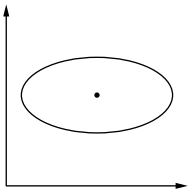

Figure 2.3: A circle for a Mahalanobis distance that uses a diagonal covariance matrix $\mathbf{\Sigma}$, so that the eigenvectors of $\mathbf{\Sigma}$ are parallel to the coordinate axes.

the eigensystem of the covariance matrix $\mathbf{\Sigma}$ and then scaling it (with the diagonal matrix) to unit standard deviation in each direction. Therefore the argument of the square root of the Mahalanobis distance, written as

$$(\vec{x} - \vec{y})^\top \mathbf{\Sigma}^{-1} (\vec{x} - \vec{y}) \;=\; (\vec{x} - \vec{y})^\top \mathbf{U}^\top \mathbf{U} (\vec{x} - \vec{y}) \;=\; (\vec{x}\,' - \vec{y}\,')^\top (\vec{x}\,' - \vec{y}\,'),$$

where $\vec{x}\,' = \mathbf{U}\vec{x}$ and $\vec{y}\,' = \mathbf{U}\vec{y}$, is equivalent to the squared Euclidean distance in the properly scaled eigensystem of the covariance matrix.

It should be noted that in the special case, where the covariance matrix is a diagonal matrix (uncorrelated attributes), the Mahalanobis distance describes only a scaling of the axes, since in this case the orthogonal matrix $\mathbf{R}$ is the unit matrix. As a consequence circles w.r.t. this distance measure are axes-parallel ellipses, like the one shown in Figure 2.3.

It should also be noted that the Mahalanobis distance was originally defined with the covariance matrix of the given data set $\mathbf{X}$ [Mahalanobis 1936] (see above). However, in clustering and classification it is more often used with cluster-specific covariance matrices, so that individual sizes and shapes of clusters can be modeled (see Section 2.3). This is highly advantageous, even though one has to cope with the drawback that there is no unique distance between two data points anymore, because distances w.r.t. one cluster may differ from those w.r.t. another cluster.

Even though there are approaches to prototype-based clustering and classification that employ other members of the Minkowski family—see, for example, [Bobrowski and Bezdek 1991], who study the city block and the maximum distance—or a cosine-based distance [Klawonn and Keller 1999] (which, however, does not lead to substantial differences, see Section 8.3.1), the vast majority of approaches rely on either the Euclidean distance or the Mahalanobis distance. Besides the convenient properties named above, an important reason for this preference is that some of the update methods discussed in Chapter 5 need the derivative of the underlying distance measure, and this derivative is much easier to compute for quadratic forms like the Euclidean distance or the Mahalanobis distance.

2.2 Radial Functions

In principle, prototype-based classification and clustering methods can work directly with a distance measure. However, it is often more convenient to use a *similarity measure* instead, which can be obtained by applying a (non-linear) mapping of a certain type to a distance measure. Such a mapping can be described with a so-called *radial function* and provides flexible means to control the region and strength of influence of a cluster prototype.

Definition 2.2 *A function* $f : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ *satisfying*

- $\lim_{r\to\infty} f(r) = 0$ *and*
- $\forall r_1, r_2 \in \mathbb{R}_0^+ : r_2 > r_1 \Rightarrow f(r_2) \leq f(r_1)$
  (i.e., f is monotonically non-increasing)

is called a **radial function**.

The reason for the name *radial function* is that if its argument is a distance, it can be seen as being defined on a *radius* r (Latin for *ray*) from a point, from which the distance is measured. As a consequence it has the same value on all points of a circle w.r.t. the underlying distance measure.

Radial functions possess the properties we expect intuitively from a similarity measure: the larger the distance, the smaller the similarity should be, and for infinite distance the similarity should vanish. Sometimes it is convenient to add the condition $f(0) = 1$, because similarity measures are often required to be normalized in the sense that they are 1 for indistinguishable data points. However, I refrain from adding this requirement here, because there are several important approaches that use radial functions not satisfying this constraint (for example, standard fuzzy c-means clustering).

Examples of well-known radial functions include:

- **Generalized Cauchy function**

$$f_{\text{Cauchy}}(r; a, b) \quad = \quad \frac{1}{r^a + b}$$

  The standard Cauchy function is obtained from this definition for $a = 2$ and $b = 1$. Example graphs of this function for a fixed value $b = 1$ and different values of a are shown in Figure 2.4 on the left. For $b = 1$, the limits for $a \to 0$ and $a \to \infty$ are shown in Figure 2.5 on the left and right, respectively. Example graphs for a fixed value of $a = 2$ and then different values of b are shown in Figure 2.6. Only for $b = 1$ we get a standard similarity measure satisfying $f(0) = 1$.

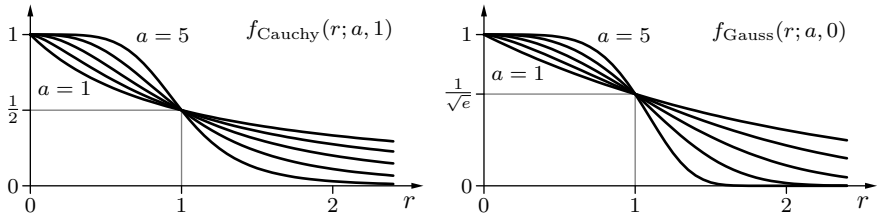

Figure 2.4: Generalized Cauchy (left, $b = 1$) and Gaussian function (right) for different exponents a of the radius r ($a \in \{1, 1.4, 2, 3, 5\}$).

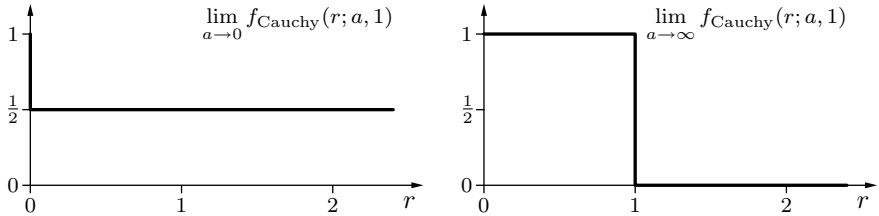

Figure 2.5: Limits of the generalized Cauchy for $a \to 0$ (left) and $a \to \infty$ (right). The limits of the generalized Gaussian function are analogous.

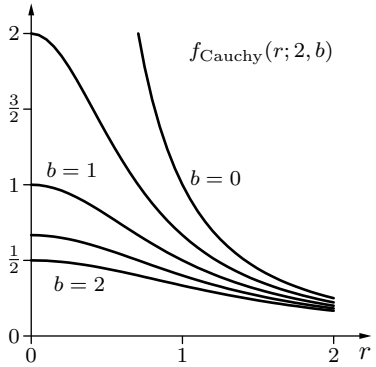

Figure 2.6: Generalized Cauchy function for different values of the parameter b ($b \in \{0, 0.5, 1, 1.5, 2\}$, $a = 2$). Note that independent of the value of a we get a standard similarity measure (satisfying the condition $f(0) = 1$) only for $b = 1$.

- **Generalized Gaussian function**

$$f_{\text{Gauss}}(r; a, b) \quad = \quad e^{-\frac{1}{2}r^a}$$

Example graphs of this function for different values of a (note that the parameter b has no influence and thus may be fixed at 0) are shown in Figure 2.4 on the right. Its general shape is obviously very similar to that of the generalized Cauchy function, although I will point out a decisive difference in Section 2.4. Note that the limiting functions for $a \to 0$ and $a \to \infty$ are analogous to those shown in Figure 2.5 for the generalized Cauchy function with the only difference that for $a \to 0$ the limit value is $e^{-\frac{1}{2}}$ instead of $\frac{1}{2}$. Note also that all members of this family are proper similarity measures satisfying $f(0) = 1$.

- **Trapezoidal (triangular/rectangular) function**

$$f_{\text{trapez}}(r; a, b) \quad = \quad \begin{cases} 1, & \text{if } r \leq b, \\ \frac{r-b}{a-b}, & \text{if } b < r < a, \\ 0, & \text{if } r \geq a. \end{cases}$$

A triangular function is obtained from this definition for $b = 0$, a rectangular function for $a = b$. Example graphs of these functions are shown in Figure 2.7 on the left.

- **Splines of different order**
Splines are piecewise polynomial functions. The highest exponent of each polynomial is called the *order* of the spline. They satisfy certain smoothness constraints and are defined by a set of knots and values for the function value and certain derivatives at these knots. Triangular functions can be seen as splines of order 1. An example of a cubic spline (order 3) is shown in Figure 2.7 on the right.

Of course, an abundance of other radial functions may be considered, like the *cosine down to zero*, which is defined as $f_{\text{cosine}}(r) = 2\cos(r) + 1$ for $r < \pi$ and 0 otherwise. However, most of these functions suffer from certain drawbacks. Trapezoidal functions as well as splines, for example, have only a bounded region in which they do not vanish, which leads to problems in several of the clustering and classification methods studied later. Therefore I confine myself to the first two functions in the above list, which have several convenient properties, one of which is that for a fixed value of b they have a reference point at $r = 1$ through which all curves pass that result from different values of a. This makes it very easy to relate them to reference radii that describe the size of clusters (see the next section).

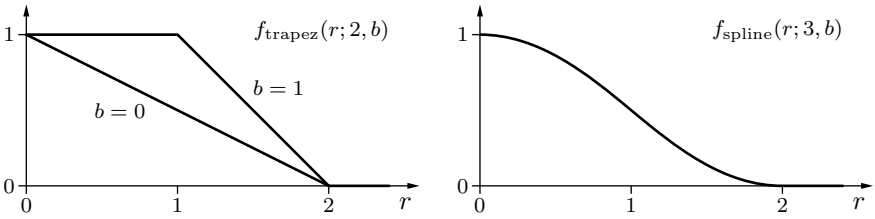

Figure 2.7: Trapezoidal/triangular function (left) and cubic spline (right).

In some approaches the radial function describes a *probability density* of an underlying data generation process rather than a similarity. In such a case the radial function has to be normalized, so that its integral over the whole data space is 1. Formally, this is achieved by enhancing the radial function with a normalization factor. However, whether a normalization is possible depends on the chosen function and its parameters.

The generalized Cauchy function can be normalized to a unit integral if the parameter b is non-negative and the parameter a is greater than the dimension m of the data space. This normalized version is defined as

$$\gamma_{\text{Cauchy}}(a, b, m, \boldsymbol{\Sigma}) \cdot f_{\text{Cauchy}}(r; a, b),$$

where $\boldsymbol{\Sigma}$ is the covariance matrix of the underlying distance measure and $\gamma_{\text{Cauchy}}(m, a, b, \boldsymbol{\Sigma})$ is the mentioned normalization factor,

$$\gamma_{\text{Cauchy}}(a, b, m, \boldsymbol{\Sigma}) = \frac{a\Gamma\left(\frac{m}{2}\right) \sin \frac{m\pi}{a}}{2\pi^{\frac{m}{2}+1} b^{\frac{m}{a}-1}} \cdot |\boldsymbol{\Sigma}|^{-\frac{1}{2}}.$$

A detailed derivation of this factor can be found in Section A.2.1 in the appendix. Γ denotes the so-called *generalized factorial*, which is defined as

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} \mathrm{d}t, \qquad x > 0.$$

Similarly, the generalized Gaussian function can be scaled to a unit integral if its (only) parameter $a > 1$. This normalized version is defined as

$$\gamma_{\text{Gauss}}(a, b, m, \boldsymbol{\Sigma}) \cdot f_{\text{Gauss}}(r; a, b),$$

where $\gamma_{\text{Gauss}}(m, a, b, \boldsymbol{\Sigma})$ is the normalization factor

$$\gamma_{\text{Gauss}}(a, b, m, \boldsymbol{\Sigma}) = \frac{a\Gamma\left(\frac{m}{2}\right)}{2^{\frac{m}{a}+1} \pi^{\frac{m}{2}} \Gamma\left(\frac{m}{a}\right)} \cdot |\boldsymbol{\Sigma}|^{-\frac{1}{2}}.$$

A detailed derivation can be found in Section A.2.1 in the appendix.

2.3 Prototype Properties

The cluster prototypes I consider in this thesis are essentially point prototypes (points in the data space), sometimes enhanced by size, shape, and weight parameters. Formally, I describe a set of clusters by the parameters $\mathbf{C} = \{\mathbf{c}_i \mid 1 \leq i \leq c\}$ (c is the number of clusters of the model) where each

$$\mathbf{c}_i = (\vec{\mu}_i, \mathbf{\Sigma}_i, \varrho_i), \qquad 1 \leq i \leq c,$$

specifies a cluster prototype. $\vec{\mu}_i$ is an m-dimensional real vector that states the **location** of the cluster and which is usually called the **cluster center**. $\mathbf{\Sigma}_i$ is an $m \times m$ real, symmetric[3], and positive definite[4] matrix describing the cluster's **size** and **shape**. It is usually called the **covariance matrix** of the cluster, since it is often computed in a similar way as the covariance matrix of a data set. (Details about how this matrix can be split into two factors, so that size and shape parameters can be distinguished, are given below.) Finally, ϱ_i is a real number that is the **weight** of the cluster. The ϱ_i are often (though not always) required to satisfy the constraint $\sum_{i=1}^{c} \varrho_i = 1$, so that they can be interpreted as (prior) probabilities of the clusters.

The similarity of a data point $\vec{x}$ to a cluster prototype $\mathbf{c}_i$ is described by a **membership function**, into which these and other parameters enter (like the two radial function parameters a and b discussed in the preceding section as well as the dimension m of the data space):

$$u_i^\circ(\vec{x}) = u^\circ(\vec{x}; \mathbf{c}_i) = \gamma(a, b, m, \mathbf{\Sigma}_i) \cdot f_{\text{radial}}(d(\vec{x}, \vec{\mu}_i; \mathbf{\Sigma}_i); a, b).$$

Here d is a **distance measure** as it was discussed in Section 2.1 and f_{radial} is a **radial function** as it was considered in Section 2.2. This radial function and its parameters a and b are the same for all clusters. γ is an optional **normalization factor** for the radial function. It was also discussed in Section 2.2 w.r.t. the interpretation of the radial function as a probability density and then scales the radial function so that its integral over the data space becomes 1. In all other cases this factor is simply set to 1.

Depending on the clustering or classification model, the **membership degrees** may be normalized in some way over all clusters. Such **normalization modes** are discussed in Section 2.4, in which I also consider the weight ϱ_i of a cluster. It does not appear in the above formula, as it has no proper meaning for cluster prototypes considered in isolation.

[3] An $m \times m$ matrix $\mathbf{A}$ is called *symmetric* if it equals its transpose, i.e. $\mathbf{A} = \mathbf{A}^\top$.

[4] An $m \times m$ matrix $\mathbf{A}$ is called *positive definite* iff for all m-dimensional vectors $\vec{v} \neq \vec{0}$, it is $\vec{v}^\top \mathbf{A} \vec{v} > 0$.

While the cluster center $\vec{\mu}_i = (\mu_{i1}, \ldots, \mu_{im})$, which is a simple point in the data space, needs no further explanation, it is useful to inspect a cluster's covariance matrix a little more closely. I write such a matrix as

$$\boldsymbol{\Sigma}_i = \begin{pmatrix} \sigma_{i,11} & \cdots & \sigma_{i,1m} \\ \vdots & \ddots & \vdots \\ \sigma_{i,m1} & \cdots & \sigma_{i,mm} \end{pmatrix}.$$

That is, $\sigma_{i,jk}$ refers to the element in the j-th row and k-th column of the covariance matrix of cluster $\mathbf{c}_i$. Similar to common usage in statistics, the diagonal elements $\sigma_{i,jj}$ may alternatively be written as $\sigma^2_{i,j}$, with the square replacing the double second index.

To distinguish between the size and the shape of a cluster, I exploit the fact that (the square root of) the determinant $|\boldsymbol{\Sigma}_i|$ can be seen as a measure of the size of a unit (hyper-)sphere (i.e., the set of points for which the distance from a given point is no greater than 1) w.r.t. the Mahalanobis distance that is parameterized with the covariance matrix $\boldsymbol{\Sigma}_i$. More precisely: $\sqrt{|\boldsymbol{\Sigma}_i|}$ is proportional[5] to the described (hyper-)ellipsoid's (hyper-)volume (see Section A.4 in the appendix, especially page 283, for a detailed explanation of this fact). As a consequence we can distinguish between the size and the shape of a cluster by writing its covariance matrix as

$$\boldsymbol{\Sigma}_i = \sigma_i^2 \mathbf{S}_i,$$

where $\sigma_i = \sqrt[2m]{|\boldsymbol{\Sigma}_i|}$ and $\mathbf{S}_i$ is a symmetric and positive definite matrix satisfying $|\mathbf{S}_i| = 1$. Since $\mathbf{S}_i$ has a unit determinant (and thus unit size in basically all possible interpretations of the meaning of the term *size*, see below), it can be seen as a measure of only the shape of the cluster, while its size is captured in the other factor. This view is exploited in the shape and size regularization methods studied in Sections 6.1.2 and 6.1.3.

Note that it is to some degree a matter of taste how we measure the *size* of a cluster. From the above considerations it is plausible that it should be of the form σ_i^κ, but good arguments can be put forward for several different choices of κ. The choice $\kappa = m$, for example, where m is the number of dimensions of the data space, yields $\sigma_i^m = \sqrt{|\boldsymbol{\Sigma}_i|}$. This is directly the (hyper-)volume of the (hyper-)ellipsoid that is described by the Mahalanobis distance parameterized with $\boldsymbol{\Sigma}_i$ (see above).

[5] The proportionality factor is $\gamma = \frac{\pi^{\frac{m}{2}} r^m}{\Gamma(\frac{m}{2}+1)}$, which is the size of a unit (hyper-)sphere in m-dimensional space, measured with the Euclidean distance (see Section A.4 in the appendix, especially page 283, for details).

However, the choice $\kappa = m$ has the disadvantage that it depends on the number m of dimensions of the data space. Since it is often convenient to remove this dependence, $\kappa = 1$ and $\kappa = 2$ are favorable alternatives. The former can be seen as an equivalent "isotropic"[6] standard deviation, because a (hyper-)sphere with radius σ_i w.r.t. the Euclidean distance has the same (hyper-)volume as the (hyper-)ellipsoid that is described by the Mahalanobis distance parameterized with $\mathbf{\Sigma}_i$. Similarly, $\sigma_i^2 = |\mathbf{\Sigma}_i|$ (i.e., the choice $\kappa = 2$) may be seen as an equivalent "isotropic" variance.

Up to now I considered only general covariance matrices, which can describe (hyper-)ellipsoids in arbitrary orientation. However, in applications it is often advantageous to restrict the cluster-specific covariance matrices to diagonal matrices, i.e., to require

$$\mathbf{\Sigma}_i = \mathrm{diag}(\sigma_{i,1}^2, \ldots, \sigma_{i,m}^2).$$

The reason is that such covariance matrices describe (as already studied in Section 2.1) (hyper-)ellipsoids the major axes of which are parallel to the axes of the coordinate system. As a consequence, they are much easier to interpret, since humans usually have considerable problems to imagine (hyper-)ellipsoids in general orientation, especially in high-dimensional spaces. A related argument was put forward in [Klawonn and Kruse 1997], in which a fuzzy clustering result was used to derive fuzzy rules from data. In this case axes parallel (hyper-)ellipsoids minimize the information loss resulting from the transformation into fuzzy rules.

2.4 Normalization Modes

A clustering result or classification model consists not only of one, but of several clusters. Therefore the question arises how to assign a data point to a cluster, either in a crisp way, so that each data point is assigned to exactly one cluster, or to several clusters using degrees of membership. The basis for this assignment is, of course, the set of membership degrees of a data point $\vec{x}$ to the different clusters i, $1 \leq i \leq c$, of the model, as they were defined in the preceding section, i.e., $u_i^\circ(\vec{x}) = u^\circ(\vec{x}; \mathbf{c}_i)$, $1 \leq i \leq c$. In addition, we have to respect the weights ϱ_i of the clusters, which may express a preference for the assignment, thus modifying the relation of the membership degrees. Finally, it can be useful to transform the membership

[6] From the Greek "iso" — equal, same and "tropos" — direction; "isotropic" means that all directions have the same properties, which holds for a sphere seen from its center, but not for a general ellipsoid, the extensions of which differ depending on the direction.

degrees in order to ensure certain properties of the resulting assignment, which is particularly important if degrees of membership are used. This whole process I call the **normalization** of the membership degrees.

In order to simplify the explanation, I take the normalization procedure apart into three steps: In the first step, **weighting**, the membership degree of a data point $\vec{x}$ to the i-th cluster is multiplied with the weight of the cluster, i.e. $\forall i; 1 \leq i \leq c$:

$$u_i^*(\vec{x}) = \varrho_i \cdot u_i^\circ(\vec{x}).$$

The second step consists in the following **transformation** $\forall i; 1 \leq i \leq c$:

$$u_i^\bullet(\vec{x}; \alpha, \beta) = \max\left\{0, (u_i^*(\vec{x}))^\alpha - \beta \max_{k=1}^{c}(u_k^*(\vec{x}))^\alpha\right\}, \quad \alpha > 0, 0 \leq \beta < 1.$$

This transformation is inspired by the approach presented in [Klawonn and Höppner 2003] and can be seen as a heuristic simplification of that approach. The idea underlying it is as follows: both the Cauchy function and the Gaussian function do not vanish, not even for very large arguments. As a consequence any data point has a non-vanishing degree of membership to a cluster, regardless of how far away that cluster is. The subsequent normalization described below (except when it consists in a hard assignment) does not change this. Under these conditions certain clustering algorithms, for example the popular fuzzy c-means algorithm (see Sections 3.1 and 5.2), can show an undesirable behavior: they encounter serious problems if they are to recognize clusters that differ considerably in how densely they are populated. The higher number of points in a densely populated region, even if they all have a low degree of membership, can "drag" a cluster prototype away from a less densely populated region, simply because the sum of a large number of small values can exceed the sum of a small number of large values. This can leave the less densely populated region uncovered.

A solution to this problem consists in making it possible that a data point has a vanishing degree of membership to a cluster, provided that cluster is far away [Klawonn and Höppner 2003]. At first sight, this may look like a contradiction to my rejection of radial functions that have a limited region of non-vanishing values (like splines or trapezoidal functions) in Section 2.2. Such functions automatically vanish if only the data point is far enough away from a cluster. However, the effect of the above transformation is different. There is no fixed distance at which the degree of membership to a cluster vanishes nor any limit for this distance. It rather depends on how good the "best" cluster (i.e., the one yielding the highest (weighted) degree of membership) is, since the value subtracted from the degree of membership depends on the degree of membership to this cluster.

Intuitively, the above transformation can be interpreted as follows: The degree of membership to the "best" cluster (the one yielding the highest degree of membership) never vanishes. Whether the data point has a non-vanishing degree of membership to a second cluster depends on the ratio of the (untransformed) degrees of memberships to this second cluster and to the best cluster. This ratio must exceed the value of the parameter β, otherwise the degree of membership to the second cluster will be zero.

To illustrate the effect of the value of β in this transformation, Figures 2.8 to 2.10 show the degrees of membership for a one-dimensional domain with two clusters, located at -0.5 (black curve) and 0.5 (grey curve), for a Euclidean distance and the inverse squared distance (Figure 2.8), the standard Cauchy function (Figure 2.9), and the standard Gaussian function (Figure 2.10). Since the transformation is the identity for the left diagrams, it becomes clearly visible how a positive value for β reduces the membership to one cluster to zero in certain regions. Thus it has the desired effect.

Note that the effect of the parameter α can often be achieved by other means, in particular by the cluster size or the choice of the radial function. For example, if the generalized Gaussian function is used with the cluster weights ϱ_i fixed to 1 and $\beta = 0$, we have

$$u_i^*(\vec{x}) = u_i^\circ(\vec{x}) = \exp\left(-\frac{1}{2}(d(\vec{x}, \vec{\mu}_i; \mathbf{\Sigma}_i))^a\right)$$

and therefore

$$u_i^*(\vec{x}) = u_i^\circ(\vec{x}) = \left(\exp\left(-\frac{1}{2}(d(\vec{x}, \vec{\mu}_i; \mathbf{\Sigma}_i))^a\right)\right)^\alpha = \exp\left(-\frac{\alpha}{2}(d(\vec{x}, \vec{\mu}_i; \mathbf{\Sigma}_i))^a\right).$$

In this form it is easy to see that the parameter α could be incorporated into the covariance matrix parameterizing the Mahalanobis distance, thus changing the size of the cluster (cf. Section 2.3).

Similarly, if the unnormalized generalized Cauchy function with $b = 0$ is used, and the cluster weights ϱ_i are fixed to 1, we have

$$u_i^*(\vec{x}) = u_i^\circ(\vec{x}) = \frac{1}{(\vec{x} - \vec{\mu}_i)^a}$$

and therefore

$$u_i^\bullet(\vec{x}; \alpha, 0) = \left(\frac{1}{(\vec{x} - \vec{\mu}_i)^a}\right)^\alpha = \frac{1}{(\vec{x} - \vec{\mu}_i)^{a\alpha}}.$$

This is equivalent to changing the parameter a of the generalized Cauchy function to $a' = a\alpha$. A difference that cannot be reproduced by changing

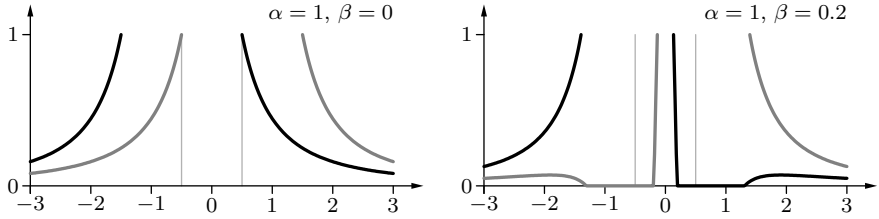

Figure 2.8: Transformed membership degrees for the generalized Cauchy function ($a = 2$, $b = 0$) for two cluster centers at -0.5 and 0.5.

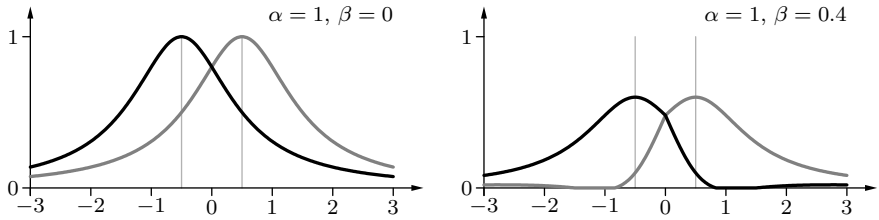

Figure 2.9: Transformed membership degrees for the generalized Cauchy function ($a = 2$, $b = 1$) for two cluster centers at -0.5 and 0.5.

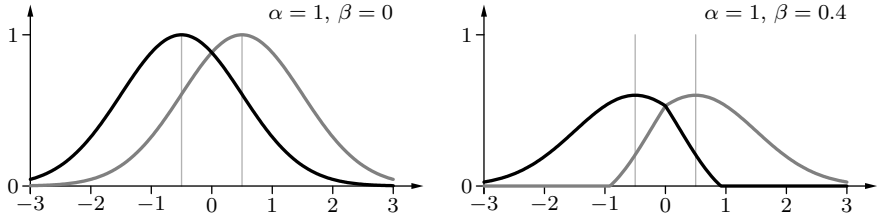

Figure 2.10: Transformed membership degrees for the generalized Gaussian function ($a = 2$) for two cluster centers at -0.5 and 0.5.

the cluster size or the parameters of the radial function results only if the generalized Cauchy function is used with $b > 0$.

However, the main reason for introducing the parameter α is that in fuzzy clustering an exponent like α enters the update formulae, where it is coupled to another exponent that controls how the case weight for the estimation of the cluster parameters is computed from the membership degrees. This issue is discussed in more detail in Chapter 5.

Note that choosing $\alpha = 1$ and $\beta = 0$ in the transformation rule leaves the (weighted) membership degrees unchanged, i.e., $u_i^\bullet(\vec{x}; 1, 0) = u_i^*(\vec{x})$. This is actually the standard choice for most algorithms.

Note also that a similar effect can be achieved using the transformation

$$u_i^\bullet(\vec{x}; \alpha, \beta) = \begin{cases} (u_i^*(\vec{x}))^\alpha, & \text{if } (u_i^*(\vec{x}))^\alpha \geq \beta \max_{k=1}^{c} (u_k^*(\vec{x}))^\alpha, \\ 0, & \text{otherwise,} \end{cases}$$

with $\alpha > 0$, and $0 \leq \beta < 1$. The advantage of this alternative transformation is that it does not change the ratios of the (weighted) membership degrees. However, the former transformation is closer to the inspiring rule by [Klawonn and Höppner 2003]. This rule was derived from a generalization of the fuzzifier in fuzzy clustering and is defined as follows: sort the membership degrees $u_i^*(\vec{x})$ descendingly, i.e., determine a mapping function $\varsigma : \{1, \ldots, c\} \to \{1, \ldots, c\}$ for the cluster indices such that

$$\forall i; 1 \leq i < c : \quad u_{\varsigma(i)}^*(\vec{x}) \geq u_{\varsigma(i+1)}^*(\vec{x}).$$

Then define the quantity $\check{c}(\vec{x})$, which measures the number of clusters to which the data point $\vec{x}$ has a non-vanishing membership degree, as

$$\check{c}(\vec{x}) = \max \left\{ k \;\middle|\; u_{\varsigma(k)}^*(\vec{x}) > \frac{\beta}{1 + \beta(k-1)} \sum_{i=1}^{k} u_{\varsigma(i)}^*(\vec{x}) \right\}.$$

Based on this quantity the transformed degrees of membership are then (note that the parameter α does not appear in this formula)

$$\begin{aligned} u_i^\bullet(\vec{x}; \alpha, \beta) &= 2 \left((1 + \beta(\check{c}(\vec{x}) - 1)) \frac{u_i^*(\vec{x})}{U} - \beta \right) \\ &= \frac{2(1 + \beta(\check{c}(\vec{x}) - 1))}{U} \left(u_i^*(\vec{x}) - \frac{\beta}{1 + \beta(\check{c}(\vec{x}) - 1)} U \right), \end{aligned}$$

where

$$U = \sum_{k:\varsigma(k) \leq \check{c}(\vec{x})} u_{\varsigma(k)}^*(\vec{x}).$$

The relation to the first transformation studied above can be seen if one normalizes the membership degrees to sum 1 over all clusters. Then the following transformation is effectively equivalent:

$$u_i^\bullet(\vec{x}; \alpha, \beta) = \max \left\{ 0, u_i^*(\vec{x}) - \frac{\beta}{1 + \beta(\check{c}(\vec{x}) - 1)} \sum_{k:\varsigma(k) \leq \check{c}(\vec{x})} u_{\varsigma(k)}^*(\vec{x}) \right\}.$$

An obvious disadvantage of this approach is that it does not allow for such nice and simple interpretations as the two transformations studied before. Its advantage, however, is that it can be justified very cleanly in the area of fuzzy clustering by a derivation from a specific objective function. This derivation is discussed in more detail in Sections 3.1 and 5.2.

After the membership degrees have been weighted and transformed in the first two steps, the final third step is the actual **normalization**, which ensures certain properties of the *set* of membership degrees as a whole, rather than properties of individual membership degrees. Depending on the general approach and the algorithm, the following choices have been tried:

- **hard/crisp assignment**

$$u_i^{(\text{hard})}(\vec{x}) = \begin{cases} 1, & \text{if } i = \operatorname{argmax}_{k=1}^{c} u_k^{\bullet}(\vec{x}), \\ 0, & \text{otherwise.} \end{cases}$$

- **normalization to sum 1**

$$u_i^{(\text{sum1})}(\vec{x}) = \frac{u_i^{\bullet}(\vec{x})}{\sum_{k=1}^{c} u_k^{\bullet}(\vec{x})}$$

- **normalization to maximum 1**

$$u_i^{(\text{max1})}(\vec{x}) = \frac{u_i^{\bullet}(\vec{x})}{\max_{k=1}^{c} u_k^{\bullet}(\vec{x})}$$

- **no normalization**

$$u_i^{(\text{raw})}(\vec{x}) = u_i^{\bullet}(\vec{x})$$

A hard assignment to clusters can lead to ties, i.e., situations in which two or more clusters share the highest degree of membership. Such ties are usually broken arbitrarily, i.e., the data point is assigned to a cluster that is randomly chosen from those to which it has the highest degree of membership. Note that a hard assignment can (almost) be mimicked by letting the parameter β of the membership transformation rule go to one and using a normalization to sum 1 or maximum 1.

Unfortunately, none of these normalization modes is perfect: all have their specific advantages and disadvantages. A **hard assignment**, which is used in many classical clustering and classification approaches like, for example, hard c-means clustering and (supervised and unsupervised) learning vector quantization, suffers from the fact that often data points cannot be

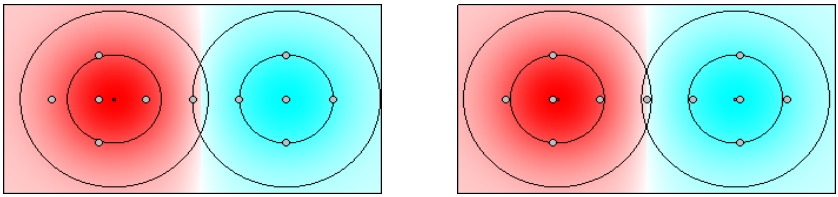

Figure 2.11: Results of the hard c-means (left) and the fuzzy c-means algorithm (right) for a symmetric data set. Fuzzy membership degrees allow for assigning the point in the middle to both clusters with a degree of 0.5.

assigned unambiguously to one (and only one) cluster. In such situations a crisp decision which cluster such data points belong to can often be made only with a certain degree of arbitrariness, enforcing sharp boundaries where there is actually ambiguity. An extreme example is the highly symmetric data set shown in Figure 2.11, which is to be divided into two clusters. The problem obviously resides with the data point in the middle. Assigning it uniquely to one cluster, as shown on the left (which is the result of hard c-means clustering[7]) breaks the symmetry unintuitively.

For such situations a normalization to sum 1 or maximum 1 is much more appropriate, as it allows us to assign the point in the middle equally to both clusters. This is shown on the right in Figure 2.11 (result of fuzzy c-means clustering with a normalization to sum 1), thus maintaining in their solution the symmetry of the data set. However, symmetric data distributions are not the only case where graded assignments are advantageous. Whenever the clusters are not clearly separated, but rather form regions of higher data point density, which are separated by regions of lower data point density, a sharp boundary is not intuitive and should be replaced by a soft transition between the clusters, described by degrees of membership.

A normalization to sum 1 is particularly meaningful if the set of clusters describes a so-called **mixture model** [Everitt and Hand 1981] of the data. In such a mixture model the probability density function of the process that generated the data is assumed to be a mixture of a certain number of probability density functions, each of which is described by one cluster.

[7]Details about the clustering algorithms with which the results shown in Figure 2.11 were obtained (hard and fuzzy c-means clustering) are given in later chapters. Here they only serve the purpose of demonstrating the effects of a hard assignment versus membership degrees normalized to sum 1 or maximum 1.

Formally, we consider the **likelihood** of the data point in this case, that is,

$$f_{\vec{X}}(\vec{x}; \mathbf{C}) = \sum_{y=1}^{c} f_{\vec{X},Y}(\vec{x}, y; \mathbf{C}) = \sum_{y=1}^{c} p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}|y; \mathbf{C}).$$

$\vec{X}$ is a random vector that has the points of the data space as possible values (i.e., $\mathrm{dom}(\vec{X}) = \mathbb{R}^m$) and Y is a random variable that has the cluster indices as possible values (i.e., $\mathrm{dom}(Y) = \{1, \ldots, c\}$). $p_Y(y; \mathbf{C})$ is the probability that the data point belongs to (was generated by) the y-th component of the mixture. This probability is stated by the cluster weights ϱ_y. $f_{\vec{X}|Y}(\vec{x}|y; \mathbf{C})$ is the conditional probability density function of the data points given the cluster. It is modeled by the clusters' membership functions u_y°. That is,

$$p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}|y; \mathbf{C}) = \varrho_y \cdot u_y^\circ(\vec{x}) = u_y^*(\vec{x}).$$

Of course, to be able to serve as a conditional probability function, the membership function u_y° has to be normalized to integral 1 over the data space (cf. page 19 in Section 2.2 and page 20 in Section 2.3 for details).

If the set of clusters describes a mixture model, the normalization of the membership degrees to sum 1 is equivalent to computing the **posterior probabilities** of the clusters given the data point. This follows, by a simple application of Bayes' rule and the rule of total probability:

$$\begin{aligned} p_{Y|\vec{X}}(y|\vec{x}) &= \frac{p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}|y; \mathbf{C})}{f_{\vec{X}}(\vec{x}; \mathbf{C})} \\ &= \frac{p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}|y; \mathbf{C})}{\sum_{i=1}^{c} p_Y(i; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}|i; \mathbf{C})} \\ &= \frac{\varrho_y \cdot u_y^\circ(\vec{x})}{\sum_{i=1}^{c} \varrho_i \cdot u_i^\circ(\vec{x})} = \frac{u_y^*(\vec{x})}{\sum_{i=1}^{c} u_i^*(\vec{x})} = u_y^{(\mathrm{sum1})}(\vec{x}). \end{aligned}$$

Of course, the last step follows only if no membership transformation is used, i.e., if $\alpha = 1$ and $\beta = 0$. However, this is no real restriction, as such a transformation is usually not necessary with the most common radial function used in mixture models, that is, the Gaussian function.

A normalization to sum 1 also has its disadvantages, though, as can be seen from Figures 2.12 to 2.14, which show the normalized versions of the (transformed) degrees of membership depicted in Figures 2.8 to 2.10, respectively. With the exception of the inverse squared distance (Figure 2.12) the maximum of the membership degrees does no longer coincide with the

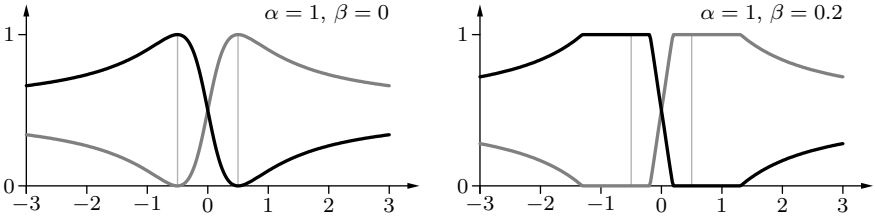

Figure 2.12: Membership degrees normalized to sum 1 for the generalized Cauchy function ($a = 2$, $b = 0$) for two cluster centers at -0.5 and 0.5.

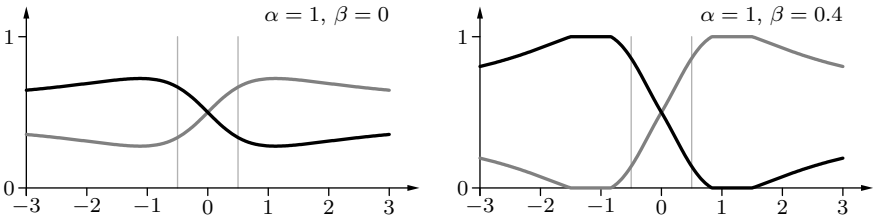

Figure 2.13: Membership degrees normalized to sum 1 for the generalized Cauchy function ($a = 2$, $b = 1$) for two cluster centers at -0.5 and 0.5.

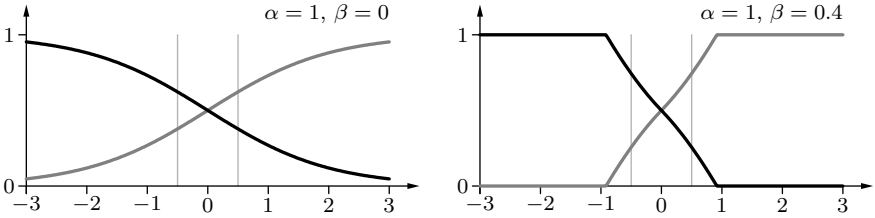

Figure 2.14: Membership degrees normalized to sum 1 for the generalized Gaussian function ($a = 2$) for two cluster centers at -0.5 and 0.5.

positions of the cluster centers (-0.5 and 0.5, as indicated by the thin vertical lines). If a positive value for β annuls, in some region, the degree of membership to the cluster that is further away (thus, for two clusters, forcing the membership degree to the other cluster to be one), the cluster centers do not lie in the regions with membership degree 1 (again with the exception of the inverse squared distance, Figure 2.12). Such a behavior is definitely not what we expect intuitively of degrees of membership.

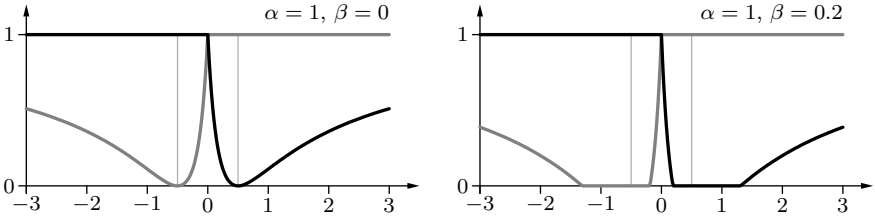

Figure 2.15: Normalization to maximum 1 for the generalized Cauchy function ($a = 2$, $b = 0$) for two cluster centers at -0.5 and 0.5.

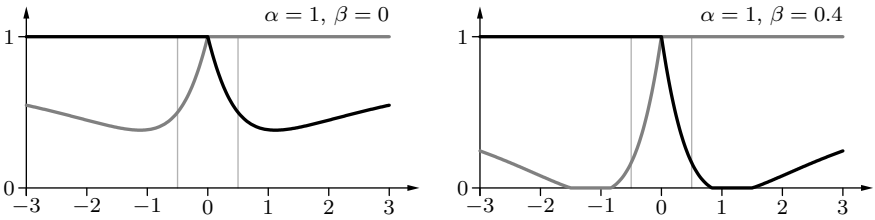

Figure 2.16: Normalization to maximum 1 for the generalized Cauchy function ($a = 2$, $b = 1$) for two cluster centers at -0.5 and 0.5.

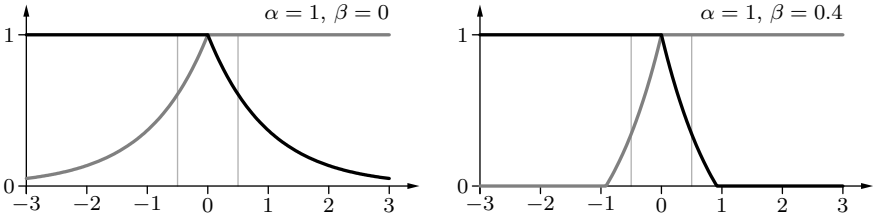

Figure 2.17: Normalization to maximum 1 for the generalized Gaussian function ($a = 2$) for two cluster centers at -0.5 and 0.5.

Even worse, if we consider the degrees of membership to one cluster, once we move beyond the position of the other, they *increase* again, with the exception of the Gaussian function (2.14). This behavior is an effect of the normalization, since it is not present in the figures showing the unnormalized degrees of membership (2.8 to 2.10). It is even less intuitive than the fact that the maxima do not coincide with the cluster centers. The same effect can be observed for a normalization to maximum 1 (Figures 2.15 and 2.16).

For a formal analysis of this effect, which will reveal a decisive difference between the generalized Cauchy function and the generalized Gaussian function already mentioned earlier, consider a one-dimensional space with two cluster centers at μ_1 and μ_2 that are Δ apart, i.e., $\mu_2 = \mu_1 + \Delta$, $\Delta > 0$. (Note that instead of assuming a one-dimensional space we may just as well consider a multi-dimensional space and restrict our considerations for the time being to the line through the two cluster centers.) For simplicity, assume that membership degrees are transformed with $\alpha = 1$ and $\beta = 0$. Then, regardless of the radial function used, its parameters, and whether the membership degrees are normalized to sum 1 or maximum 1, we have

$$\lim_{x \to \infty} \frac{u_1(x)}{u_2(x)} = \lim_{x \to \infty} \frac{u_1^*(x)}{u_2^*(x)},$$

since the normalization terms cancel each other. For a generalized Cauchy function, in which we fix the parameter a to 2 merely in order to simplify the computations, this leads to [Döring *et al.* 2005]

$$\begin{aligned}
\lim_{x \to \infty} \frac{u_1^*(x)}{u_2^*(x)} &= \lim_{x \to \infty} \frac{f_{\text{Cauchy}}(d(x, \mu_1; \mathbf{1}); 2, b)}{f_{\text{Cauchy}}(d(x, \mu_2; \mathbf{1}); 2, b)} \\
&= \lim_{x \to \infty} \frac{(x - \mu_2)^2 + b}{(x - \mu_1)^2 + b} \\
&= \lim_{x \to \infty} \frac{(x - \mu_1 - \Delta)^2 + b}{(x - \mu_1)^2 + b} \\
&= \lim_{x \to \infty} \frac{(x - \mu_1)^2 + b}{(x - \mu_1)^2 + b} + \lim_{x \to \infty} \frac{2\Delta(x - \mu_1) + \Delta^2}{(x - \mu_1)^2 + b} \\
&= 1 + 0 \;=\; 1.
\end{aligned}$$

Consequently we have for a normalization to sum 1 that

$$\lim_{x \to \infty} u_1(x) = \lim_{x \to \infty} u_2(x) = \frac{1}{2}.$$

Note that we get the same result for other values of a, only that the formal derivation is a little more complicated than for the standard case $a = 2$. Note also that the result may easily be transferred to directions in a multi-dimensional space that deviate from the line connecting the two clusters. Again this only leads to some technical complications, because then the value of Δ is not constant, but depends on the angle under which the two cluster centers are seen. Finally, note that the result may be generalized to c clusters, for which the membership degrees approach $\frac{1}{c}$ in the limit.

The limit behavior of the generalized Gaussian function, on the other hand, is completely different. Again setting the parameter $a = 2$ to simplify the computations, we obtain here [Döring *et al.* 2005]

$$\begin{aligned}
\lim_{x\to\infty} \frac{u_1(x)}{u_2(x)} &= \lim_{x\to\infty} \frac{f_{\text{Gauss}}(d(x,\mu_1;\mathbf{1});2,0)}{f_{\text{Gauss}}(d(x,\mu_2;\mathbf{1});2,0)} \\
&= \lim_{x\to\infty} \frac{\exp\left(-\frac{1}{2}(x-\mu_1)^2\right)}{\exp\left(-\frac{1}{2}(x-\mu_2)^2\right)} \\
&= \lim_{x\to\infty} \exp\left(-\frac{1}{2}\left((x-\mu_1)^2 - (x-\mu_1-\Delta)^2\right)\right) \\
&= \lim_{x\to\infty} \exp\left(-\frac{1}{2}\left(2x\Delta - 2\mu_1\Delta - \Delta^2\right)\right) \\
&= \lim_{x\to\infty} \exp\left(-x\Delta - \mu_1\Delta - \frac{\Delta^2}{2}\right) \quad = \quad 0.
\end{aligned}$$

Consequently we have for a normalization to sum 1 or maximum 1 that

$$\lim_{x\to\infty} u_1(x) = 0 \qquad \text{and} \qquad \lim_{x\to\infty} u_2(x) = 1.$$

For $x \to -\infty$ the relation is reversed, so that we have

$$\lim_{x\to-\infty} u_1(x) = 1 \qquad \text{and} \qquad \lim_{x\to-\infty} u_2(x) = 0.$$

That is, in the limit a data point will be assigned exclusively to the closer cluster. Note that, just as for the generalized Cauchy function, the above derivation does not really depend on the choice $a = 2$. We only have to ensure that $a > 1$ (for $a = 1$ we obviously get a constant ratio, which depends on the distance Δ of the two clusters). In a similar way, we can generalize the result to arbitrary directions in a multi-dimensional domain and to several clusters. That is, in the limit a data point is assigned with a degree of 1 to the closest cluster and with a degree of 0 to all others.

Since I need this observation later, it should be noted that the same effect can be achieved if one lets the determinant of the size and shape parameter $\mathbf{\Sigma}_i$ go to zero, because this "increases", through the scaling matrix of the eigenvalue decomposition of $\mathbf{\Sigma}$, the effective distance of a data point to a cluster center (cf. Section 2.1). In other words, if the size of the clusters goes to zero, the data points are also assigned exclusively to the closest cluster. This effect can be used to represent hard assignment algorithms (like classical learning vector quantization) as limiting cases of Gaussian radial function based algorithms that use membership degrees.

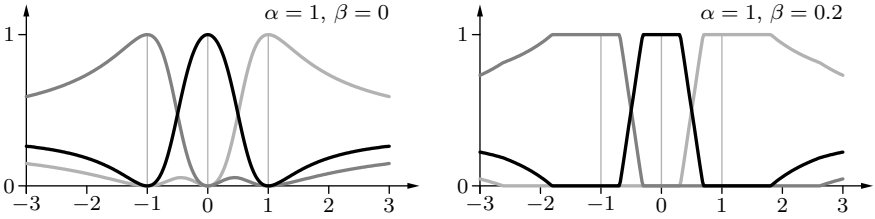

Figure 2.18: Membership degrees normalized to sum 1 for the generalized Cauchy function ($a = 2$, $b = 0$) for three cluster centers at -1, 0, and 1.

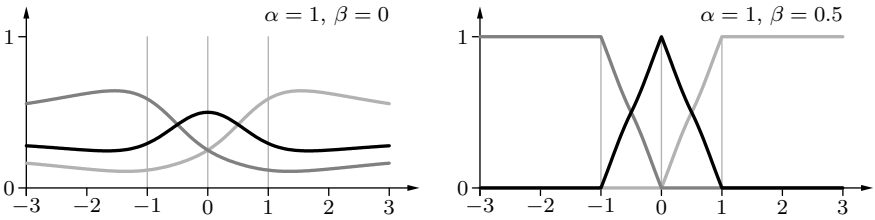

Figure 2.19: Membership degrees normalized to sum 1 for the generalized Cauchy function ($a = 2$, $b = 1$) for three cluster centers at -1, 0, and 1.

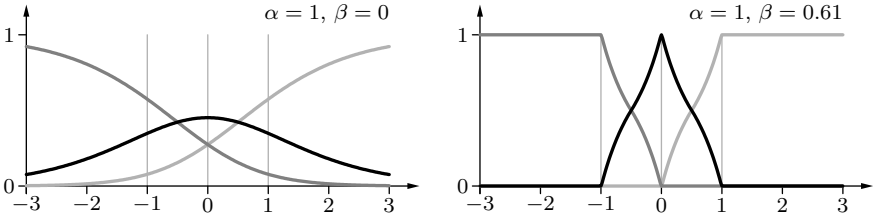

Figure 2.20: Membership degrees normalized to sum 1 for the generalized Gaussian function ($a = 2$) for three cluster centers at -1, 0, and 1.

As a further illustration of the effects of normalization, Figures 2.18 to 2.20 show, for a one-dimensional domain, the degrees of membership for three clusters at 0 (black curve), -1 and 1 (grey curves). The left diagrams always show the situation for an identity transformation, i.e., for the standard case $\alpha = 1$ and $\beta = 0$. The right diagrams, on the other hand, use special values for the parameter β of the membership transformation to achieve a specific effect, like that the membership degrees to one cluster

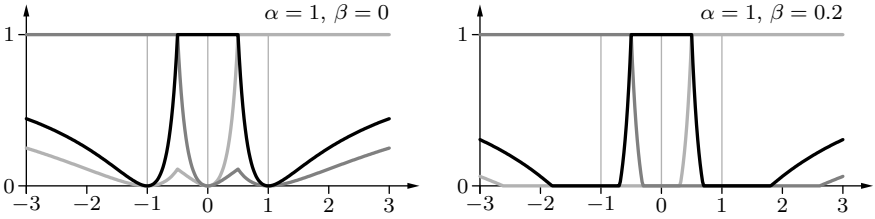

Figure 2.21: Normalization to maximum 1 for the generalized Cauchy function ($a = 2$, $b = 0$) for three cluster centers at -1, 0, and 1.

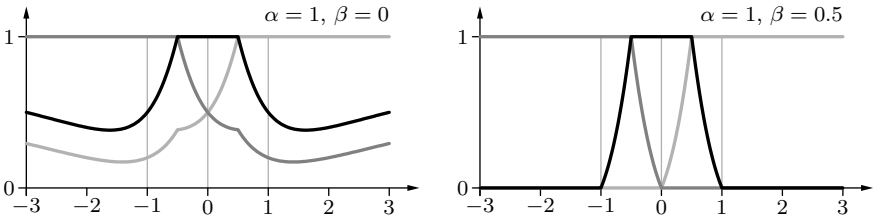

Figure 2.22: Normalization to maximum 1 for the generalized Cauchy function ($a = 2$, $b = 1$) for three cluster centers at -1, 0, and 1.

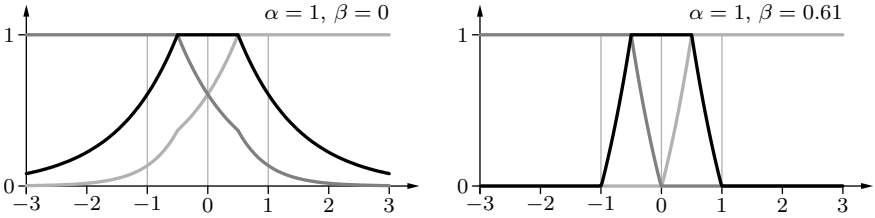

Figure 2.23: Normalization to maximum 1 for the generalized Gaussian function ($a = 2$) for three cluster centers at -1, 0, and 1.

vanish at the position of the other clusters. However, this effect depends on the relation between the distance of the cluster centers and the value of β, which is usually not adapted in a clustering algorithm. It actually also requires equidistant cluster centers. Hence these illustrations should not be seen as arguments for these special values of β, but only as illustrations what effects can result from different choices of β. Figures 2.21 to 2.23 show the corresponding situations for a normalization to maximum 1.

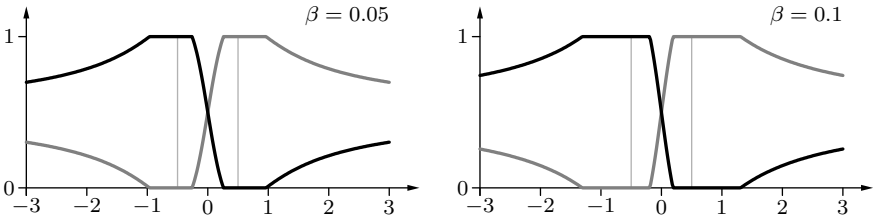

Figure 2.24: Special membership degrees normalization for the generalized Cauchy function ($a = 2$, $b = 0$) for two cluster centers at -0.5 and 0.5.

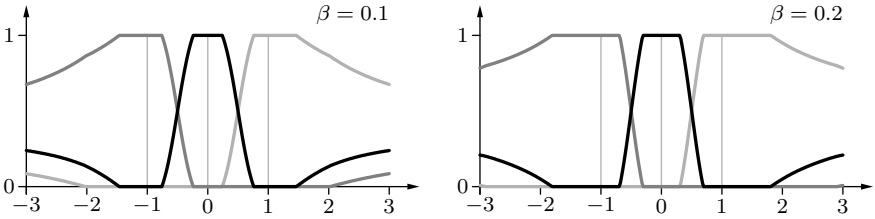

Figure 2.25: Special membership degrees normalization for the generalized Cauchy function ($a = 2$, $b = 0$) for two cluster centers at -1, 0, and 1.

Finally, Figures 2.24 and 2.25 show the effects of the membership transformation suggested by [Klawonn and Höppner 2003] (cf. page 26). They also give an idea of how it relates to the membership transformation

$$u_i^\bullet(\vec{x}; \alpha, \beta) = \max\left\{0, (u_i^*(\vec{x}))^\alpha - \beta \max_{k=1}^{c}(u_k^*(\vec{x}))^\alpha\right\}, \quad \alpha > 0, 0 \leq \beta < 1,$$

the effects of which for a normalization to sum 1 are shown in Figures 2.12 and 2.18. Obviously, the behavior is very similar, only that the values for β have to be chosen differently. It also reveals another disadvantage of the scheme by [Klawonn and Höppner 2003]: the values of β, with which certain effects can be achieved, depend more strongly on the number of clusters.

Due to the unintuitive effects of a normalization to sum 1, which were discussed above, using no normalization seems to be an attractive alternative. However, approaches that work with unnormalized membership degrees—like, for example, possibilistic fuzzy clustering [Krishnapuram and Keller 1993]—have their drawbacks too. Although the membership degrees are now easier to interpret, since they do not suffer from the abovemen-

tioned strange behavior, serious problems arise with the algorithm that is used to find the clusters. The reason is that doing without normalization decouples the clusters, so that each of them is optimized independently (in particular if the parameter β of the membership transformation is set to 0). As a consequence the clusters have a strong tendency to become identical. Other results are obtained only due to the initialization and the fact that the optimization can get stuck in a local minimum. This problem is discussed in more detail in Section 3.1, which—among other things—reveals the core problem of possibilistic fuzzy clustering.

In order to profit from the advantages of unnormalized membership degrees nevertheless, at least to some degree, some more flexible normalization schemes than those listed above have been tried, in particular in the area of learning vector quantization. An example is **restricted hard assignment**. In this approach a data point is assigned crisply if there is a cluster that is within some given maximum distance of the data point (or, equivalently, to which the data point has at least a given minimum degree of membership) [Acciani *et al.* 1999, Borgelt *et al.* 2004]. This distance is described by a (sometimes cluster-specific and adaptable) radius or radius factor. Otherwise the data point is assigned to all clusters without normalization.

The advantage of such an approach is that the clusters are still coupled through the hard assignment inside a certain radius. But since the influence is restricted to a region close to the cluster center, the transition from one cluster to another is made with degrees of membership that are easy to interpret. The problem of the sharp drop at the hard assignment radius may be avoided by measuring the distance for the unnormalized membership degrees from this radius (which is equivalent to an r-insensitive distance). The drawbacks of a restricted hard assignment are that the choice of the radius is difficult and automatic adaptation rules for this radius are more or less heuristic in nature and do not always have the desired effect.

2.5 Classification Schemes

Up to now I studied only how to assign a data point to one or several clusters. For classification purposes, however, we also need a method to assign it to a class, which is a different issue, because a class may consist of several clusters. Nevertheless the assignments to the clusters form, of course, the basis for the assignments to the classes.

There are basically two prototype-based classification schemes, that is, schemes of assigning data points to classes based on a cluster structure. The

first approach is known as the **nearest prototype classifier** or, slightly more generally, the **maximum membership classifier**. In this approach each cluster is endowed with a class label $\zeta_i \in \{1, \ldots, s\}$, $1 \leq i \leq c$ (several clusters may share the same label). Then the prototype that is closest to the data point (or yields the highest degree of membership) is used to classify a data point by labeling it with the associated class:

$$o = \zeta_k \qquad \text{where} \qquad k = \operatorname{argmax}_{i=1}^{c} u_i(\vec{x}).$$

The advantage of this approach is that it is very simple, needs only a minimal extension over a cluster structure (namely the assignment of the class labels ζ_i to the clusters), and is very easy to interpret. However, if a single cluster for one class is surrounded by several clusters of another, the region in which a data point is labeled with the class of this one cluster may be too big, since several clusters of the other class may claim that it belongs to them, so that their joint degree of membership should outweigh a higher degree of membership to the single cluster.

To handle such situations, the second scheme may be adopted, namely a classification based on a **linear function** of the degrees of membership. This approach is employed, for example, in so-called **radial basis function neural networks** [Rojas 1993, Haykin 1994, Anderson 1995, Nauck *et al.* 2003]. The idea is to set up a class membership function g_k, $1 \leq k \leq s$, for each of the classes (s is the number of classes) as a linear function

$$g_k(\vec{x}) = w_{k0} + \sum_{i=1}^{c} w_{ki}\, u_i(\vec{x})$$

of the cluster memberships. This classification scheme is usually applied without cluster membership transformation or normalization and without cluster weights, so that the $u_i(\vec{x})$ in the above function are equivalent to $u_i^\circ(\vec{x})$, $1 \leq i \leq c$. This shows that the missing cluster weights are actually incorporated into the coefficients w_{ki}. Alternatively, we may see the coefficients as providing a different cluster weight for each of the classes.

Note that each of the functions g_k can be seen as a classifier in its own right, which discriminates the class it is associated with from all other classes. Note also that by combining the functions g_k the above classification scheme may just as well be written in matrix-vector form as

$$\vec{g}(\vec{x}) = \mathbf{W}\vec{u}^\circ(\vec{x}),$$

where $\mathbf{W} = (w_{ki})_{1 \leq k \leq s, 0 \leq i \leq c}$, $\vec{u}^\circ(\vec{x}) = (1, u_1^\circ(\vec{x}), \ldots, u_c^\circ(\vec{x}))^\top$ (note the additional 1 in the first element) and $\vec{g}(\vec{x}) = (g_1(\vec{x}), \ldots, g_s(\vec{x}))^\top$.

A crisp class assignment may be obtained from the results (i.e., from the vector $\vec{g}(\vec{x})$ of class membership degrees) by assigning the data point to the class that yields the highest value. That is, the classifier predicts the class

$$o = \operatorname{argmax}_{k=1}^{s} g_k(\vec{x}).$$

A (formal) probability distribution over the classes may be computed by normalizing the vector $\vec{g}(\vec{x})$, so that the sum of its elements is 1. Note, however, that such a normalization to sum 1 gives meaningful probabilities only if the cluster structure describes a so-called **mixture model** [Everitt and Hand 1981] for each of the classes. In such a mixture model the probability density function for each class is assumed to be a mixture of a certain number of probability density functions, each of which is described by one cluster (cf. page 29 in Section 2.4 for the analogous case without classes). Formally, we consider in this case

$$\begin{aligned} f_{\vec{X}|Z}(\vec{x}|z;\mathbf{C}) &= \sum_{y \in I(z)} f_{\vec{X},Y|Z}(\vec{x},y|z;\mathbf{C}) \\ &= \sum_{y \in I(z)} p_{Y|Z}(y|z;\mathbf{C}) \cdot f_{\vec{X}|Y,Z}(\vec{x}|y,z;\mathbf{C}). \end{aligned}$$

$I(z) \subseteq \{1,\ldots,c\}$ is a set that contains the indices of those clusters that describe a mixture component of the z-th class. $\vec{X}$ is a random vector that has the points of the data space as possible values, i.e., $\mathrm{dom}(\vec{X}) = \mathbb{R}^m$, Y is a random variable that has indices of clusters as possible values, i.e., $\mathrm{dom}(Y) = \{1,\ldots,c\}$, and Z is a random variable that has indices of classes as possible values, i.e., $\mathrm{dom}(Z) = \{1,\ldots,s\}$. $p_{Y|Z}(y|z;\mathbf{C})$ is the probability that the data point belongs to (was generated by) the i-th component of the mixture given that it belongs to the k-th class. $f_{\vec{X}|Y,Z}(\vec{x}|y,z;\mathbf{C})$ is the conditional probability density function of the data points given the class and the cluster Note that in this model each cluster is usually associated with only one class (even though the above formula allows for using the same cluster for several classes), that is, the $I(z)$, $1 \leq z \leq s$, form a partition of the set $\{1,\ldots,c\}$ of cluster indices. (As a consequence, we may discard the random variable Z and its value z from this conditional probability function.) With the above probability density we define

$$\begin{aligned} g_z(\vec{x}) &= f_{\vec{X},Z}(\vec{x},z;\mathbf{C}) \\ &= p_Z(z) \cdot f_{\vec{X}|Z}(\vec{x}|z;\mathbf{C}) \\ &= p_Z(z) \sum_{y \in I(z)} p_{Y|Z}(y|z) \cdot f_{\vec{X}|Y,Z}(\vec{x}|y,z;\mathbf{C}), \end{aligned}$$

where $p_Z(z)$ is the prior probability of the z-th class. That is, we have a special case of a linear function classifier, in which the conditional probability density $f_{\vec{X}|Y,Z}(\vec{x}|y,z;\mathbf{C})$ of a data point given the cluster and the class is represented by the cluster membership functions u_y°. Of course, for this to be possible, these membership functions u_y° have to be normalized to integral 1 over the data space (cf. page 19 in Section 2.2 and page 20 in Section 2.3 for details). The coefficients w_{zy} of the linear classification functions are defined as $\forall z; 1 \leq z \leq s : \forall y; 1 \leq y \leq c :$

$$w_{z0} = 0 \qquad \text{and}$$
$$w_{zy} = p_{Y,Z}(y,z) = p_{Y|Z}(y|z) \cdot p_Z(z).$$

That is, there is no bias term and the coefficients combine the cluster membership degrees probabilistically. Consequently the coefficients satisfy

$$\textstyle\sum_{y \in I(z)} w_{zy} = \sum_{y \in I(z)} p_{Y,Z}(y,z) = p_Z(z) \qquad \text{and}$$
$$w_{zy} = p_{Y,Z}(y,z) > 0 \quad \rightarrow \quad \forall k; k \neq z : \; w_{ky} = p_{Y,Z}(y,k) = 0.$$

That is, together the coefficients for one class specify the prior probability of that class and each cluster is used in the mixture model for only one class.

If the cluster structure describes a mixture model, the normalization of the class assignments in $\vec{g}(\vec{x})$ to sum 1 is equivalent to computing the posterior probabilities of the classes given the data point, since

$$p_{Z|\vec{X}}(z|\vec{x}) = \frac{f_{\vec{X},Z}(\vec{x},z;\mathbf{C})}{f_{\vec{X}}(\vec{x};\mathbf{C})} = \frac{f_{\vec{X},Z}(\vec{x},z;\mathbf{C})}{\sum_{k=1}^{s} f_{\vec{X},Z}(\vec{x},k;\mathbf{C})} = \frac{g_z(\vec{x})}{\sum_{k=1}^{s} g_k(\vec{x})}.$$

This is a very well-known and well-founded classification scheme, which, for example, is used with one cluster per class in a (naïve or full) **Bayes classifier** (cf. Section 4.2 for more details).

Another special case of linear function classification is the following very simple and fairly common approach: the classes are assigned based on the sum of the (unnormalized) membership degrees to all clusters sharing the same class label. Of course, this is just a special linear function with a bias value of $w_{k0} = 0$ for all classes and weights w_{ki} that are 1 for all clusters belonging to the considered class and 0 for all other classes.

Note that, if a normalization of the class memberships to sum 1 is used, this classification scheme is also a special case of a mixture model classifier. The reason is that the normalization—due to which only the ratios of the different weights w_{ki} matter, but not their absolute values—makes it equivalent to a mixture model in which all $p_{Y,Z}(y,z;\mathbf{C})$ are equal.

2.6 Related Approaches

The cluster prototype model described in the preceding sections covers a large variety of approaches, ranging from probabilistic mixture models and fuzzy clustering to radial basis function neural networks and Bayes classifiers. However, there are also approaches, which are only partly covered or in special cases, but which are closely related nevertheless. Some of these approaches (I do not claim to be exhaustive) I discuss briefly in this section. A detailed explanation of these approaches is beyond the scope of this thesis, though, and thus I presuppose some basic knowledge about them. An interested reader should consult the referenced literature for details.

In the first place there are the so-called **kernel-based methods** with **support vector machines** as their currently most prominent representative [Vapnik 1995, Vapnik 1998, Cristianini and Shawe-Taylor 2000, Schölkopf and Smola 2002]. Approaches in this direction are captured formally by the cluster model studied here if the kernel function used is a radial function defined on a distance measure.[8] In this case the support vectors found by a support vector machine may formally resemble cluster centers and the classification is actually achieved with a linear function classifier as it was discussed in the preceding section.

Concerning the formal model there are, indeed, only some minor differences between these approaches. Among these are, in the first place, that only a single kernel function is used, which is parameterized in the same way for all support vectors, so that it describes a (hyper-)spherical region of influence[9], while the clusters, although they share the same radial function, can have different shapes and sizes. This is not a real restriction, though, since a lot of clustering algorithms also require the size and shape parameters of the clusters to be equal, mainly because this enhances robustness.

Secondly, the support vectors may not be arbitrary points in the data space, but are selected from the given data set. This restriction can turn out to be inconvenient, especially if data are sparse, but usually it does not lead to unsurmountable complications. If the data space allows for measuring distances, but not for computing mean vectors, one is actually forced to select the cluster centers from the given data set as, for instance, in *medoid clustering*[Kaufman and Rousseeuw 1990, Chu *et al.* 2001].

[8] Not all kernel functions that are commonly used are in this category. For example, Gaussian kernels are, but most polynomial kernels are not.

[9] Of course, support vector machines may rely on the Mahalanobis distance, but only on the version that is parameterized with the covariance matrix of the whole data set. There are no covariance matrices that are specific to a support vector.

Nevertheless, in most cases the relationship between cluster centers and support vectors is actually merely formal. The reason is that the objective of the cluster model, as it was developed in the preceding sections, is to capture the distribution of the data points with few prototypes. Certain limited deviations from an ideal representation may be accepted if this improves classification performance, but the foremost concern is to reflect the characteristics of the data distribution in the model. The idea of support vector machines, however, is to describe the decision boundary between classes by finding the data points that are closest to it.[10] (See Section 3.5 for more explanations about the objective of support vector machines.) It is rather an accident if they also capture the data distribution well. As a consequence, I almost neglect support vector machines in this thesis.

Another fairly closely related approach are **fuzzy rule-based systems** [Zadeh 1965, Dubois and Prade 1980, Pedrycz 1988, Böhme 1994, Kruse *et al.* 1994] and in particular **neuro-fuzzy systems** [Nauck *et al.* 2003], because they may be seen as a special case or a modification of radial basis function neural networks. The antecedent of each fuzzy rule, which consists of a conjunction of fuzzy sets on the individual data dimensions, describes a softly bounded region in the data space and thus can capture a subset of the data points that are similar to each other. The consequent of the rule may assign a class to them in a crisp way or may also specify a fuzzy set.

However, not all fuzzy systems can be interpreted in this way. Limitations include, in the first place, that the fuzzy sets used must be symmetric, because asymmetric membership functions cannot be mapped to a radial function based on a distance measure. The reason is that even though the Mahalanobis distance allows for different shapes and sizes, it always treats directions alike that are opposite to each other. This fundamental symmetry is inherent in the model developed in the preceding sections and cannot be removed without having to accept disadvantages w.r.t. interpretability as well as the update mechanisms of the learning algorithms.

Secondly, the fuzzy sets may not have different shape. Differing extensions in the different data dimensions can be handled by a scaling matrix for the distance measure and thus need not be ruled out, but it must not be that one dimension uses triangular fuzzy sets, another bell-shaped ones. The reason is, of course, that in the cluster model there is only one radial function, which cannot be changed depending on the data dimension. If,

[10] Actually closest in a transformation of the data points to some high-dimensional space, which is implicitly brought about by the kernel function, but with radial function kernels this usually also means closest in the original space. In this case the transformation mainly has the effect of allowing complex decision boundaries.

however, the input domain has just a single dimension, all fuzzy rule-based systems, provided they use symmetric fuzzy sets having the same shape, can be mapped to the cluster model studied here. This is obvious, because we only have to choose the (possibly scaled) Euclidean distance and a radial function that matches the membership function of the fuzzy sets.

On the other hand, if we have multiple input dimensions, it depends on the t-norm (cf. page 222 in Section 7.2.2 for a definition) by which the conjunction is modeled as well as the way in which the implication is evaluated, whether the antecedent of a fuzzy rule can actually be seen as a description of a cluster. This is definitely not the case for fuzzy systems based on *relational equations*, since they use the Gödel implication and thus specify constraints rather than clusters.

A fuzzy system that can be mapped to a cluster view is the standard Mamdani–Assilian model [Mamdani and Assilian 1975] with t-norm minimum, regardless of the shape of the fuzzy sets, provided they are symmetric and have the same shape: simply define the radial function in the same way as the fuzzy sets are defined on the individual dimensions and use a maximum distance. Another fuzzy system that can be interpreted as a cluster model is one that is based on Gaussian fuzzy sets and the t-norm product. The reason is that a product of standard Gaussian functions is equivalent to a multi-dimensional Gaussian function based on a Mahalanobis distance parameterized with a diagonal covariance matrix, i.e.,

$$\prod_{j=1}^{m} f_{\text{gauss}}(d(x_j, y_j; \sigma_j^2); 2, 0) = f_{\text{gauss}}(d(\vec{x}, \vec{y}; \text{diag}(\sigma_1^2, \ldots, \sigma_m^2)); 2, 0).$$

It is actually well-known from functional equation theory that the standard Gaussian function is the only function that has this property.[11] However, this does not mean that fuzzy systems with t-norm product cannot be mapped if they use non-Gaussian fuzzy sets. In principle, we may change the radial function of the clusters to meet the shape requirements—it need not be identical to the membership function of the fuzzy sets.

As already pointed out in Section 1.2, this thesis focuses on *point prototypes*, that is, clusters are described by cluster centers, which are points in the data space, possibly enhanced by size and shape parameters. However, one may also consider more complex prototypes and measure the similarity to *shapes* in the data space rather than the similarity to a single point. One may, for example, use lines, planes, circles or (hyper-)spheres, ellipses

[11] Note that the parameter a of the generalized Gaussian function must be two due to the quadratic nature of the Mahalanobis distance.

or (hyper-)ellipsoids, rectangles or (hyper-)boxes etc. as cluster prototypes, approaches which are combined under the name of **shell clustering**. These methods can be highly useful in image processing, because they may be employed to detect shapes and thus to identify objects in images. An analysis of these methods is, however, beyond the scope of this thesis. A fairly detailed overview of fuzzy clustering approaches that use non-point prototypes can be found, for example, in [Höppner *et al.* 1999]. An abundance of references is also given in [Bezdek *et al.* 1999].

As a final remark, I should point out that cluster models that combine membership degrees to clusters with a linear function may not only be used for classification purposes, but also for the **prediction of numerical values**. Actually it is, in this context, often more convenient to see the discrimination between two classes as the prediction of a special real-valued function. This function may either be crisp and then assigns a value of 1 to all members of one class and a value of 0 to all other data points, or it is graded and thus represents class membership degrees or class probabilities. More than two classes are treated by setting up a classifier for each class, which discriminates it from all other classes (cf. Section 2.5), and then deciding on the class that yields the clearest assignment.

In this thesis, however, I confine myself to cluster models for classification, since only little extension is necessary to enable cluster models to predict numerical values generally (with a linear function to combine the cluster membership degrees, one merely has to revoke all restrictions on the coefficients w_{ik}). As we will see in the next chapter, classification actually allows for a wider variety of objective functions than numerical prediction and thus may be seen, in a way, as the more general task.

Chapter 3

Objective Functions

Even though not all clustering and classification algorithms, which try to find a set of prototypes to describe the data, start from an objective function to derive the induction procedure (some use heuristics that sometimes may even appear a bit *ad hoc*), their goal can often be characterized as the optimization of a certain quantity. This objective quantity also provides an immediate measure of the quality of the clustering result and thus defines when an algorithm has been successful. At least it enables us to compare two results of the same algorithm, which may have been obtained due to different initializations, by comparing the value of the objective function. It should be noted, though, as is also emphasized in [Bezdek *et al.* 1999], that an optimal result w.r.t. a chosen objective function is not necessarily the "best" result. Since human perception easily finds clusters, we may have expectations about the result, or an application may require specific properties, which are not properly represented by the objective function.

In this chapter I review objective functions for prototype-based classification and clustering, which can be divided into two main categories: those that are based on the minimization of the sum of certain (squared) deviations and those that start from a probabilistic model and try to maximize the likelihood or likelihood ratio of the data. These two main classes, which both contain clustering as well as classification approaches, are studied in Sections 3.1 (clustering based on the *least sum of squared distances*), 3.2 (classification based on the *least sum of squared errors*), 3.3 (clustering based on the *maximization of the likelihood* of the data), and 3.4 (classification based on the *maximization of the likelihood ratio* of the classes). In Section 3.5 I briefly consider alternative approaches.

3.1 Least Sum of Squared Distances

The oldest and most common objective function for clustering algorithms is the sum of (squared) distances of the data points to the cluster centers they are assigned to. The idea underlying this objective function is that good cluster centers should be as representative for the data points as possible, which holds intuitively if the deviation of the data points from the positions of the cluster centers is as small as possible. Therefore the sum of (squared) distances is to be minimized. This is directly analogous to the definition of location measures in descriptive statistics, where values (or vectors) are chosen, around which the dispersion of the data is minimal. (The arithmetic mean, for instance, minimizes the sum of squared differences to the sample values, the median minimizes the sum of absolute differences.)

Formally, the sum of (squared) distances can be written as

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}\, d_{ij}^2,$$

where $\mathbf{X} = \{\vec{x}_j \mid 1 \leq j \leq n\}$ is the given data set, which consists of n m-dimensional vectors (cf. Section 1.1), and $\mathbf{C} = \{\mathbf{c}_i \mid 1 \leq i \leq c\}$ is the set of cluster prototypes (cf. Chapter 2). $d_{ij} = d(\vec{x}_j, \vec{\mu}_i; \mathbf{\Sigma}_i)$ denotes the distance between datum $\vec{x}_j$ and the i-th cluster (cf. Section 2.1) and $u_{ij} \in \{0, 1\}$ states whether the datum $\vec{x}_j$ is assigned to the i-th cluster ($u_{ij} = 1$ if it is and $u_{ij} = 0$ otherwise). The $c \times n$ binary matrix $\mathbf{U} = (u_{ij})_{1 \leq i \leq c, 1 \leq j \leq n}$ combines the individual assignments and is called the **partition matrix**, because it describes how the data points are distributed on the different clusters. That the distribution is actually a partition, that is, that each data point is assigned to exactly one cluster, is ensured formally by requiring

$$\forall j; 1 \leq j \leq n : \quad \sum_{i=1}^{c} u_{ij} = 1.$$

This constraint also serves the purpose to rule out the trivial solution

$$\forall i; 1 \leq i \leq c : \forall j; 1 \leq j \leq n : \quad u_{ij} = 0,$$

which is obviously minimal, but not useful at all. Furthermore, one often finds the (basically only formal and rarely operational) constraint

$$\forall i; 1 \leq i \leq c : \quad \sum_{j=1}^{n} u_{ij} > 0,$$

which is meant to ensure that no cluster is empty, i.e., has no data points.

Being forced by the former constraint to set exactly one u_{ij} to one for a given datum $\vec{x}_j$, it is immediately clear that for a given set $\mathbf{C}$ of clusters the minimum is obtained by assigning $\vec{x}_j$ to the closest cluster, i.e., the cluster $\mathbf{c}_i$ for which the distance d_{ij} is minimal. Choosing cluster $\mathbf{c}_k$ instead, with $d_{kj} > d_{ij}$, would increase the objective function J by $d_{kj}^2 - d_{ij}^2 > 0$.

The above objective function is written with squared distances, since this is the most common choice. It leads to so-called **c-means clustering** [Ball and Hall 1967, Hartigan und Wong 1979, Lloyd 1982] (cf. Section 5.2). However, one may just as well use the distances directly, i.e., define

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}\, d_{ij},$$

which leads to so-called **c-medoids clustering** [Kaufman and Rousseeuw 1990, Krishnapuram *et al.* 1999, Chu *et al.* 2001], where a **medoid** is the multi-dimensional analog of a *median*. It is less popular than *c*-means clustering due to some technical problems, which can already be seen from the (one-dimensional) median: if the number of data points is even and the two values x_l and x_r in the middle of a sorted list of the sample values differ, then any value in $[x_l, x_r]$ minimizes the sum of absolute differences and thus it is not uniquely determined. In multiple dimensions this occurs less often (except when the city block distance is used, with which any vector in a certain (hyper-)box may minimize the sum of distances), but nevertheless it can be very tedious to determine the medoid.

In order to cope with this problem, one requires, as it is often also the case for the median, that the chosen value must come from the data set.[1] As a consequence, one tries to find the most central element of the data points that are assigned to a cluster and uses it as the cluster center.

A further generalization of an objective function based on the least sum of (squared) distances can be achieved by using a similarity measure instead of a distance. That is, the objective function may be defined as

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} \frac{u_{ij}}{u_i^{\bullet}(\vec{x}_j)},$$

where $u_i^{\bullet}$ is the weighted and transformed membership function of the cluster $\mathbf{c}_i$ as it was defined in Section 2.4. That both the elements u_{ij} of

[1] For the median, for example, this is mandatory if the sample values come from an attribute with an ordinal scale instead of a metric scale, because then the arithmetical mean of the two central elements cannot be (meaningfully) computed.

the partition and the different membership functions (raw u_i°, weighted u_i^*, transformed $u_i^\bullet$, and normalized u_i) are denoted by u may appear to be confusing at first sight, but is actually very natural. As we will see in Chapter 5, the elements u_{ij} of the partition matrix are usually computed as the normalized membership degrees, that is, $u_{ij} = u_i(\vec{x}_j) = u(\vec{x}_j, \mathbf{c}_i)$, which is only reflected by the identical symbol.

From this generalized objective function we can easily recover the objective functions of c-means and c-medoids clustering by choosing the Cauchy radial function with parameters $a = 2$, $b = 0$ and $a = 1$, $b = 0$, respectively (cf. page 16 in Section 2.2). In addition, we may choose arbitrary other exponents of the distance (other values for the parameter a), may set b to other values than 0, or employ a Gaussian radial function.

Note, however, that as long as we consider a crisp assignment of the data points to the clusters (i.e., $u_{ij} \in \{0, 1\}$), the choice of the radial function or the value of the parameter b has no influence on the assignment for a fixed set $\mathbf{C}$ of clusters. The reason is that in this case only the relative order of the membership degrees is relevant, not their absolute value, and since both the (generalized) Cauchy function as well as the (generalized) Gaussian function are monotonically decreasing, they lead to the same order of the membership degrees. As a consequence, a data point is always assigned to the cluster to which it has the smallest distance (or highest degree of membership).

However, this does *not* mean that all of these objective functions have their optimal value for the same set of clusters. For the choice of the cluster parameters the chosen radial function *is* relevant, as we will see in Chapter 5. We may not even conclude that the optimal partition is the same, because with differing choices of the clusters, different partitions may be optimal. Only for an already determined and fixed set $\mathbf{C}$ of clusters, the assignment is identical for all (parameterizations of the) radial functions.

Most classical clustering algorithms assign each datum to exactly one cluster, thus forming a crisp partition of the given data. However, as we already saw in Section 2.4, such a crisp assignment is not always appropriate, since it can lead to somewhat arbitrary decisions about the location of the cluster boundaries. In order to cope with this problem, normalizations of the membership degrees, for instance, to sum 1 are employed instead of a crisp assignment, as discussed in Section 2.4. In connection with the type of objective function we are considering here, this leads to so-called **fuzzy clustering** [Ruspini 1969, Dunn 1973, Bezdek 1981, Bezdek *et al.* 1999, Höppner *et al.* 1999] and in particular to **fuzzy c-means clustering**.

In order to incorporate degrees of membership into the objective function, it may seem, at first sight, to be sufficient to simply extend the allowed

range of values of the u_{ij} from the set $\{0, 1\}$ to the real interval $[0, 1]$, but to make no changes to the objective function itself, as it is used for crisp clustering (i.e. the sum of squared distances). However, this is not the case: the optimum of the objective function is obtained for a crisp assignment, regardless of whether we enforce a crisp assignment or not and also regardless of the distance measure or similarity function used.

This can easily be demonstrated as follows: let $k_j = \operatorname{argmin}_{i=1}^{c} d_{ij}^2$, that is, let k_j be the index of the cluster closest to the data point $\vec{x}_j$. Then it is

$$\begin{aligned} J(\mathbf{X}, \mathbf{U}, \mathbf{C}) &= \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}\, d_{ij}^2 \\ &\geq \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}\, d_{k_j j}^2 = \sum_{j=1}^{n} d_{k_j j}^2 \underbrace{\sum_{i=1}^{c} u_{ij}}_{=1 \text{ (due to the first constraint)}} \\ &= \sum_{j=1}^{n} \left(1 \cdot d_{k_j j}^2 + \sum_{\substack{i=1 \\ i \neq k_j}}^{c} 0 \cdot d_{ij}^2 \right). \end{aligned}$$

Therefore it is best to set $u_{k_j j} = 1$ and $u_{ij} = 0$ for $1 \leq i \leq c$, $i \neq k_j$. In other words: the objective function is minimized by assigning each data point crisply to the closest cluster. Analogous arguments hold, of course, for membership functions instead of (squared) distances, since both the Cauchy function and the Gaussian function are monotonically strictly decreasing.

A careful analysis of what is necessary to make a graded assignment optimal for an objective function that is based on (squared) distances was first presented in [Klawonn and Höppner 2003]. Since the distances are fixed by the positions of the data points and the cluster centers, and the membership function does not change the above argument either, we must apply a function to the (graded) assignments u_{ij} in the objective function.

Formally, we consider the objective function

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} h(u_{ij})\, d_{ij}^2,$$

where h is the mentioned function. In order to explore what properties this function h must have, we consider the terms of the objective function that refer to a single data point $\vec{x}_j$. For simplicity, we also confine ourselves to two clusters $\mathbf{c}_1$ and $\mathbf{c}_2$. That is, we consider

$$J(\vec{x}_j, u_{1j}, u_{2j}) = h(u_{1j})\, d_{1j}^2 + h(u_{2j})\, d_{2j}^2$$

and study how it behaves for different values u_{1j} and u_{2j}. A minor technical complication is introduced into this study by the fact that we may not want to rule out a crisp assignment categorically: if the distances d_{1j} and d_{2j} differ significantly, a crisp assignment to the closer cluster should still be possible. Hence we assume that they differ only slightly, so that we actually have a situation in which a graded assignment is desired.

Our goal, of course, is to minimize $J(\vec{x}_j, u_{1j}, u_{2j})$ by choosing u_{1j} and u_{2j} appropriately. Since we have only two clusters, the constraint $\sum_{i=1}^{c} u_{ij} = 1$ enables us to express u_{2j} by u_{1j}. Hence we consider

$$J(\vec{x}_j, u_{1j}) = h(u_{1j})\, d_{1j}^2 + h(1 - u_{1j})\, d_{2j}^2$$

A necessary condition for a minimum of this functional is

$$\frac{\partial}{\partial u_{1j}} J(\vec{x}_j, u_{1j}) \quad = \quad h'(u_{1j})\, d_{1j}^2 - h'(1 - u_{1j})\, d_{2j}^2 \quad = \quad 0,$$

where the prime ($'$) denotes taking the derivative w.r.t. the argument of the function. This leads to the simple condition

$$h'(u_{1j})\, d_{1j}^2 = h'(1 - u_{1j})\, d_{2j}^2,$$

which yields another argument that a graded assignment cannot be optimal without any function h: if h is the identity, we have $h'(u_{1j}) = h'(1-u_{1j}) = 1$ and thus the equation cannot hold if the distances differ.

For the further analysis let us assume, without loss of generality, that $d_{1j} < d_{2j}$. Hence we have $h'(u_{1j}) > h'(1 - u_{1j})$. In addition, we know that $u_{1j} > u_{2j} = 1 - u_{1j}$, because the closer cluster should get the higher degree of membership. In other words, the function h must be the steeper, the greater its argument, and, of course, it may coincide with the identity only for the extreme arguments 0 and 1. Therefore it must be a convex function on the unit interval [Klawonn and Höppner 2003].

The most common choice for the function h is $h(u_{ij}) = u_{ij}^2$, which leads to the standard objective function of fuzzy clustering [Ruspini 1969, Dunn 1973, Bezdek 1981, Bezdek *et al.* 1999, Höppner *et al.* 1999], namely

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^2\, d_{ij}^2.$$

Here $u_{ij} \in [0, 1]$ is the (now graded) assignment of the data point $\vec{x}_j$ to the cluster $\mathbf{c}_i$ and d_{ij} is the distance between data point $\vec{x}_j$ and cluster $\mathbf{c}_i$. The $c \times n$ matrix $\mathbf{U} = (u_{ij})$ is called the **fuzzy partition matrix**, where "fuzzy" indicates that its elements now come from the interval $[0, 1]$.

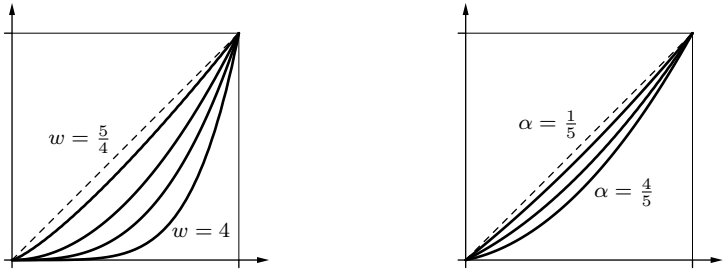

Figure 3.1: Different mapping functions h for the (graded) assignments in the objective function $J(\mathbf{X}, \mathbf{U}, \mathbf{C})$. Left: $h(u_{ij}) = u_{ij}^w$, $w = \frac{5}{4}, 2, 3, 5$. Right: $h(u_{ij}) = \alpha u_{ij}^2 + (1-\alpha)u_{ij}$, $\alpha = \frac{1}{5}, \frac{1}{2}, \frac{4}{5}$.

The first generalization was suggested in [Bezdek 1981] by introducing an arbitrary exponent w instead of 2, which is called the **fuzzifier** or **weighting exponent**. That is, we now choose $h(u_{ij}) = u_{ij}^w$, $w \in (1, \infty)$ (see Figure 3.1 on the left), which leads to the objective function

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^w \, d_{ij}^2, \qquad w \in (1, \infty).$$

The fuzzifier w determines the "fuzziness" of the clustering: with higher values for w the boundaries between the clusters become softer, with lower values they get harder. (This can easily be seen from the left diagram in Figure 3.1.) Hard clustering results in the limit for $w \to 1_+$.

Although it has been applied successfully to a large number of clustering tasks, this choice of the function h has a serious drawback, which was already mentioned in passing in Section 2.4. In a way, it "over-corrects" the weakness of an objective function in which the (graded) assignments appear directly: with this choice of h, the optimum is necessarily achieved for a graded assignment. A crisp assignment is completely ruled out, regardless of the ratio of the distances to the different clusters.

As a consequence, we get the undesired effects described in Section 2.4, namely that the clustering algorithm encounters serious problems if it is to recognize clusters that differ considerably in population density. The higher number of points in a densely populated cluster, even though they have a low degree of membership to a cluster prototype that is far away, can "drag" that cluster prototype away from a less densely populated cluster.

As was revealed in [Klawonn and Höppner 2003], the reason for the inconvenient property that the optimum is now necessarily obtained for a graded assignment (with the only exception of a data point that coincides with a cluster center) lies in the fact that for $h(u_{ij}) = u_{ij}^w$ we have $h'(0) = 0$. To understand this, consider again the relation

$$h'(u_{1j})\, d_{1j}^2 = h'(1 - u_{1j})\, d_{2j}^2$$

which was derived above for the assignment of a data point $\vec{x}_j$ to two clusters. Obviously, this equation cannot hold if both distances are positive and one of the factors $h'(u_{1j})$ and $h'(1-u_{1j})$ is zero (note that they cannot both be zero at the same time, because h' must be, as was derived above, a strictly increasing function). Therefore, if $h'(u_{ij})$ cannot be zero and $h'(0) = 0$, it follows that u_{ij} cannot be zero. Likewise, since $h'(1 - u_{ij})$ cannot be zero, u_{ij} cannot be 1 either. This rules out a crisp assignment of any data point, with the exception of those that coincide with a cluster center.

As a consequence of this argument we have that, if we want to allow crisp assignments (if only for extreme distance ratios), we must choose a function h, the derivative of which does not vanish at 0. In [Klawonn and Höppner 2003] it was suggested to use $h(u_{ij}) = \alpha u_{ij}^2 + (1-\alpha)u_{ij}$, $\alpha \in (0, 1]$, which is shown for three different choices of α in Figure 3.1 on the right. This choice leads to the objective function

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} (\alpha u_{ij}^2 + (1-\alpha)u_{ij})\, d_{ij}^2, \qquad \alpha \in (0, 1].$$

However, the parameter α of this objective function is difficult to interpret and thus difficult to choose in applications. To find a better parameterization, consider again the above relation for the assignment of a data point $\vec{x}_j$ to two clusters. Suppose that $d_{ij} < d_{kj}$ and $u_{1j} = 1$ and consider

$$\frac{d_{2j}^2}{d_{1j}^2} = \frac{h'(0)}{h'(1)} = \frac{1-\alpha}{2\alpha + (1-\alpha)} = \frac{1-\alpha}{1+\alpha}.$$

This shows that the minimum ratio of the smaller to the larger squared distance, at which we first get a crisp assignment, is $\beta = \frac{1-\alpha}{1+\alpha}$. Since this is an easily interpretable quantity, it is plausible to parameterize the objective function with it, which, due to $\alpha = \frac{1-\beta}{1+\beta}$ and $1 - \alpha = \frac{2\beta}{1+\beta}$, leads to

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} \left(\frac{1-\beta}{1+\beta} u_{ij}^2 + \frac{2\beta}{1+\beta} u_{ij} \right) d_{ij}^2, \qquad \beta \in [0, 1).$$

For $\beta = 0$ we have the standard objective function of fuzzy clustering.

Another direction in which an objective function that is based on the sum of squared distances may be generalized concerns the constraint

$$\forall j; 1 \leq j \leq n: \quad \sum_{i=1}^{c} u_{ij} = 1.$$

Intuitively, it ensures that each data point has the same total influence by requiring that the (graded) assignments of a data point are normalized to sum 1. As a consequence an approach that relies on this constraint is often called **probabilistic fuzzy clustering**, since with it the (graded) assignments of a data point formally resemble the probabilities of its being a member of the corresponding clusters. The partitioning property of a probabilistic clustering algorithm, which "distributes" the weight of a data point to the different clusters, is due to this constraint.

However, we already saw in Section 2.4 that such a normalization also has its drawbacks, namely that the maximum of the membership function may not coincide with the cluster center anymore and that the membership degrees may increase again once we move beyond the center of another cluster. Since such a behavior can cause confusion, it is attractive to discard it and allow arbitrary (graded) assignments of a data point to different clusters. Such an approach is usually called **possibilistic fuzzy clustering** [Krishnapuram and Keller 1993], since in this case the (graded) assignments of a data point formally resemble the possibility degrees[2] of its being a member of the corresponding clusters.

However, we cannot simply discard the normalization constraint. As I pointed out at the beginning of this section, it also serves the purpose to rule out the trivial minimization of the objective function by setting all $u_{ij} = 0$, $1 \leq i \leq c$, $1 \leq j \leq n$. Therefore, if we want to eliminate the normalization constraint, we have to introduce other means to suppress the trivial solution. In [Krishnapuram and Keller 1993] it was suggested to extend the objective function by a term that penalizes low (graded) assignments. Since the objective function is to be minimized, this additional term basically sums their differences to 1. That is, the objective function (with a fuzzifier w) is

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{w} \, d_{ij}^{2} + \sum_{i=1}^{c} \nu_i \sum_{j=1}^{n} (1 - u_{ij})^{w},$$

where the ν_i, $1 \leq i \leq c$, are cluster-specific penalty weights.

[2] For an extensive study of possibility theory, which is beyond the scope of this thesis, see [Dubois and Prade 1988]. A brief intuitive introduction can also be found in Chapter 2 of [Borgelt and Kruse 2002].

Unfortunately, there is another problem involved in dropping the normalization constraint, namely that it decouples the clusters. Since the (graded) assignment to each cluster can be chosen independently, the objective function is minimized by optimizing each cluster independently. This becomes particularly clear, if one writes the objective function as

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \left(\sum_{j=1}^{n} u_{ij}^{w} \, d_{ij}^{2} + \nu_i \sum_{j=1}^{n} (1 - u_{ij})^{w} \right)$$

and notices that the terms of the outer sum are independent. The consequence is immediately clear: except in very rare cases, in which the data points exhibit a high symmetry so that there is more than one global minimum, the objective function is minimized only if all cluster prototypes are identical. If two clusters differed, then one of the two terms of the outer sum that refer to these two clusters would be smaller (with the exception of cases with a highly symmetrical data point distribution). Therefore the value of the objective function could be reduced by setting the cluster with the greater term equal to the cluster with the smaller term, which is possible, because the terms have no influence on each other.

Strangely enough it took the scientific community a long time to realize this irritating fact. Even though [Barni *et al.* 1996] already reported problems with possibilistic fuzzy clustering, namely that clusters often became identical in the possibilistic fuzzy c-means algorithm, and [Krishnapuram and Keller 1996] commented on these observations, suggesting an ineffective cure, the true cause was revealed only in [Timm *et al.* 2001, Timm and Kruse 2002, Timm 2002, Timm *et al.* 2004]. The reason why it was overlooked so long is that for clearly separated clusters the objective function has pronounced local minima, each of which may be found by a cluster, depending on the initialization. Thus possibilistic fuzzy clustering sometimes yields meaningful results. If the size of the clusters (cf. Section 2.3) is properly chosen, it may even yield a useful result if the clusters are not well separated, even though it is not a true minimum of the objective function.

To illustrate this fact, and also the effect of choosing different (fixed) cluster sizes, I visualized the vector field describing the gradient of the objective function for a single cluster center in possibilistic clustering based on the Euclidean distance. The direction of the gradient vector is coded by the hue on a standard color circle and its length by the saturation of the color. As an example, Figure 3.2 shows the gradient for a single data point that is located at the center. The closer the cluster center gets to this point, the smaller is the gradient and thus the darker the color.

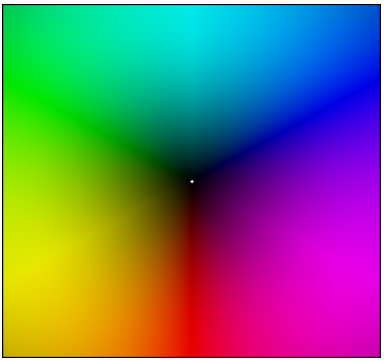

Figure 3.2: Color coding of the gradient of the objective function of possibilistic fuzzy clustering for a single cluster center. The hue represents the direction (all vectors point towards the center) and the saturation the strength of the attraction. This picture was generated with a single data point in the center.

Since the pictures get clearer in this way, I consider an objective function based on cluster membership functions instead of pure distances, that is

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \left(\sum_{j=1}^{n} \frac{u_{ij}^{w}}{u_i^{\circ}(\vec{x}_j)} + \nu_i \sum_{j=1}^{n} (1 - u_{ij})^w \right).$$

Note that the membership function is the raw one (i.e., u_i°), so there is no cluster weighting or membership transformation. However, the resulting vector field is scaled before visualization, so that the color saturation is shifted to a range in which the different levels are clearly distinguishable, sometimes accepting a cut-off at maximum saturation.

As a test case I took the well-known Iris data set [Anderson 1935], attributes petal length and petal width. Figures 3.3 to 3.5 show the vector field for a Cauchy function with $a = 2$, $b = 1$, Figures 3.6 to 3.8 the vector field for a Gaussian function with $a = 2$. The cluster radii are 1, $\frac{1}{4}$, and $\frac{1}{6}$, respectively. The data points are shown as small white circles. The group in the lower left corner are the Iris setosa cases, the group in the upper right corner comprises Iris versicolor (lower left part of the data point cloud) and Iris virginica (upper right part of the data point cloud).

Figures 3.3 already shows the problem very clearly. There is only one minimum, namely in the middle of the black area in the Iris versicolor/virginica cloud. The dark area to the lower left of it only shows a region of smaller gradient, but does not contain a attraction center, as can be seen from the colors surrounding it. However, if the cluster radius is reduced (Figures 3.4 and 3.5), an attraction center forms in the Iris setosa cloud.

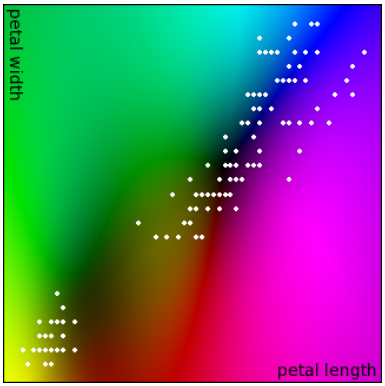

Figure 3.3: Vector field of the attraction of a single spherical cluster prototype in possibilistic fuzzy clustering, radius 1, Cauchy function with parameters $a = 2$ and $b = 1$, for the Iris data, attributes petal length (horizontal) and width (vertical).

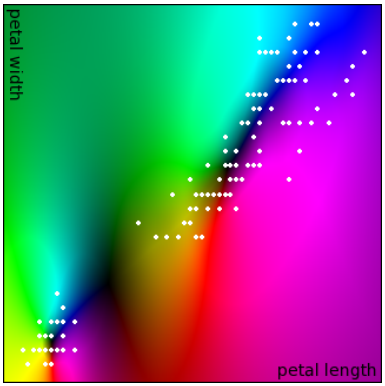

Figure 3.4: Vector field of the attraction of a single spherical cluster prototype in possibilistic fuzzy clustering, radius $\frac{1}{4}$, Cauchy function with parameters $a = 2$ and $b = 1$, for the Iris data, attributes petal length (horizontal) and width (vertical).

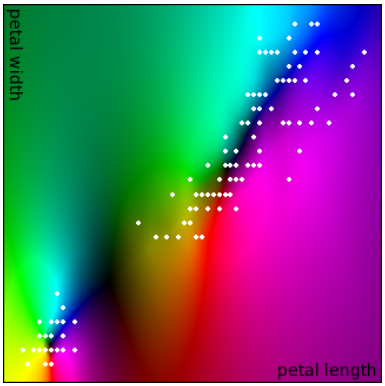

Figure 3.5: Vector field of the attraction of a single spherical cluster prototype in possibilistic fuzzy clustering, radius $\frac{1}{6}$, Cauchy function with parameters $a = 2$ and $b = 1$, for the Iris data, attributes petal length (horizontal) and width (vertical).

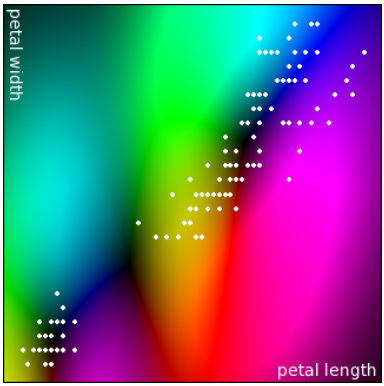

Figure 3.6: Vector field of the attraction of a single spherical cluster prototype in possibilistic fuzzy clustering, radius 1, Gaussian function with parameter $a = 2$, for the Iris data, attributes petal length (horizontal) and width (vertical).

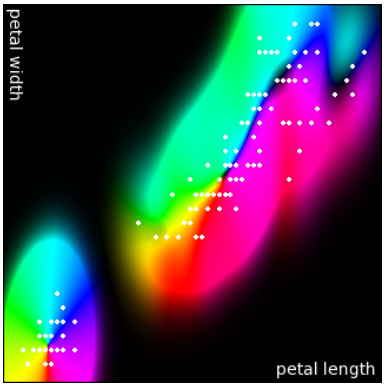

Figure 3.7: Vector field of the attraction of a single spherical cluster prototype in possibilistic fuzzy clustering, radius $\frac{1}{4}$, Gaussian function with parameter $a = 2$, for the Iris data, attributes petal length (horizontal) and width (vertical).

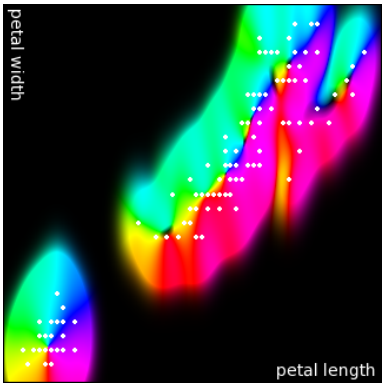

Figure 3.8: Vector field of the attraction of a single spherical cluster prototype in possibilistic fuzzy clustering, radius $\frac{1}{6}$, Gaussian function with parameter $a = 2$, for the Iris data, attributes petal length (horizontal) and width (vertical).

Similar observations can be made for the Gaussian function. Although there are already two local minima with a cluster radius of 1 (Figure 3.6, the dark region between the two data point clouds may also contain an unstable equilibrium point) due to the fact that the Gaussian function is steeper, their number increases with lower radius. With radius $\frac{1}{4}$, two more local minima have almost formed in the upper right of the picture, although they are not stable yet. With radius $\frac{1}{6}$, however, there are several local minima along the Iris versicolor/virginica data point cloud.

In such a situation the result of clustering depends heavily on the initialization. Any distribution of the clusters to these local minima is possible. We cannot even single out the best one by comparing the values of the objective function for these outcomes, because the best in this sense is the one in which all clusters are identical. As a consequence, possibilistic fuzzy clustering should be applied with much caution, if at all.

A working approach that tackles the problem has been suggested in [Timm *et al.* 2001, Timm 2002, Timm *et al.* 2004], namely introducing a term that models a mutual repulsion of the clusters. This leads to

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{w} d_{ij}^{2} + \sum_{i=1}^{c} \nu_i \sum_{j=1}^{n} (1 - u_{ij})^{w} + \sum_{i=1}^{c} \xi_i \sum_{\substack{k=1 \\ k \neq i}}^{c} f(\zeta \cdot d(\vec{\mu}_i, \vec{\mu}_k)),$$

where f is a radial function, for example, the Cauchy function or the Gaussian function, and the ξ_i and ζ are parameters that control the strength of the cluster repulsion. By this repulsion it is explicitely prevented that clusters become identical. However, the parameters that enter this approach are difficult to choose and thus its application can be cumbersome. Therefore I do not consider this approach any further.

Another approach, which is also directed at the normalization constraint, consists not in dropping it (as in possibilistic clustering), but in replacing it, for example, by

$$\forall j; 1 \leq j \leq n : \quad \sum_{i=1}^{c} \sqrt[w]{u_{ij}} = 1,$$

while the (unextended) objective function J is defined without a membership degree mapping function h. This approach is very closely related to the approach based on using $h(u_{ij}) = u_{ij}^{w}$ in the objective function J. In particular, for $w = 2$ the cluster parameters, for which the optimum is obtained, are identical, only the membership degrees u_{ij} differ (cf. Section 5.2.3 for details). Note that the parameter w appearing here plays basically the same role as the fuzzifier in the first generalization of fuzzy clustering.

Furthermore, other functions than $\sqrt[w]{u_{ij}}$ may be used to define the constraint. However, by analogy to the requirement that the membership mapping function h has to be convex to make it possible that the optimum is obtained with a graded assignment of the data points (cf. the discussion on page 50), it is clear that the function should be concave. Alternatively, one may reason (using a substitution) that its inverse must be convex.

3.2 Least Sum of Squared Errors

When it comes to classification instead of clustering, it is much easier to find appropriate objective functions, because the goal of classification is much clearer defined. In clustering we search for a model that "captures the distribution of the data well" by finding clusters that are "representative for a group of data points", goals that are open to a fairly wide range of interpretations. In classification, on the other hand, we are given a desired output z_j for each of the data points $\vec{x}_j$, $1 \leq j \leq n$, and thus we can derive an objective function from these desired outputs.

The most direct way to define an objective function is to compute for each data point $\vec{x}_j$ the output o_j of a given model and to compare it the desired output z_j. Then we either count the number of correct classifications, which gives an objective function that is to be maximized, or the number of misclassifications, which gives an objective function that is to be minimized. The latter is more common and is well-known as **0-1 loss**.

However, even though this approach is very simple and intuitive, it suffers from the severe drawback that it is a fairly coarse measure. With it a large number of models may appear to be equivalent, since considerable changes to the model may be needed in order to change the classification for at least one data point. Although it is clear that in the end, when a decision has to be made, only the number of correct and wrong classification count, for the induction and adaptation of the classification model a more sensitive measure is needed. For this task 0-1 loss is actually a fairly bad choice and thus I do not consider it as an objective function.

A more fine-grained evaluation of a classification model can be obtained if the model assigns degrees of membership or probabilities to the different classes. This is naturally the case if a cluster model is turned into a classifier by linear functions combining the membership degrees as it was studied in Section 2.5. In this case there is one linear function for each of the classes and the values of these functions, possibly normalized to sum 1, can be seen as graded assignments to the classes. However, it is also possible to obtain

graded assignments with a nearest prototype classifier, if one extends it in such a way that it reports not only the class of the closest cluster, but the degrees of membership to the closest cluster of each class. These degrees of membership may also be captured by classification functions g_k, $1 \leq k \leq s$, as it is the case for a linear function classifier (cf. page 38 in Section 2.5), only that these functions are not necessarily linear in nature.

Given a graded assignment an objective function can be defined by computing its divergence from a binary encoding of the desired output class (cf. page 3 in Section 1.1). Formally, it can be defined as

$$e_{\text{sqr}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) = \sum_{j=1}^{n} \sum_{k=1}^{s} (\delta_{k,z_j} - g_k(\vec{x}_j))^2.$$

The vector $\vec{z} = (z_1, \ldots, z_n)$ states the classes $z_j \in \{1, \ldots, s\}$ of the data points $\vec{x}_j$, $1 \leq j \leq n$ (cf. page 4 in Section 1.1), and the weight matrix $\mathbf{W} = (w_{ki})_{1 \leq k \leq s, 0 \leq i \leq c}$ contains the parameters of the (linear) classification functions associated with the classes (cf. page 38 in Section 2.5). Finally, δ is the so-called *Kronecker symbol*,

$$\delta_{k,z} = \begin{cases} 1, & \text{if } k = z, \\ 0, & \text{otherwise,} \end{cases}$$

which describes the encoding of the correct class as a binary vector. This objective function is known as the **sum of squared errors** or **quadratic loss** and is obviously to be minimized. It may also be applied if the desired output is a numeric value, by using only a single **prediction function** g, which computes an output value for each data point, and defining

$$e_{\text{sqr}}(\mathbf{X}, \vec{z}; \mathbf{C}, \vec{w}) = \sum_{j=1}^{n} (z_j - g(\vec{x}_j))^2.$$

This possibility should be kept in mind, even though I focus on classification, because both versions appear in connection with the training of radial basis function neural networks (cf. Section 5.1). It should also be noted that the objective function for classification may be seen as the sum of the objective functions for s numerical functions, each of which predicts the membership to one class against all others as a binary function.

The above objective function sums the *square* of the errors, although computing the **sum of absolute errors** or **absolute loss** as

$$e_{\text{abs}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) = \sum_{j=1}^{n} \sum_{k=1}^{s} |\delta_{k,z_j} - g_k(\vec{x}_j)|$$

(analogously for the numeric prediction case) may appear more natural.[3] However, there are two reasons for preferring the sum of squared errors. The first is that an error-based objective function is often used with gradient methods, for which its derivative has to be computed. This is easy with squared errors, but can be inconvenient with absolute errors, because even if the derivative of the absolute value is reasonably completed for a vanishing argument[4], an annoying distinction of cases is necessary.

The second reason is that with squared errors large deviations have a greater influence and thus the optimization algorithm tends to avoid large errors, even at the cost of increasing smaller ones. However, this can also turn out to be a disadvantage, as it makes the procedure susceptible to the influence of outliers, which can distort the result. The sum of absolute errors is a much more robust objective function in this sense [Press *et al.* 1992].

A final issue one has to consider in connection with an error-based objective function for classification are **misclassification costs**. In the above discussion I always assumed implicitly that the same costs are incurred, regardless of which class is confused with which. However, in applications this is not always the case. For a company that sends out advertisements to possible customers it is usually less costly to send a superfluous mail to a non-buyer, who could have been excluded by a more detailed analysis, than to lose a potential buyer by omitting him/her from the mailing. In the medical domain it may be less critical to diagnose a patient as having a disease he/she has not, as this can be corrected without harm by further tests, than to misclassify an actually ill patient as healthy, which may then be discovered only after the illness has spread and intensified.

The simplest way to include misclassification costs into an error-based objective function, to which I confine myself here, is by weighting the classes depending on how important it is to predict them correctly. In this case the sum of squared errors, for example, is modified to

$$e(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) = \sum_{j=1}^{n} \sum_{k=1}^{s} \xi_k \cdot (\delta_{k,z_j} - g_k(\vec{x}_j))^2,$$

where the ξ_k, $1 \leq k \leq s$, are the class-specific weights. Equivalently, one may weight the data points w.r.t. the classes associated with them. This may be technically easier as it can be performed in a preprocessing step.

[3] It should be clear that simply summing the differences (not their absolute values) is not a good idea, as positive and negative differences may cancel each other.

[4] Often it is defined that $\frac{\mathrm{d}}{\mathrm{d}x}|x| = \mathrm{sgn}(x) = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{if } x = 0, \\ -1, & \text{if } x < 0. \end{cases}$

3.3 Maximum Likelihood

As I already pointed out in Section 2.4, a set of clusters in the view I adopt in this thesis can be seen as a description of a probabilistic mixture model [Everitt and Hand 1981]. This presupposes that the membership functions of the clusters are normalized to integral 1 over the data space, so that they can be interpreted as (conditional) probability density functions, and the cluster weights sum up to 1, so that they can be seen as prior probabilities of the clusters. In addition, of course, the membership degrees must not be transformed, i.e., it must be $u_i^\bullet(\vec{x}) = u_i^*(\vec{x})$ (cf. page 23 in Section 2.4).

Formally, we have in this case (cf. page 29 in Section 2.4)

$$\begin{aligned}\sum_{y=1}^{c} u_y^\bullet(\vec{x}) &= \sum_{y=1}^{c} u_i^*(\vec{x}) = \sum_{y=1}^{c} \varrho_y \cdot u_y^\circ(\vec{x}) \\ &= \sum_{y=1}^{c} p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}|y; \mathbf{C}) \\ &= \sum_{y=1}^{c} f_{\vec{X},Y}(\vec{x}, y; \mathbf{C}) = f_{\vec{X}}(\vec{x}; \mathbf{C}).\end{aligned}$$

As a consequence, we can write the **likelihood** of the complete data set $\mathbf{X}$ (implicitly assuming that all data points are independent) as

$$L(\mathbf{X}; \mathbf{C}) = f_{\mathcal{X}}(\mathbf{X}; \mathbf{C}) = \prod_{j=1}^{n} f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}) = \prod_{j=1}^{n} \sum_{i=1}^{c} u_i^\bullet(\vec{x}).$$

Drawing on the principle of **maximum likelihood estimation** [Fisher 1925], we may choose this function as an objective function. It is based on the idea to estimate the parameters of a probabilistic model in such a way that it becomes maximally likely to observe the given data. This is the basis of probabilistic approaches to clustering that employ the **expectation maximization (EM) algorithm** [Dempster *et al.* 1977, Everitt and Hand 1981, Jamshidian and Jennrich 1993, Bilmes 1997].

Most often the mixture model underlying a maximum likelihood approach is chosen to be a **Gaussian mixture model**, that is, the cluster membership functions are (conditional) normal distributions. Then we have

$$L(\mathbf{X}; \mathbf{C}) = \prod_{j=1}^{n} \sum_{i=1}^{c} \varrho_i \cdot \gamma_{\text{Gauss}}(2, 0, m, \mathbf{\Sigma}_i) \cdot f_{\text{Gauss}}(d(\vec{x}_j, \vec{\mu}_i; \mathbf{\Sigma}_i); 2, 0)$$

with the functions γ_{Gauss} and f_{Gauss} defined as on page 19 in Section 2.2.

An important advantage of a Gaussian mixture model, which has been known for a long time already, is that a set of finite mixtures of m-dimensional normal distributions is identifiable. That is, if at all, a probability density function f can be written in only one way as a mixture of finitely many normal distributions [Bock 1974]. Unfortunately, this does not mean that there is a unique optimum of the likelihood function for a given data set, even if this data set was generated by sampling from a mixture of normal distributions. The problem is that an unconstrained optimization of the likelihood function leads to a situation in which all normal distributions, with the exception of one, are contracted to Dirac pulses at individual data points (i.e., $\sigma_i^2 \to 0$). The remaining normal distribution is the maximum likelihood estimate for the whole data set.

Therefore, to ensure useful results, maximum likelihood clustering is usually applied with constraints on the covariance matrices that enter the Mahalanobis distance on which the Gaussian function is based. A classical approach, for example, is to set all covariance matrices to $\sigma^2\mathbf{1}$, with a variance σ^2 that has to be chosen appropriately for the data set. (For low dimensional data sets that are normalized to variance 1 in each dimension, my experience is that $\sigma = \frac{1}{c}$ is a choice that often works well.) In addition, one often fixes the prior probabilities of the clusters to equal values.

With standard Gaussian membership functions, maximum likelihood is closely connected to a least sum of (squared) distances approach (cf. Section 3.1). This is very similar to the **method of least squares** for finding regression lines or regression polynomials, which can be seen as a maximum likelihood estimator for the parameters of the regression function if one assumes normally distributed deviations (with constant variance) from the function value. With a mixture model, however, the situation is slightly more complicated, because the combination of a product and a sum in the objective function leads to technical problems in the optimization.

A standard approach to handle these problems consists in assuming that there is a hidden attribute Y, which states the cluster (i.e. the component of the mixture) that "generated" a given data point. With such an attribute we can restrict our considerations to one term, namely the one referring to the mixture component indicated by the value $y_j \in \{1, \ldots, c\}$, $1 \leq j \leq n$, of this attribute. Formally, the objective function can be written as

$$f_{\mathcal{X},\vec{Y}}(\mathbf{X}, \vec{y}; \mathbf{C}) = \prod_{j=1}^{n} \frac{\varrho_{y_j}}{\sqrt{(2\pi)^m |\mathbf{\Sigma}_{y_j}|}} \cdot \exp\left(-\frac{1}{2} d^2(\vec{x}_j, \vec{\mu}_{y_j}; \mathbf{\Sigma}_{y_j})\right),$$

where y_j indicates the cluster that "generated" the data point $\vec{x}_j$.

Unfortunately, although it is a very natural way of stating the corresponding cluster, this notation (with random variables Y_j) is a little inconvenient for showing the equivalence to a least sum of squared distances approach. A better way is to draw on the following technical trick: we represent the individual factors of the above product as

$$f_{\vec{X}_j}(\vec{x}_j, \vec{u}_j; \mathbf{C}) = \prod_{i=1}^{c} \left(\frac{\varrho_i}{\sqrt{(2\pi)^m |\mathbf{\Sigma}_i|}} \cdot \exp\left(-\frac{1}{2} d_{ij}^2\right) \right)^{u_{ij}}.$$

Here u_{ij} is a binary variable that indicates whether data point $\vec{x}_j$ was "generated" by the i-th cluster (then it is $u_{ij} = 1$) or not (then it is $u_{ij} = 0$). In other words: the vector $\vec{u}_j = (u_{1j}, \ldots u_{cj})$ is a binary encoding of the cluster index y_j where $u_{ij} = \delta_{i,y_j}$, $1 \leq i \leq c$. Hence this product for $f_{\vec{X}_j}(\vec{x}_j, \vec{u}_j; \mathbf{C})$ effectively reduces to one factor that describes the probability density of the data point and the cluster that "generated" it.

Inserting this product into the objective function, we get

$$f_{\mathcal{X},\vec{Y}}(\mathbf{X}, \mathbf{U}; \mathbf{C}) = \prod_{j=1}^{n} \prod_{i=1}^{c} \left(\frac{\varrho_i}{\sqrt{(2\pi)^m |\mathbf{\Sigma}_i|}} \cdot \exp\left(-\frac{1}{2} d_{ij}^2\right) \right)^{u_{ij}}.$$

Next we take the natural logarithm of this objective function, which obviously does not change the location of the maximum, because the natural logarithm is a strictly monotone function.[5] Thus we obtain

$$\ln f_{\mathcal{X}}(\mathbf{X}, \mathbf{U}; \mathbf{C}) = \sum_{j=1}^{n} \sum_{i=1}^{c} u_{ij} \left(\ln \frac{\varrho_i}{\sqrt{(2\pi)^m |\mathbf{\Sigma}_i|}} - \frac{1}{2} d_{ij}^2 \right).$$

Provided that the prior probabilities ϱ_i of the different clusters as well as their covariance matrices $\mathbf{\Sigma}_i$ are fixed, the only variable term in this sum is the last, which consists of the (squared) distances of the data points to the (variable) cluster centers. Therefore, in this case *maximizing* this (log-)likelihood function is equivalent to *minimizing*

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{j=1}^{n} \sum_{i=1}^{c} u_{ij}\, d_{ij}^2.$$

However, this is the objective function of hard c-means clustering as it was introduced on page 46 in Section 3.1.

[5] Of course, taking the natural logarithm does change the *value* of the maximum. However, we are not interested in this value, but only in the *location* of the maximum.

Nevertheless there is a decisive difference between the approaches based on maximum likelihood and on the least sum of squared distances. In the latter, the u_{ij}, $1 \leq i \leq c$, $1 \leq j \leq n$, are introduced as parameters of the model, which are to be determined by the optimization process. Therefore their values can be chosen arbitrarily. In maximum likelihood, however, the values of the u_{ij} are fixed by the data generation process (they indicate which component of the Gaussian mixture "generated" the data point $\vec{x}_j$). We do not know their values, but we may *not* choose them arbitrarily.

How this problem is handled—namely by seeing the Y_j as random variables and then considering the *expected value* of the likelihood function, which leads to the well-known **expectation maximization algorithm** [Dempster *et al.* 1977, Everitt and Hand 1981, Jamshidian and Jennrich 1993, Bilmes 1997]—is discussed in detail in Section 5.2. It is closely related to fuzzy clustering approaches, as it also employs an alternating optimization scheme. Fuzzy clustering actually comes very close in the **fuzzy maximum likelihood estimation (FMLE) algorithm**, also known as the **Gath–Geva algorithm** [Gath and Geva 1989]: in the form in which it was proposed originally it is almost identical to expectation maximization. However, as will become clear later, a derivation of this algorithm that is based on strict analogies to standard fuzzy clustering leads to a clear distinction between the two approaches [Döring *et al.* 2004].

Another connection of maximum likelihood and expectation maximization to fuzzy clustering is brought about through **(fuzzy) learning vector quantization** [Kohonen 1986, Kohonen 1990, Kohonen 1995, Karayiannis and Pai 1996, Karayiannis and Bezdek 1997]: the competitive learning rule it employs can be derived as a limit case of a Gaussian mixture model. Details about this view can be found in Section 5.3.

3.4 Maximum Likelihood Ratio

A maximum likelihood approach can not only be used for clustering, as described in the preceding section, but also for classification. The simplest approach is to split the data set w.r.t. the classes associated with the data points, so that one obtains one data set for each class. Then one builds a maximum likelihood model separately for each of the classes, using the objective function studied in the preceding section. However, such an approach has the disadvantage that it is not really geared towards a high classification performance: a maximum likelihood model built separately for each of the classes need not yield a particularly good classifier.

A better approach starts from the idea that in order to meet the ultimate goal, namely to achieve a good classification performance, one rather has to consider how the likelihoods of a data point in connection with different classes relate to each other than to focus on the likelihood of the data point for the correct class, because the highest likelihood yields the classification. The basis for such an approach is that a cluster model together with a set of (linear) classification functions yields a probability distribution over the data space for each of the classes (cf. Section 2.5). In addition, we know the correct classes and thus the desired prediction for a given data set. Hence we can compute for each data point its likelihood in connection with the correct class and the complementary likelihood with any incorrect class.

Now it is plausible that for an optimal classifier the likelihood with the correct class should be maximal, while the likelihood with any incorrect class should be minimal. Therefore it has been suggested to use the ratio of the former to the latter as an objective function [Seo and Obermayer 2003]. That is, one tries to maximize the **likelihood ratio**

$$\begin{aligned} L^{(1)}_{\text{ratio}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) &= \prod_{j=1}^{n} \frac{L(\vec{x}_j, z_j; \mathbf{C}, \mathbf{W})}{L(\vec{x}_j, \overline{z_j}; \mathbf{C}, \mathbf{W})} \\ &= \prod_{j=1}^{n} \frac{f_{\vec{X}_j, Z_j}(\vec{x}_j, z_j; \mathbf{C}, \mathbf{W})}{f_{\vec{X}_j, Z_j}(\vec{x}_j, \overline{z_j}; \mathbf{C}, \mathbf{W})} \\ &= \prod_{j=1}^{n} \frac{p_{Z_j|\vec{X}_j}(z_j|\vec{x}_j; \mathbf{C}, \mathbf{W})}{p_{Z_j|\vec{X}_j}(\overline{z_j}|\vec{x}_j; \mathbf{C}, \mathbf{W})}. \end{aligned}$$

Obviously, this ratio can be seen as stating the **odds** of a correct classification with a classifier that predicts the class that yields the highest likelihood. The numerator of this ratio is the posterior probability of the correct class z_j given the data point $\vec{x}_j$. That is,

$$p_{Z_j|\vec{X}_j}(z_j|\vec{x}_j; \mathbf{C}, \mathbf{W}) = \frac{\sum_{y \in I(z_j)} f_{\vec{X}_j|Y,Z_j}(\vec{x}_j|y, z_j; \mathbf{C}, \mathbf{W})\, p_{Y,Z_j}(y, z_j; \mathbf{W})}{f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}, \mathbf{W})}.$$

Here $I(z) \subseteq \{1, \ldots, c\}$ is again, as on page 39 in Section 2.5, a set that contains the indices of those clusters that describe a mixture component of the z-th class. Therefore the numerator aggregates all probability density functions associated with the class z_j of the data point $\vec{x}_j$. Note that the probability distribution p_{Y,Z_j} has only $\mathbf{W}$ as a parameter, since the elements of this matrix, which are the coefficients in the (linear) combination functions, directly state these probabilities (cf. page 40 in Section 2.5).

The symbol $\overline{z_j}$ in the denominator is used to denote all possible class indices except z_j and thus the denominator is the likelihood of the data point in connection with an incorrect class. That is,

$$p_{Z_j|\vec{X}_j}(\overline{z_j}|\vec{x}_j; \mathbf{C}, \mathbf{W}) = \sum_{k \in \{1,\ldots,s\}-\{z_j\}} \frac{\sum_{y \in I(k)} f_{\vec{X}_j, Y|Z_j}(\vec{x}_k|y, k; \mathbf{C}, \mathbf{W})\, p_{Y,Z_j}(y, k; \mathbf{W})}{f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}, \mathbf{W})}.$$

If the clusters are partitioned onto the different classes (as it is usually the case), so that each cluster is associated with exactly one class, the two sums in this expression may conveniently be combined into one, ranging over $y \in \{1, \ldots, c\} - I(z_j)$. That is, one sums the probability densities at the data point $\vec{x}_j$ that result from all clusters *not* associated with the class z_j.

The denominator in both posterior probabilities considered above is the likelihood of seeing the data point $\vec{x}_j$, regardless of its class z_j, as it is specified by the full cluster model. That is,

$$f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}, \mathbf{W}) = \sum_{k \in \{1,\ldots,s\}} \sum_{y \in I(k)} f_{\vec{X}_j|Y,Z_j}(\vec{x}_k|y, k; \mathbf{C}, \mathbf{W})\, p_{Y,Z_j}(y, k; \mathbf{W}).$$

Again, if the clusters are partitioned onto the different classes, the two sums in this expression may conveniently be combined into one, ranging over $y \in \{1, \ldots, c\}$, that is, over all clusters. Note, however, that this probability cancels when forming the likelihood ratio. Therefore we have

$$L^{(1)}_{\text{ratio}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) = \prod_{j=1}^{n} \frac{\sum_{y \in I(z_j)} f_{\vec{X}_j|Y,Z_j}(\vec{x}_j|y, z_j; \mathbf{C}, \mathbf{W})\, p_{Y,Z_j}(y, z_j; \mathbf{W})}{\sum_{k \in \{1,\ldots,s\}} \sum_{y \in I(k)} f_{\vec{X}_j|Y,Z_j}(\vec{x}_k|y, k; \mathbf{C}, \mathbf{W})\, p_{Y,Z_j}(y, k; \mathbf{W})}.$$

However, the fact that the above likelihood ratio is unbounded (i.e., in principle it can go to infinity), can lead to technical problems in the optimization process (cf. Section 5.3.4). Therefore it is often preferable to maximize the ratio of the likelihood of a correct classification to the likelihood of seeing the data point [Seo and Obermayer 2003]. In this case one tries to maximize

$$\begin{aligned} L^{(2)}_{\text{ratio}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) &= \prod_{j=1}^{n} \frac{L(\vec{x}_j, z_j; \mathbf{C}, \mathbf{W})}{L(\vec{x}_j; z_j; \mathbf{C}, \mathbf{W}) + L(\vec{x}_j; \overline{z_j}; \mathbf{C}, \mathbf{W})} \\ &= \prod_{j=1}^{n} \frac{L(\vec{x}_j, z_j; \mathbf{C}, \mathbf{W})}{L(\vec{x}_j; \mathbf{C}, \mathbf{W})} \end{aligned}$$

$$= \prod_{j=1}^{n} \frac{f_{\vec{X}_j,Z_j}(\vec{x}_j, z_j; \mathbf{C}, \mathbf{W})}{f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}, \mathbf{W})}$$

$$= \prod_{j=1}^{n} p_{Z_j|\vec{X}_j}(z_j|\vec{x}_j; \mathbf{C}, \mathbf{W}).$$

Note that this second likelihood ratio, which is equivalent to the posterior probability of the correct classes, is obviously bounded by 1 from above, because the posterior probabilities are all no greater than 1. As already mentioned above, this is important for the optimization procedure, because an unbounded ratio can lead to technical problems, in particular "run-away" effects for the cluster parameters (diverging cluster centers). These effects can make it necessary to introduce restrictions into the update process, which are a little difficult to justify from a probabilistic point of view.

Note also that it is often technically easier to optimize the (natural[6]) logarithm of these ratios (as for the likelihood considered in the preceding section). Again this only changes the *value* of the maximum, but not its *location*. However, it enables us to write the first ratio as

$$\begin{aligned}
&\ln L^{(1)}_{\text{ratio}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) \\
&= \sum_{j=1}^{n} \ln p_{Z_j|\vec{X}_j}(z_j|\vec{x}_j; \mathbf{C}, \mathbf{W}) - \sum_{j=1}^{n} \ln p_{Z_j|\vec{X}_j}(\overline{z_j}|\vec{x}_j; \mathbf{C}, \mathbf{W}) \\
&= \sum_{j=1}^{n} \ln f_{\vec{X}_j,Z_j}(\vec{x}_j, z_j; \mathbf{C}, \mathbf{W}) - \sum_{j=1}^{n} \ln f_{\vec{X}_j,Z_j}(\vec{x}_j, \overline{z_j}; \mathbf{C}, \mathbf{W})
\end{aligned}$$

and the second as

$$\begin{aligned}
&\ln L^{(2)}_{\text{ratio}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) \\
&= \sum_{j=1}^{n} \ln p_{Z_j|\vec{X}_j}(z_j|\vec{x}_j; \mathbf{C}, \mathbf{W}) \\
&= \sum_{j=1}^{n} \ln f_{\vec{X}_j,Z_j}(\vec{x}_j, z_j; \mathbf{C}, \mathbf{W}) - \sum_{j=1}^{n} \ln f_{\vec{X}_j}(z_j; \mathbf{C}, \mathbf{W}).
\end{aligned}$$

Finally, note that there is a close connection of this approach to learning vector quantization for classification, since it can be seen as limit case of a Gaussian mixture model approach with classes [Seo and Obermayer 2003]. Details about this view can be found in Section 5.3.

[6] The base of the logarithm is actually irrelevant as it only introduces a constant factor.

3.5 Other Approaches

The approaches considered in the preceding four sections can nicely be categorized into those based on minimizing (squared) deviations (distances or errors) and those based on maximizing likelihood (or likelihood ratio). In both of these categories we find approaches for clustering as well as classification and thus they cover the major directions of tackling clustering and classification problems with prototypes.

However, for **classification** there is another highly popular approach, which is also based on (point-)prototypes and distance measures, but which does not fall into one of these categories. This approach is the so-called **maximum margin classifier**, which is closely connected to the theory of **support vector machines** (SVMs) [Vapnik 1995, Vapnik 1998, Cristianini and Shawe-Taylor 2000, Schölkopf and Smola 2002]. Although this classifier works directly only for two classes, it can be extended to multiple classes by building several binary classifiers that separate one class from all others or discriminate pairs of classes. The results of these classifiers are then combined, for example, by simply deciding on the class yielding the highest score in all pairwise separations of one class from all others.

The idea underlying the maximum margin classifier is to map the data points in such a way into some high-dimensional space that the (two) classes become **linearly separable**, i.e., can be separated by a linear function in the coordinates describing a data point. The classifier is then basically a (hyper-)plane that separates the classes in such a way that the **margin**, that is, the smallest distance of a data point to the (hyper-)plane, is maximized (hence the name *maximum margin classifier*).

The data points that define the margin, that is, the data points whose images are closest to the separating (hyper-)plane, are called the **support vectors**. Their number depends on the mapping to the high-dimensional space, in particular the number of dimensions of that space, and the location of the data points. The support vectors take the place of the prototypes in the classification schemes studied above and explain the name **support vector machine** for a classifier based on this approach.

Since it is not always easy to find a mapping so that the classes become linearly separable, especially if the mapping function is restricted to come from a limited class of functions, the maximum margin classifier has been generalized to work with violations of a perfect separation. This generalization, which is known as the **soft margin approach**, is brought about by specifying a **target margin**, that is, a minimum distance all data points should have from the separating (hyper-)plane. In addition, a so-called

slack variable is introduced for each data point, which measures by how much the data point fails to meet the target, i.e., by how much it is closer to the separating (hyper-)plane (or even to the wrong side of it). The task is then to minimize the sum of the values of the slack variables.

Although the maximum margin classifier appears to be very similar to a prototype-based classifier, especially, since the support vectors seem to be analogs of the (point) prototypes, I do not consider support vector machines in this thesis. The reason is that the support vectors do not (or only by accident) provide any information about the distribution of the data points in the sense of a cluster prototype, which describes a group of similar data points. They only define the location of the separating (hyper-)plane (the classification boundary) by being those data points whose images have a minimal distance to this (hyper-)plane. Thus they are actually likely to be extreme points, fairly far away from the majority of the data points, and thus not "prototypical" for the data set in an intuitive sense.

When it comes to **clustering**, some alternative methods do not rely on an explicit objective function to define the goal and success of a clustering algorithm (cf. the first two sections of this chapter) and to derive the parameter update procedure (cf. Chapter 5). They rather define an update scheme for the cluster parameters directly and take the convergence point of this update scheme (if it exists) as the result of the clustering algorithm. An example of such an approach is **alternating cluster estimation** (ACE) [Runkler and Bezdek 1999, Höppner *et al.* 1999], which is closely related to fuzzy clustering, but does not try explicitely to optimize a given objective function. As a consequence the resulting algorithms are often much more flexible than their objective function based counterparts.

In this thesis I make use of an analogous approach in Chapter 6, where I study extensions of the basic update procedures discussed in Chapter 5. Since certain frame conditions are sometimes difficult to incorporate into an objective function (for example, restrictions of the ranges of values for some cluster parameters), I rely on an objective function only to derive a basic update scheme, the steps of which are then modified in order to satisfy given constraints or to speed up the clustering process.

A frequently heard objection against a clustering approach that abandons the strict goal of optimizing an objective function is that it may fail to reach a stable point, while convergence is (almost) guaranteed if the update scheme is derived from an objective function. However, such an objection conveys, in my opinion, a distorted picture of the situation. In the first place, although some update schemes have been shown to converge reliably (like, for example, the fuzzy c-means algorithm [Bezdek 1980, Selim and

Ismael 1984, Ismael and Selim 1986, Bezdek *et al.* 1987]), such proofs are lacking for more flexible approaches, even though they are based on an objective function. Furthermore, even if an update procedure can be shown to converge theoretically, it may not do so in practice due to numerical problems like roundoff errors. Finally, for a large variety of update schemes convergence problems are very rare or even non-existent in practice and they usually lead to very good results, and very often to even better results than the methods that are derived directly from an objective function.

Chapter 4

Initialization Methods

As already mentioned, the majority of prototype-based clustering and classification algorithms are iterative in nature. That is, they employ an update scheme, which improves the parameters of the clusters and the classification functions step by step. Such an iterative improvement is usually necessary even if the goal of the clustering or classification method is specified as the optimization of an objective function, because the objective functions discussed in the preceding chapter are difficult to optimize directly.

For an iterative improvement algorithm two things are needed: (1) a method to initialize the parameters and (2) a method to update these parameters to new values, which is then applied repeatedly until some termination criterion is met. In this chapter I study the former, while the next chapter is devoted to update methods (derived from objective functions).

The initialization methods examined in this chapter focus on the initialization of the cluster centers, because they are most difficult to choose appropriately. I categorize the discussed methods as *data independent* (the locations of the data points are completely ignored, Section 4.1), *simple data dependent* (the data points enter the initialization, but only in a simple way, Section 4.2) and *sophisticated* (approaches that can (almost) be seen as clustering algorithms in their own right, Section 4.3).

Parameters like the covariance matrices of the Mahalanobis distance, on the other hand, are most conveniently initialized to a unit matrix. The cluster sizes and weights are best set to equal values, with the size chosen w.r.t. the extension of the data space and the cluster number (e.g. $\sigma \leq \frac{D}{2c}$, where D is the diagonal of the data space). Only for classification a special weight initialization is recommended, which is reviewed in Section 4.4.

4.1 Data Independent Methods

All data independent initialization methods for cluster centers sample random points from a probability distribution on the data space. They only differ in the probability distribution they sample from and the exact procedure used for the sampling. For the choice of the probability distribution and its parameters these methods may rely on some information about the data points (like, for instance, the range of values they take in the different dimensions of the data space) in order to restrict the sampling to that region of the data space in which the data points are located. In this sense they may not be completely data independent, even though they do not consider the specific locations of individual data points.

The most common data independent initialization method for the cluster centers is to **sample from a uniform distribution** on the smallest (hyper-)box enclosing the data points. It has the advantage that it is very simple and efficient, since random numbers from a uniform distribution can easily be obtained in a computer. It is also highly likely that the initial cluster centers are well spread out over the data space, which is a desirable effect, as it increases the chances of detecting all clusters. If, in contrast to this, the initial positions of the cluster centers are close together, there is—at least with some methods—the danger that they share few data point clouds among them, with more than one cluster center per cloud, while other data point clouds are not covered by a cluster prototype.

The disadvantage of this method is that for high-dimensional spaces, in which the smallest bounding (hyper-)box is unlikely to be populated uniformly, the initial positions of the cluster centers can be very far away from the data points (the cluster centers may, by accident, be chosen in regions of the data space that are void of data points). However, this is a disadvantage of all data independent initialization methods, since they do not take any information about the location of the data points into account.

A slight variation of this most common method is to sample from a uniform distribution of user-specified extensions centered in the smallest (hyper-)box enclosing the data points. This has the advantage that the user can control how far spread out the initial cluster centers should be and can also compensate effects outliers may have on the bounding (hyper-)box. This method is best implemented by initializing all cluster centers to the center of the data space and then adding a uniformly distributed random offset of user-specified maximum size in each dimension. Adding a random offset to an initial choice is actually a good idea for many initialization methods, whether data dependent or data independent.

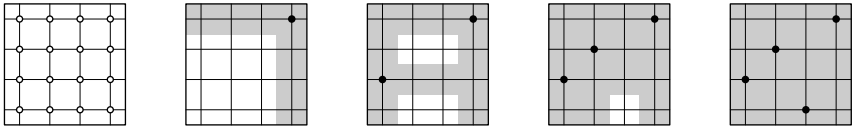

Figure 4.1: Latin hypercube sampling in two dimensions with $c = 4$.

An alternative to sampling from a uniform distribution is to **sample from a normal distribution**, which has the center of the data space as its excepted value and a user-specified variance. However, this method is less popular, since sampling from a normal distribution is more complicated and has few noticeable advantages over the simpler uniform sampling.

A very nice extension of the simple uniform sampling method is so-called **Latin hypercube sampling**, which aims at initial cluster centers whose coordinates are well spread out in the individual dimensions. Latin hypercube sampling was originally developed in statistics for situations, in which controlled experiments are to be carried out to obtain data about some domain, but in which each experiment is very expensive. Hence one has to plan the experiments in such a way that the parameter space is covered as well as possible with a very limited number of sample points.[1]

Latin hypercube sampling works as illustrated for two dimensions in Figure 4.1. In an initial step a grid on the data space is constructed by choosing c equidistant lines for each dimension (where c is the number of samples one plans to draw, and hence it is the number of clusters here). Figure 4.1 shows such a grid for $c = 4$ in the diagram on the very left.

The grid points are the sample candidates, which are chosen with equal probability. However, after a grid point (an initial cluster center) is selected, all points having the same coordinate in one or more dimensions are banned for future selection. This is shown in Figure 4.1 by grey bars marking the grid lines that are ruled out by previous selections: after the grid point at the top right is chosen, the first row and last column are ruled out (second diagram). The subsequent choice rules out the third row and the first column (third diagram) and so forth. As a consequence fewer and fewer grid points are available until the final choice is uniquely determined (since all other coordinates for each of the dimensions have been ruled out).

[1] Of course, in order to study some domain statistically, it is not a good idea to rely on very few sample points. However, if you can afford only very few experiments (like, for instance, crash tests with cars, varying the car type, passengers, direction of impact etc.), planning them with Latin hypercube sampling is often the best that one can do.

A special case of Latin hypercube sampling is to choose equidistant cluster centers on the **diagonal of the data space**. This case results if in the Latin hypercube scheme always the same grid coordinate is chosen for all dimensions. Seen the other way round, Latin hypercube sampling can be seen as a perturbation of (the coordinates of) such points on the data space diagonal. Actually one of the simplest ways of implementing Latin hypercube sampling is to start from equally spaced points on the diagonal of the data space and then to shuffle the coordinates of these points independently for each of the dimensions of the data space.

4.2 Simple Data Dependent Methods

As already pointed out in the preceding section, it is a drawback of all data independent initialization methods that the chosen initial cluster centers may be far away from the data points, since there is nothing to prevent these methods from choosing points in regions that are bare of data points. Better initial positions can usually be obtained by taking the locations of the data points into account, either in an aggregated form as a mean vector, or by using the data points directly.

The simplest data dependent initialization method is, of course, to draw random **samples from the data set** (without replacement). It has the obvious advantage that initial positions in empty regions of the data space are avoided, but suffers from the disadvantage that it is not too unlikely that the cluster centers are not properly spread out over the data set. This is especially true if the data set consists of data point clouds with considerably differing numbers of data points. In this case it is likely that a high number of initial cluster centers are chosen in the densely populated regions, while less densely populated regions tend to be neglected.

A sampling method that was explicitely devised to achieve a good coverage of the data set, and thus to overcome the drawbacks of a simple random sampling, is a method known as **Maximindist** [Bachelor and Wilkins 1969]: The first cluster center is either chosen randomly from the data set or initialized to the center of the data space or the mean of all points in the data set (the first is the most common). In subsequent steps one chooses the data point, whose *minimal distance* to all already chosen cluster centers is *maximal* (hence the name *Maximindist*: maximize minimal distance).

An obvious advantage of this approach is that it avoids initial cluster centers that are close together, because it tries to maximize their distance. In addition, the sequence of maximized minimal distances can be used to

guess the number of clusters: if this distance falls significantly from one selection step to the next, it is likely that with the last selection one exceeded the number of clusters that can reasonably be found in the data set. Hence one should retract one step and remove the last chosen cluster center.

A severe disadvantage of *Maximindist* is that it is fairly sensitive to outliers. Since outliers have a large distance to all other data points in the data set, the procedure tends to choose them and even tends to choose them early. Of course, this is not desirable, because outliers are definitely not prototypical for any clusters at all and thus are very bad starting points for a clustering algorithm. In addition, if the sequence of maximized minimal distances is studied to guess the number of clusters, outliers can convey a wrong impression. Furthermore, *Maximindist* is a fairly expensive initialization method, at least when compared to a simple random sampling, because $c \cdot n$ distances have to be computed (where c is the number of clusters and n is the number of data points, i.e. the size of the data set).

A different way of taking the specific locations of the data points into account, which is less sensitive to outliers than both random sampling and *Maximindist*, is to estimate the parameters of a **simple probabilistic model** of the data and then to sample from this model. If the task is clustering, one may estimate, for example, the parameters of a single multivariate normal distribution from the data. For classification tasks, one multivariate normal distribution for each of the classes may be determined. Such an approach is equivalent to building a (naïve or full) Bayes classifier for the data and then sampling from this Bayes classifier. Since one may subsume the initialization for a clustering task under this approach by viewing the data set as classified, but having only one class, I confine myself to explaining this approach for classified data.

Let us start by recalling the ideas underlying a (naïve or full) **Bayes classifier** [Good 1965, Duda and Hart 1973, Langley *et al.* 1992] for purely numeric data. A Bayes classifier tries to maximize the probability of a correct class assignment by trying to predict the most probable class. To do so, it estimates the **posterior probabilities** $p_{Z|\vec{X}}(k|\vec{x})$ for all $k \in \{1, \ldots, s\}$, where s is the number of classes (i.e., the probabilities of the classes *after* observing the data point $\vec{x}$). As in Chapter 2, and in particular on page 40 in Section 2.5, $\vec{X}$ is a random vector that has the data space as its domain and Z is a random variable that has the different class indices as possible values. A Bayes classifier predicts the class for which (the estimate of) this posterior probability is maximal, i.e., it predicts the class

$$k^* = \underset{k \in \{1,\ldots,s\}}{\operatorname{argmax}} p_{Z|\vec{X}}(k|\vec{x}).$$

However, it is clear that it is practically infeasible to estimate the posterior probabilities $p_{Z|\vec{X}}(k|\vec{x})$ directly for each point $\vec{x}$ of the data space. Therefore a Bayes classifier exploits, in a first step, Bayes' rule to represent them as

$$p_{Z|\vec{X}}(k|\vec{x}) = \frac{f_{\vec{X}|Z}(\vec{x}|k)\, p_Z(k)}{f_{\vec{X}}(\vec{x})} = \frac{f_{\vec{X}|Z}(\vec{x}|k)\, p_Z(k)}{\sum_{i=1}^{s} f_{\vec{X}|Z}(\vec{x}|i)\, p_Z(i)}.$$

Since the denominator of the rightmost fraction can be computed once we determined the numerators for all $k \in \{1, \ldots, s\}$, we may neglect it and see it as a normalization factor that is computed in the end by scaling the numerators so that they add up to 1 over the different classes.

For computing the first factor in the numerators, one usually assumes, as already mentioned above, that the conditional probability of the data points given a class is an m-dimensional normal distribution. That is,

$$f_{\vec{X}|Z}(\vec{x}|k; \vec{\mu}_k, \mathbf{\Sigma}_k) = \frac{1}{\sqrt{(2\pi)^m |\mathbf{\Sigma}_k|}} \cdot \exp\left(-\frac{1}{2} d^2(\vec{x}, \vec{\mu}_k; \mathbf{\Sigma}_k)\right).$$

Here $\vec{\mu}_k$ is the expected value vector and $\mathbf{\Sigma}_k$ the covariance matrix associated with class k. They are estimated from the data set by

$$\hat{\vec{\mu}}_k = \frac{\sum_{j=1}^{n} \delta_{k,z_j}\, \vec{x}_j}{\sum_{j=1}^{n} \delta_{k,z_j}} \qquad \text{and} \qquad \hat{\mathbf{\Sigma}}_k = \frac{\sum_{j=1}^{n} \delta_{k,z_j} (\vec{x}_j - \hat{\vec{\mu}}_k)(\vec{x}_j - \hat{\vec{\mu}}_k)^\top}{\sum_{j=1}^{n} \delta_{k,z_j}},$$

where the z_j, $1 \leq j \leq n$, state the indices of the classes associated with the data points $\vec{x}_j \in \mathbf{X}$ (cf. Section 3.2 and 3.4). $\delta_{k,z}$ is the *Kronecker symbol* (cf. page 60 in Section 3.2 for a definition), which is used to restrict the sums to those data points that are associated with the class k under consideration. As is common notation in statistics, the hat symbol ($\hat{}$) indicates that the quantities are estimates of the distribution parameters.

The second factor in the numerator of Bayes' rule, which states the **prior probability** $\varrho_k = p_Z(k)$ of class k (i.e., the probability of class k *before* observing the data point), is estimated as

$$\hat{p}_Z(k) = \hat{\varrho}_k = \frac{1}{n} \sum_{j=1}^{n} \delta_{k,z_j},$$

i.e., as the relative frequency of class k. As a result we obtain a probabilistic classifier that has the same structure as those described in Section 2.5. The only difference is that a Bayes classifier uses only one (normal distribution) cluster per class, while the classifiers discussed in Section 2.5 are more general and may comprise several clusters per class.

iris type	iris setosa	iris versicolor	iris virginica
prior probability	0.333	0.333	0.333
petal length	1.46 ± 0.17	4.26 ± 0.46	5.55 ± 0.55
petal width	0.24 ± 0.11	1.33 ± 0.20	2.03 ± 0.27
covariance	0.0065	0.0640	0.0673

Table 4.1: The parameters of a naïve and a full Bayes classifier for the iris data. The parameters of the normal distribution for the individual dimensions are stated as $\hat{\mu} \pm \hat{\sigma}$ (i.e. expected value $\pm$ standard deviation). The last row states the covariances needed for a full Bayes classifier.

If full covariance matrices $\mathbf{\Sigma}_k$, $1 \leq k \leq s$, are used, the classifier is called a **full Bayes classifier**. A common alternative is the so-called **naïve Bayes classifier**, which makes the additional assumption that the dimensions of the data space are conditionally independent given the class. That is, it is assumed that the conditional probability of a data point $\vec{x} = (x_1, \ldots, x_m)$ given any class $k \in \{1, \ldots, s\}$ can be written as

$$f_{\vec{X}|Z}(\vec{x}|k) = \prod_{i=1}^{m} f_{X_i|Z}(x_i|k).$$

It is easy to see that such an assumption is equivalent to assuming that all covariance matrices are diagonal, i.e.,

$$\mathbf{\Sigma}_k = \mathrm{diag}(\sigma_{k1}^2, \ldots, \sigma_{km}^2),$$

since this allows us to decompose the Mahalanobis distance accordingly. The elements of $\mathbf{\Sigma}_k$ may then be estimated from the data set by

$$\hat{\sigma}_{ki} = \frac{\sum_{j=1}^{n} \delta_{k,z_j} (x_{ji} - \hat{\mu}_{ki})^2}{\sum_{j=1}^{n} \delta_{k,z_j}},$$

where $\vec{x}_j = (x_{j1}, \ldots, x_{jm})$ and $\vec{\mu}_k = (\mu_{k1}, \ldots, \mu_{km})$, with the latter estimated in the same way as for a full Bayes classifier (see above).

As an illustrative example I consider the well-known iris data [Anderson 1935, Fisher 1936, Blake and Merz 1998]. The classification problem is to predict the iris type (iris setosa, iris versicolor, or iris virginica) from measurements of the sepal length and width and the petal length and width. Here, however, I confine myself to the latter two measures, which are the

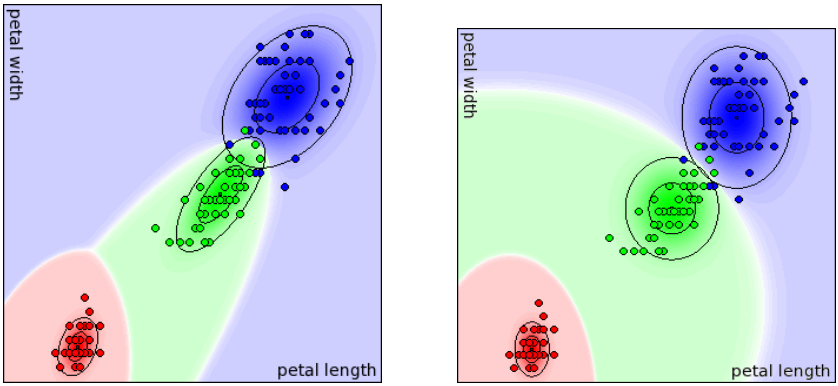

Figure 4.2: The probability density functions used by a full Bayes classifier (general ellipses, left) and a naïve Bayes classifier (axis-parallel ellipses, right) for the iris data. The ellipses are the 1σ- and 2σ-boundaries.

most informative w.r.t. a prediction of the iris type. (In addition, one cannot visualize a four-dimensional space.) The parameters of the two types of Bayes classifiers as they can be estimated from these two measures and all 150 cases (50 cases of each iris type) are shown in Table 4.1.

The conditional probability density functions used to predict the iris type are shown in Figure 4.2 [Borgelt *et al.* 2001]. The ellipses are the 1σ- and 2σ-boundaries of the (bivariate) normal distributions. The left diagram shows the full Bayes classifier, with ellipses in general orientation, while the right diagram depicts the naïve Bayes classifier, with diagonal covariance matrices and thus axes-parallel ellipses (cf. Section 2.1). It is easy to see that the full Bayes classifier captures the data much better, which is confirmed by the number of misclassifications: while the full Bayes classifier misclassifies only three example cases, the naïve Bayes classifier leads to six errors.

After a Bayes classifier is constructed, the cluster centers are initialized by sampling from the estimated normal distributions, either choosing the same number of centers for each class or a number that reflects their prior probability. If one desires to construct a classifier with one cluster per class, the Bayes classifier—including covariance matrices and cluster weights—may be used directly as a starting point [Nürnberger *et al.* 1999]. However, the covariance matrices may also be used for initialization if multiple clusters per class are used. In this case it may be advisable, though, to reduce their size by multiplying them with a factor less than 1.

4.3 More Sophisticated Methods

The data dependent methods discussed in the preceding section exploited the information contained in the data set only in a very simple manner, sampling from the data set or estimating a simple probabilistic model from it. In contrast to this, the more sophisticated methods I review in this section are much more complex and can even be seen as clustering methods in their own right. As a consequence they are computationally considerably more expensive and therefore one has to check carefully whether they improve the quality of the initialization enough to warrant the additional costs that are incurred. This is especially true if one of the simpler clustering approaches (like fuzzy c-means clustering) is employed, because these are very efficient and usually also not so sensitive to the initialization.

In the following I consider first initialization methods for clustering and later for classification purposes. Although the former methods can always be used for the latter task by splitting the data set w.r.t. the classes and then finding initial clusters for each of the classes separately, there are some approaches that work on the complete data set and are geared to find clusters that yield a good classification. It should also be noted that I do not claim to provide a complete coverage of the methods of either type, as there are too many special data dependent approaches to clustering and classification that may be used to initialize the prototype-based methods discussed in this thesis. Rather I try to focus on some core ideas, exemplified by typical representatives of methods derived from them.

For the initialization of **clustering** algorithms, one class of initialization approaches is based on the idea to analyze of the structure of a non-parametric estimate of the probability density function that generated the data set. In particular, one carries out a kernel-based estimation and studies the structure of the modes of the resulting density estimate. Important representatives of such an approach are **mountain clustering** [Yager and Filev 1994] and **subtractive clustering** [Chiu 1994], which differ only in the set of candidate points that is considered for the initial positions of the cluster centers: while in mountain clustering a regular grid is defined on the data space (like for Latin hypercube sampling, cf. Section 4.1), in subtractive clustering one relies on the points in the given data set.

Once the set of candidate cluster centers is fixed, both approaches work in basically the same way. First a user has to choose a **kernel function** for a non-parametric estimation of the probability density function. This kernel function is usually a radial function defined on a distance measure (most often the Euclidean distance), as it was studied in Section 2.1.

The idea underlying such a kernel function is that in continuous domains one cannot estimate probabilities by simply computing relative frequencies, as one may for estimating the probabilities of a multinomial distribution on a finite domain. The reason is simply that in a continuous domain it is highly likely that all data points differ in their values. Of course, one can always discretize a continuous domain, thus forming bins, for which a relative frequency estimation works again. However, such an approach can introduce strange effects resulting from the positioning of the grid, which determines which data points fall into the same bin.

A very natural and straightforward approach to cope with such effects is to estimate the probability in a grid cell not only from the data points that fall into it. Rather one also takes into account the data points falling into neighboring cells, making sure that these points have a smaller influence than those located in the cell itself. In other words, one uses overlapping bins for the estimation. Of course, with such a scheme one has to introduce a proper weighting of the data points, so that each of them has the same influence on the whole estimation. That is, a unit weight for each data point should be distributed to the different estimations it enters.

Such a distribution of the unit weight of a data point can very naturally be described by a radial step function on the maximum distance, which is centered at the data point, provided the radial function is normalized to integral 1. However, we may also generalize this approach, doing without bins defined by a grid, by specifying a general radial function (though still normalized to integral 1) on a general distance measure—the abovementioned *kernel function*. With such a kernel function we may estimate the probability density at arbitrarily chosen points of the data space, by simply summing the corresponding values of the kernel functions of all data points. This is the basic idea of **kernel estimation** [Scott 1992, Everitt 1998], which goes back to the idea of **Parzen windows** [Parzen 1962].

Formally, a one-dimensional probability density f is estimated as

$$\hat{f}(x) = \frac{1}{nw} \sum_{j=1}^{n} K\left(\frac{d(x, x_j)}{w}\right),$$

where the kernel function K satisfies $\int_{\mathrm{dom}(t)} K(t)\mathrm{d}t = 1$, i.e., it is normalized to integral 1, and w is called the **window width** or **bandwidth** [Everitt 1998]. Intuitively, a kernel estimator sums a series of "bumps" located at the data points. The kernel function K specifies the shape of these bumps and the window width w determines their extension or size. One of the most common kernel functions is the Gaussian function (cf. Section 2.2).

It should be noted that due to the symmetry of the distance measure the computation of the estimate may also be seen as placing the kernel function at the point where the density is to be estimated and evaluating it at the locations of the data points. Furthermore, it should be noted that the estimated density function is the smoother, the larger the value of the window width w. Finally, it is clear that the window width w can be incorporated into the kernel function (or rather into the distance measure), provided it is made sure that the integral of K over the whole data space remains 1 (the factor $\frac{1}{w}$ in the estimation formula only accounts for the change of the integral that is brought about by the division of the distance by w and rescales K to a unit integral over the data space—cf. Section 2.2 and Appendix A.2.1, where such a renormalization is discussed for the more general case of a covariance matrix entering a Mahalanobis distance).

Based on the idea of kernel estimation, *mountain clustering* [Yager and Filev 1994] and *subtractive clustering* [Chiu 1994] can be described as follows: The kernel estimate can be seen intuitively as a "mountain range" (from this view the name of the former method is derived). Clusters should be located at the highest peaks, provided these peaks are not too close together (we do not want cluster centers on all the side peaks of the highest elevation). Unfortunately, it can be mathematically tricky to find the modes (peaks, local maxima) of a kernel estimated density function [Fukunaga 1990]. Therefore one considers only a limited set of points, namely the chosen set of candidate cluster centers and estimates the probability density only at these points. Next one finds the candidate cluster center with the highest probability density and chooses it as the first cluster center.

For consecutive choices of cluster centers we have to take precautions that side peaks of already chosen peaks are penalized. This is achieved by an operation that [Yager and Filev 1994] call the "destruction of a mountain." A kernel function, which is multiplied with a certain weight, which accounts for the number of data points one desires or expects to find in a cluster, is placed at the location of the chosen cluster center. It is evaluated at the locations of the remaining candidate cluster centers, and the resulting value is subtracted from the corresponding density estimate. Usually the kernel function used for the "destruction of a mountain" is the same as the one used for the estimation in the first place and thus the only parameter is the weighting factor, by which it is multiplied. After the destruction of a mountain, the highest peak of the modified estimate is determined, chosen as the next cluster center, its corresponding mountain is "destructed" and so on. The process stops when a predefined number of cluster centers has been selected or the highest density value lies below a specified threshold.

Depending on the chosen weighting factor for the "destruction of a mountain" it can happen that the maximum of the modified density estimate lies at an already chosen cluster center. In this case it is usually best to destruct the corresponding mountain without choosing another cluster, because simply excluding already chosen cluster centers bears a considerable danger that undesired side peaks of a high elevation get selected.

Although *mountain clustering* and *subtractive clustering* are fairly intuitive methods with some sound basis in statistics, it should be kept in mind that they are computationally expensive as they require the computation of a large number of distances (the number of data points times the number of candidate cluster centers). Thus its computational costs can easily exceed the costs for iterating the update scheme of a simple clustering algorithm, for example, of fuzzy c-means clustering, until convergence (cf. Section 5.2). On the other hand, an advantage of these approaches is that they can be used to guess the number of clusters, namely by specifying a threshold for the density estimate at which no more cluster centers get selected. As an alternative one may first select center candidates with the mode-finding method by [Schnell 1964], which however, is itself an expensive procedure.

The second major initialization method for clustering I consider here is the well-known approach of **hierarchical agglomerative clustering** [Sokal and Sneath 1963, Johnson 1967, Bock 1974, Mucha 1992]. As its name already indicates, this method constructs a hierarchy of clusters, that is, each cluster consists of subclusters. The hierarchy is formed in an agglomerative fashion: Initially each data point forms a cluster of its own. The clustering process iteratively merges those clusters that are closest to each other, until only one cluster, covering all data points, remains.

The obvious problem residing with such an approach is that as long as a cluster contains only one data point, we can easily determine the distance of two clusters. However, as soon as clusters have been merged, and thus contain multiple data points, how should we measure their distance? A large variety of so-called "**ultra metrics**" has been suggested: **single linkage** — smallest distance between a data point in one and a data point in the other cluster, **complete linkage** — largest distance between two data points, **average linkage** — average distance between two data points, **centroid method** — distance between the centroids (mean vectors) of the clusters etc. (see [Bock 1974] for a quite complete list). However, among these methods only those that lead to compact clusters are useful for initializing prototype-based clustering . This rules out single linkage, because the clusters found with this methods may be long chains of data points, but *complete linkage*, *average linkage*, and the *centroid method* are good choices.

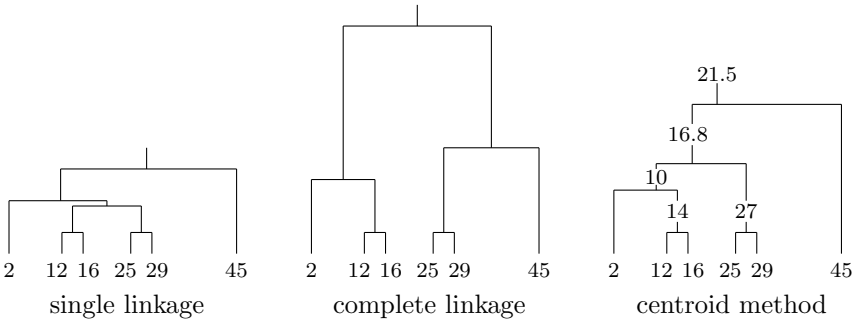

Figure 4.3: Dendrograms of cluster hierarchies built with hierarchical agglomerative clustering for a simple one-dimensional data set with three different methods of measuring the distance between clusters.

A method that is particularly recommended for the initialization of prototype-based approaches is **Ward's method** [Ward 1963]: in each step merge those two clusters that lead to the lowest increase in the average intra-cluster variance. This variance is the weighted average (weighted with the number of data points) of the equivalent isotropic variances of each cluster (cf. Section 2.3). The method is well-suited for the initialization of crisp and fuzzy clustering, because it uses basically the same objective function: the sum of squared distances to cluster centers, which are the mean vectors of the assigned data points, is obviously proportional to the average equivalent isotropic variance. Due to the close connection of a clustering approach based on the minimization of squared distances to maximization of the likelihood with a Gaussian mixture model (cf. page 64 in Section 3.3), it is also a good choice for initializing maximum likelihood approaches.

As already mentioned, consecutively merging clusters leads to a hierarchy of clusters. This hierarchy is usually depicted as a so called **dendrogram** (from the Greek $\delta\acute{\varepsilon}\nu\tau\varrho\omega\nu$: tree), in which merged clusters are connected by a bridge, the height of which represents their distance. Examples of dendrograms for a one-dimensional data set and three different methods of measuring the distances between clusters are shown in Figure 4.3.

After the cluster hierarchy has been constructed, the initial clusters are determined as follows: if the number c of clusters to be used in the subsequent prototype-based clustering is already known, one simply cuts the dendrogram horizontally in such a way that c branches are cut. (Equivalently one may stop merging clusters once their number has been reduced to c.)

However, the sequence of cluster distances in the mergers carried out to form the cluster hierarchy may also be analyzed to guess a proper cut and thus the number of clusters. An overview of methods to determine the number of clusters in this way can be found, for example, in [Kirsten 2002].

Each branch of the cut dendrogram represents one cluster, with a crisp assignment of data points. To find the corresponding center, one simply computes the mean vector of the data points assigned to a cluster. In addition, one may compute a covariance matrix for each of the clusters to initialize a Mahalanobis distance and/or the relative number of data points in the cluster to initialize the cluster weight (if these parameters are used).

Although it can yield very good initializations and can be used to guess the number of clusters (see above), hierarchical agglomerative clustering suffers from the same drawbacks as mountain clustering and subtractive clustering, namely that it is fairly expensive. It starts from a distance matrix for all data points (containing all pairwise distances) and modifies this matrix according to the cluster mergers that are carried out. As a consequence, a large number of distance computations (quadratic in the size of the data set) is necessary, and thus one should check carefully if the better initialization outweighs the incurred costs.

A closely related, but computationally much cheaper method was suggested by [MacQueen 1967]. It can be seen as using a blend of ideas from hierarchical agglomerative clustering and classical c-means clustering and consists in traversing the data set once in a randomly chosen order. The first c data points, where c is the desired number of clusters, are chosen as initial clusters, containing one data point each. When traversing the remaining data points one always finds the cluster that is closest to the new data point w.r.t. the centroid method and merges the new data point with it (a procedure that can be seen as "online" c-means clustering, cf. Section 5.2.2), computing a new cluster center.

Since the cluster centers move during the procedure, it is likely that after some steps the cluster, a data point was assigned to originally, is no longer the one yielding the center closest to the data point. Therefore [MacQueen 1967] suggested to recompute the assignment of all already processed data points as well as the resulting centers in each step. However, this renders the method very expensive computationally and hence, since I am only concerned with initializing the centers and not with finding a final clustering result, it is recommended to skip this re-assignment. Without this reassignment, the costs of the whole process are only slightly higher than those of one update step of classical c-means clustering (cf. Section 5.2.2) and hence it is well-suited even for initializing very simple clustering algorithms.

The process stops with an initial crisp partition of the data points after all data points have been processed. If one skipped the re-assignment in the individual steps, one may still carry out a final re-assignment of the data points to stabilize the result. The mean values of the clusters in the final partition may then be used to initialize the cluster centers of a prototype-based algorithm. In addition, initial values for the cluster-specific covariance matrices and the cluster weights may be determined.

As a final initialization method for clustering I would like to mention the **refinement approach** by [Fayyad *et al.* 1998]. It starts from the following observation: basically all update schemes that are discussed in the next chapter suffer from the problem of local optima. That is, the iterative improvement of the cluster parameters can get stuck in a local optimum of the objective function, thus leading to suboptimal results. The most common solution to this problem is to do several clustering runs with different initializations, yielding several clustering results. Afterwards the result having the best value for the objective function is determined and chosen as the final result [Duda and Hart 1973]. However, this approach, though usually very effective, suffers from the drawback that it is computationally fairly expensive, because the full clustering algorithm—initialization as well as iterative updating until convergence—has to be carried out several times.

As an alternative [Fayyad *et al.* 1998] suggest to refine the initialization based on the clustering results. However, since it is too costly to run the algorithm several times on the full data set, especially if this data set is large, they suggest to use clustering results obtained on (randomly chosen) subsets of the full data set. The method works as follows: several relatively small random samples are drawn from the given data set and clustered with a prototype-based clustering algorithm, in which only cluster centers are used. In this procedure the algorithm is initialized with some simple method, for instance, by randomly selecting data points as initial cluster centers. The resulting cluster centers are then pooled to form a new data set, which again is clustered with the prototype-based algorithm, doing several runs with different initializations if necessary. The cluster centers that result from "clustering the clustering results" are finally used to initialize the clustering algorithm for the full data set, which is carried out only once.

[Fayyad *et al.* 1998] report several experiments, which indicate that the refinement approach often yields results that are superior to those obtained with multiple runs and selecting the best result. The main advantages are the fairly small computational costs, which can also be controlled by the size and the number of the samples that are drawn, and the fact that it can be combined with any clustering algorithm that uses only cluster centers.

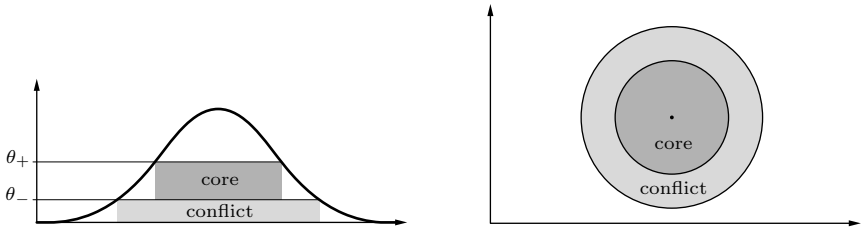

Figure 4.4: Illustration of how the thresholds θ_- and θ_+ of the dynamic decay adjustment algorithm define the conflict and the core region.

When it comes to initializing cluster prototypes for **classification** purposes, the first thing to notice is that all approaches discussed above can be applied as well. One simply splits the data set according to the class labels of the data points and then executes the initialization methods on each of these subsets. However, since with such an approach the cluster prototypes of each class are chosen without taking the location of data points of the other classes into account, their positions may not be particularly well suited for the task of classifying them. Therefore approaches that work on the undivided dataset are to be preferred, two of which I discuss here.

The first is the **dynamic decay adjustment (DDA) algorithm** [Berthold and Diamond 1995, Berthold and Diamond 1998] for constructively learning a special type of radial basis function network (cf. page 38 in Section 2.5). The cluster prototypes in this network are usually Gaussian radial functions over a Euclidean distance (although other radial functions and distance measures may be used as well), and the data points are classified by the nearest prototype or maximum membership scheme (cf. page 38 in Section 2.5). The dynamic decay adjustment algorithm selects cluster centers from the data points and dynamically adjusts the "decay rate" of the radial functions (hence its name) by adapting their reference radius σ individually for each cluster prototype.

The decay adjustment is based on two parameters that have to be specified by a user, namely a lower threshold θ_- and an upper threshold θ_+ for the membership of a data point to a cluster prototype. These parameters are illustrated in Figure 4.4, which shows a side view of the one-dimensional case on the left and top view of the two-dimensional case on the right. The grey shading shows the regions defined by the thresholds. The idea of the algorithm is to select centers and adjust their radii in such a way that

- there is no data point with a different class label inside the region defined by the lower membership threshold θ_- (the *conflict region*),
- each data point is inside the region defined by the upper membership threshold θ_+ (the *core region*) of at least one prototype having the same class label as the data point.

To achieve this, the dynamic decay adjustment algorithm works in an online fashion, that is, it considers the data points one after the other, traversing the data set several times if necessary. In each step one or both of two possible update operations are carried out. These operations are:

- **commit**: If the considered data point is not inside the core region of any cluster prototype having the same class label, a new cluster prototype with the class label of the data point is created, which is located at the data point and has a default reference radius σ.
- **shrink**: If the considered data point lies inside the conflict region of one or more cluster prototypes that are associated with different class labels, the reference radii σ of these prototypes are shrunk until the data point lies outside the conflict region(s).

Note that both operations can make it necessary to traverse the dataset again: The commit operation may have introduced a cluster that covers—with its conflict region—an already processed data point of a different class. The shrink operation may leave already processed data points uncovered that were covered when they were processed.

However, it can be shown that the dynamic decay adjustment algorithm converges after a finite number of steps, provided there are no contradictory example cases (that is, identical data points with differing class labels). With the recommended values of $\theta_- = 0.2$ and $\theta_+ = 0.4$ for a Gaussian radial function, convergence is usually very fast and has been reported to take about five **epochs** (i.e. five traversals of the dataset) for typical problems. The resulting classifier (in the form of a radial basis function network) performs very well and often better than those trained with classical methods [Berthold and Diamond 1995, Berthold and Diamond 1998].

Although it yields good radial basis function networks, for the initialization of a prototyped-based classifier the dynamic decay adjustment algorithm has the disadvantage that it may create a large number of cluster prototypes. Although their number depends to some degree on the values of the parameters θ_- and θ_+, a small set of prototypes cannot always be achieved only by adapting these parameters. Therefore it is advisable to

select a subset of the cluster prototypes based, for example, on the number of data points in their core region. Such ideas to reduce the complexity of the model have also been introduced in different contexts, namely learning fuzzy graphs [Berthold and Huber 1999] and learning hierarchical fuzzy rule systems [Gabriel and Berthold 2003a, Gabriel and Berthold 2003b].

An alternative to the DDA algorithm is to choose the cluster centers with **forward selection** or **backward elimination** based on the classification performance they yield. (If the target variable is numeric—and not, as considered here, a nominal value stating a class—these methods are also known as **forward** or **backward regression**.) The idea of these methods is the same as for the (equally named) methods of (greedy) **feature selection**.

In **forward feature selection**, one first builds a set of classifiers, each of which uses only one of the available feature, and determines their classification performance. The feature that yields the best classifier is selected. Next a set of classifiers is build, in which the feature selected in the first step is combined with all remaining features, one per classifier. Again the feature yielding the best performance is selected. In the third step classifiers using three features, two of which are the already selected ones, are evaluated and so on. The feature selection stops when a pre-specified number of features has been reached or the evaluation of the classification performance indicates that no feature yields an improvement that warrants the higher complexity involved in using another feature (see below).

The idea of **backward feature elimination** is equally simply, only the process is reversed: features to be eliminated are selected greedily based on the performance of classifiers from which they have been removed. Feature elimination stops if the elimination of any of the remaining features leads to a significant drop in the classification performance.

Forward selection or backward elimination of cluster centers works in an analogous way. However, before the selection procedure starts, one chooses a kernel function, as discussed for a kernel estimation of a probability density function (cf. page 81). With this kernel function one fixes the characteristics of the cluster, that is, its size and how the similarity "decays" when one moves away from the cluster center. All that is left to choose then are the cluster centers, which are selected from the data points.

The centers are selected in the same greedy fashion as the features. First a set of classifiers with only one cluster, which is located at a data point, is build and evaluated. The data point that yields the best performance is permanently chosen as a cluster center. Then a second cluster, located at another data point, is added and so on, each time choosing the data point that yields the highest improvement of the classification performance.

Similarly, in backward elimination the process is reversed: one starts with a classifier using all data points as centers and eliminates them greedily.

Since clusters alone do not constitute a full classifier (cf. Section 2.5), we also have to consider how to set up the classification scheme, once the clusters are chosen. If the desired classifier is desired to be a *nearest prototype* or *highest membership classifier*, we also need an assignment of class labels to the clusters. The best approach to find such an assignment is to simply sum for each cluster and each class the degrees of membership of the data points and then to assign the class with the largest sum.

If the desired classifier is a *linear function classifier*, we have to determine the weights of the classification functions, that is, the weight matrix $\mathbf{W}$ (cf. Section 2.5). The standard method for this, which is basically a multivariate linear regression, is described in the next section. Note, however, that with this method forward selection and in particular backward elimination, which are not that cheap anyway, become fairly expensive procedures. This holds, even though part of the result obtained for a simpler classifier can be used for the construction of the weight matrix of a more complicated one (with one more cluster center)—see the next section.

Finally, we have to consider when to terminate the selection of cluster centers. In the simplest case one stops when a pre-specified number of cluster centers has been chosen. Alternatively, one may base the decision on the classification performance the classifiers yields, in particular, if it is computed as the sum of squared errors (cf. Section 3.2) or as the likelihood of a correct class prediction. However, a problem with such an approach is that the classification performance on the given data set (from which the cluster centers are selected) can only improve with more clusters. In particular, the classification always gets perfect (in terms of a correct class prediction and provided that there are not contradictory example cases) if all data points are used as cluster centers. This is obvious for a nearest prototype classifier, but also holds for a linear function classifier, for which weights can be found so that the class associated with a data point has membership degree 1 and all other classes have membership degree 0. Only for a probabilistic classifier there is some residual probability for the other classes, although the associated class is assigned the highest probability.

As a consequence it is advisable to use a penalized measure to assess the classification performance, that is, a measure that takes the complexity of the classifier into account. A family of such penalized measures, which is well-known in the statistical literature on model choice, are so-called *information criteria*. An **information criterion** is defined generally as the log-likelihood of the data given the model to evaluate plus a term that

depends on the number of parameters of the model. Thus this criterion takes into account both the statistical goodness of fit and the number of parameters that have to be estimated to achieve this particular degree of fit, by imposing a penalty for increasing the number of parameters [Everitt 1998]. For a prototype-based classifier it can be defined as

$$\mathrm{IC}_\kappa(\mathbf{C}, \mathbf{W} \mid \mathbf{X}, \vec{z}) \;=\; -2 \ln P(\mathbf{X}, \vec{z} \mid \mathbf{C}, \mathbf{W}) + \kappa(\#(\mathbf{C}) + \#(\mathbf{W})),$$

where $\mathbf{X}$ is the database of sample cases, $\vec{z}$ the vector of class labels assigned to these sample cases, $\mathbf{C}$ the set of cluster prototypes and $\mathbf{W}$ the weight matrix of the classification functions (which is not used for a nearest prototype classifier). The term $\#(\mathbf{C})$ denotes the number of parameters of the cluster prototypes and $\#(\mathbf{W})$) the number of elements of the weight matrix, that is, the number c of clusters (plus 1) times the number s of classes. $P(\mathbf{X}, \vec{z} \mid \mathbf{C}, \mathbf{W})$ is the probability of the data set (including the class labels of the data points) given the model (the classifier). Hence it is clear that for $\kappa = 0$ we get a measure that is equivalent to a maximum likelihood approach to model selection. However, pure maximum likelihood is usually a bad choice, as it does not take the number of parameters into account.

Important special cases of the above general form are the so-called **Akaike Information Criterion** (AIC) [Akaike 1974] and the **Bayesian Information Criterion** (BIC) [Schwarz 1978]. The former results for $\kappa = 2$ and is derived from asymptotic decision theoretic considerations. The latter has $\kappa = \ln n$, where n is the number of sample cases, and is derived from an asymptotic Bayesian argument [Heckerman 1998].

An alternative to an information criterion is the more general approach known as the **minimum description length principle** (MDL) [Rissanen 1983, Rissanen 1987]. Intuitively, the basic idea is the following: A sender wants to transmit some data to a receiver. Since transmission is costly, it is tried to encode the message in as few bits as possible. It is assumed that the receiver knows about the symbols that may appear in the message, but does not know anything about their probabilities. Therefore the sender cannot use directly, for instance, a Huffman code [Huffman 1952] for the transmission, because without the probability information the receiver will not be able to decode it. Hence the sender must either use a simpler (and longer) code, for which this information is not required, or he/she must transmit first the coding scheme or the probabilities it is based on. If the message to be sent is long enough, transmitting the coding scheme can pay, since the total number of bits that have to be transmitted may be lower as with a standard coding that does not take into account the probability information.

For choosing a classifier the situation is imagined as follows: We assume that both the sender and the receiver know the data set and the number of classes[2], but that only the sender knows what class labels are assigned to the data points. The goal is to transmit the class labels of the data set to the receiver. To do so, one may use a code with the same number of bits per class label, and thus no model. Or one may transmit first the class probabilities and then encode the class labels with a Huffman code. Or—and this is the case I am interested in—one may construct a classifier from the data, transmit the classifier, and then either encode the class labels with the probabilities computed by the classifier or only transmit the needed corrections to the output of the classifier. The classifier leading to the smallest total description length (sum of classifier description length and data description length) is the one to choose. In the forward selection approach, for example, one stops adding clusters once the total description length increases due to the addition of another cluster.

A third penalized measure is the **predicted residual sum of squared errors** (PRESS) [Myers 1990, Hong *et al.* 2003, Chen *et al.* 2004]. The idea is to estimate the expected sum of squared errors with **leave-1-out cross validation** (cf. also Section 7.1.2). That is, once a set of cluster centers is fixed, n linear function classifiers are constructed from them, which differ only in their weight matrix. For each classifier one data point is set apart and the weight matrix is computed from the remaining $n - 1$ data points (details of this computation are discussed in the next section). The resulting classifier is evaluated on the data point that has been set apart. Finally these (one data point) errors are summed over all classifiers, yielding an error estimate for the whole data set. The weight matrix that is to be used afterwards, however, is computed from the full set of data points.

The advantage of the predicted residual sum of squared errors is that it is also applicable for non-probabilistic classifiers. Its disadvantage is that it is fairly expensive to compute due to the costly cross validation computations. Even though there are some mathematical insights that enable us to improve on truly recomputing the classifier for each data point that is set apart (see the next section for details), the necessary computations still involve the inversion of an $m \times m$ matrix (where m is the number of dimensions of the data space), which has complexity $O(m^3)$. Hence this method can be recommended only for low-dimensional spaces or small data sets.

[2]Note that a strict application of the minimum description length principle would require that the data set as well as the number of classes are unknown to the receiver. However, since they have to be transmitted in any case, they do not have an influence on the classifier ranking and thus are usually neglected or assumed to be known.

4.4 Weight Initialization

After we fixed the cluster prototypes, the weight matrix $\mathbf{W}$ of a linear function classifier has to be initialized. The idea of this initialization is very simple: when the cluster parameters (centers, radii, radial function) are chosen, we can compute the (unnormalized) degree of membership for each data point to each cluster. From these membership degrees the linear classification functions are to compute a class indicator, usually in the form of a binary encoding. Hence the task can be seen as a multivariate linear regression problem (or as s such problems, one for each class, if there are more than two classes, so that one function for each class is needed).

Formally the initialization problem can be described as follows: we construct the $n \times (c+1)$ (extended) unnormalized cluster membership matrix

$$\mathbf{U} = \begin{pmatrix} 1 & u_1^\circ(\vec{x}_1) & \cdots & u_c^\circ(\vec{x}_1) \\ \vdots & & \ddots & \\ 1 & u_1^\circ(\vec{x}_n) & \cdots & u_c^\circ(\vec{x}_n) \end{pmatrix}.$$

Note the additional first column, which takes care of the constants w_{k0} (known as *bias values* or *intercept terms*) in the linear regression functions

$$\forall k; 1 \leq k \leq s: \quad g_k(\vec{x}) = w_{k0} + \sum_{i=1}^{c} w_{ki}\, u_i^\circ(\vec{x}).$$

In addition, we consider a binary encoding of the classes $\vec{z} = (z_1, \ldots, z_n)$, $z_j \in \{1, \ldots, s\}$, $1 \leq j \leq n$, that are assigned to the data points. That is, we construct s binary vectors $\vec{z}_k$, $1 \leq k \leq s$, with

$$\forall j; 1 \leq j \leq n: \quad z_{kj} = \delta_{z_j,k} = \begin{cases} 1, & \text{if } z_j = k, \\ 0, & \text{otherwise,} \end{cases}$$

where $\delta_{z,k}$ is the Kronecker symbol (cf. page 60).[3] Then the initialization problem consists in the s linear multivariate regression problems

$$\forall k; 1 \leq k \leq s: \quad \mathbf{U}\vec{w}_k = \vec{z}_k,$$

where $\vec{w}_k = (w_{k0}, w_{k1}, \ldots, w_{kc})$ contains the weights of the linear classification function g_k. These weights are to be determined in such a way that the sum of squared errors

$$(\mathbf{U}\vec{w}_k - \vec{z}_k)^\top (\mathbf{U}\vec{w}_k - \vec{z}_k)$$

[3] Note that for $s = 2$ one such binary vector is sufficient—one may even use the class index vector $\vec{z}$ directly. Multiple vectors are only needed for more than two classes.

is minimized. How this can be achieved is reviewed in detail in Section A.6 in the appendix. The core result is the linear equation system (known as the system of *normal equations*)

$$\mathbf{U}^\top\mathbf{U}\,\vec{w}_k = \mathbf{U}^\top\vec{z}_k,$$

from which the weights can be computed, for example, as

$$\vec{w}_k = \left(\mathbf{U}^\top\mathbf{U}\right)^{-1}\mathbf{U}^\top\vec{z}_k$$

(cf. Section A.6 in the appendix for details).

For a single initialization the above equation is all we need. However, in the preceding section we considered a forward selection of cluster centers, which gives rise to a large number of regression problems, namely one for each addition of one cluster center candidate. In order to solve these problems efficiently, it is advisable to compute the inverse in the above equation with Cholesky decomposition (cf. Section A.3 in the appendix).[4] The reason is that the addition of a cluster center adds a column to the matrix $\mathbf{U}$, which leads to an additional row and column in the matrix $\mathbf{U}^\top\mathbf{U}$. However, this is the only change. Apart from this additional row and column all other matrix elements stay the same. Similarly, the vector $\mathbf{U}^\top\vec{z}_k$ gets a new element, but the already existing elements are unchanged. These properties can be exploited nicely with Cholesky decomposition, because they make it possible to adapt an existing Cholesky decomposition of the smaller matrix $\mathbf{U}$ (with one column less) instead of fully recomputing it.

Another computational issue concerns the computation of the *sum of squared errors* obtained with such a classifier as well as the predicted residual sum of squared errors. The former can easily be computed by exploiting

$$\begin{aligned} e_{\text{sqr}}(\mathbf{X},\vec{z};\mathbf{C},\mathbf{W}) &= (\mathbf{U}\vec{w}_k - \vec{z}_k)^\top(\mathbf{U}\vec{w}_k - \vec{z}_k) \\ &= \vec{w}_k^\top\mathbf{U}^\top\mathbf{U}\vec{w}_k - 2\vec{w}_k^\top\mathbf{U}^\top\vec{z}_k - \vec{z}_k^\top\vec{z}_k \\ &= \vec{w}_k^\top\mathbf{L}^\top\mathbf{L}\,\vec{w}_k - 2\vec{w}_k^\top\vec{r}_k - \vec{z}_k^\top\vec{z}_k, \end{aligned}$$

where $\mathbf{L}$ is the lower (or left) triangular matrix that results from a Cholesky decomposition of $\mathbf{U}^\top\mathbf{U}$ and $\vec{r}_k = \mathbf{U}^\top\vec{z}_k$ is the right hand side of the system of normal equations. Although this formula is numerically not very stable and should not be used for tasks where precision is the main concern, it is an efficient way of getting a good estimate of the sum of squared errors, which is sufficient for a forward selection of initial cluster centers.

[4] Note that $\mathbf{U}^\top\mathbf{U}$ is symmetric and positive definite, as it is computed in basically the same way as a covariance matrix, namely as the sum of outer products of vectors.

The *predicted residual sum of squared errors*, on the other hand, is more difficult to compute. Setting apart a data point $\vec{x}_j$, $1 \leq j \leq n$, for a cross validation means to delete the j-th row from the matrix $\mathbf{U}$. I denote this row by $\vec{u}_j^\top = (1, u_1^\circ(\vec{x}_j), \ldots, u_c^\circ(\vec{x}_j))$ and the resulting matrix without this row by $\mathbf{U}_j$. In addition, I denote the k-th vector of the binary encoding of the classes, from which the j-th element (which corresponds to the removed data point) has been deleted, by $\vec{z}_{k[j]} = (z_{k1}, \ldots, z_{k(j-1)}, z_{k(j+1)}, \ldots, z_{kn})^\top$. Then we can write the system of normal equations we have to solve if the data point $\vec{x}_j$ is left out as

$$\mathbf{U}_j^\top \mathbf{U}_j \, \vec{w}_{k[j]} = \mathbf{U}_j^\top \vec{z}_{k[j]},$$

where $\vec{w}_{k[j]}$ is the solution for this fold of the cross validation. However, actually solving one such system for each data point just to evaluate it on the left out data point and then to sum the squared errors would be much too costly. Rather one tries to exploit the fact that all these systems of normal equations are very similar to the one using all data points and tries to compute the squared error of the reduced system from the squared error of the full system. This is done as follows [Chen *et al.* 2004]: we start from

$$e_j \;=\; (\vec{u}_j^\top \vec{w}_k - z_{kj})^2 \;=\; \left(\vec{u}_j^\top \left(\mathbf{U}^\top \mathbf{U}\right)^{-1} \mathbf{U}^\top \vec{z}_k - z_{kj}\right)^2,$$

that is, the full classifier error for the data point $\vec{x}_j$, and

$$e_j^* \;=\; (\vec{u}_j^\top \vec{w}_{k[j]} - z_{kj})^2 \;=\; \left(\vec{u}_j^\top \left(\mathbf{U}_j^\top \mathbf{U}_j\right)^{-1} \mathbf{U}_j^\top \vec{z}_{k[j]} - z_{kj}\right)^2,$$

that is, the cross validation classifier error for the data point $\vec{x}_j$. The idea of the following derivation is to find a relation between these two errors, so that we can compute e_j^* from e_j. To achieve this, we note first that due to the definition of the matrix $\mathbf{U}_j$, it is obviously

$$\mathbf{U}_j^\top \mathbf{U}_j = \mathbf{U}^\top \mathbf{U} - \vec{u}_j \vec{u}_j^\top,$$

that is, only the outer product of the data point that is left out is removed from the matrix $\mathbf{U}^\top \mathbf{U}$. Therefore we obtain with the matrix inversion lemma (cf. Section A.7 in the appendix for a detailed derivation)

$$\begin{aligned}
\left(\mathbf{U}_j^\top \mathbf{U}_j\right)^{-1} &= \left(\mathbf{U}^\top \mathbf{U} - \vec{u}_j \vec{u}_j^\top\right)^{-1} \\
&= (\mathbf{U}^\top \mathbf{U})^{-1} + \frac{(\mathbf{U}^\top \mathbf{U})^{-1} \vec{u}_j \vec{u}_j^\top (\mathbf{U}^\top \mathbf{U})^{-1}}{1 - \vec{u}_j^\top (\mathbf{U}^\top \mathbf{U})^{-1} \vec{u}_j}.
\end{aligned}$$

As a consequence we have that

$$\begin{aligned}
\vec{u}_j^{\top}(\mathbf{U}_j^{\top}\mathbf{U}_j)^{-1} &= \vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1} + \frac{\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}\vec{u}_j\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}}{1-\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}\vec{u}_j} \\
&= \left(1 + \frac{\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}\vec{u}_j}{1-\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}\vec{u}_j}\right)\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1} \\
&= \frac{\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}}{1-\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}\vec{u}_j}.
\end{aligned}$$

Next we observe that it is obviously

$$\mathbf{U}_j^{\top}\vec{z}_{k[j]} = \mathbf{U}^{\top}\vec{z}_k - \vec{u}_j^{\top}z_{kj}.$$

Substituting this and the preceding equation into the expression for e_j^* yields

$$\begin{aligned}
e_j^* &= \left(\frac{\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}}{1-\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}\vec{u}_j}\left(\mathbf{U}^{\top}\vec{z}_k - \vec{u}_j^{\top}z_{kj}\right) - z_{kj}\right)^2 \\
&= \left(\frac{\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}\mathbf{U}^{\top}\vec{z}_k}{1-\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}\vec{u}_j} - \frac{\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}\vec{u}_j^{\top}}{1-\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}\vec{u}_j}z_{kj} - z_{kj}\right)^2 \\
&= \left(\frac{\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}\mathbf{U}^{\top}\vec{z}_k - z_{kj}}{1-\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}\vec{u}_j}\right)^2 \\
&= \frac{e_j}{\left(1-\vec{u}_j^{\top}(\mathbf{U}^{\top}\mathbf{U})^{-1}\vec{u}_j\right)^2}.
\end{aligned}$$

However, even though this computation speeds up the cross validation considerably, the predicted residual sum of squared errors remains a computationally very expensive method. If the model construction consists only of a forwards selection of clusters based on this measure, it may be acceptable, but it cannot really be recommended for initialization purposes.

Chapter 5

Update Methods

After the cluster prototypes have been initialized, they may be improved (w.r.t. the objective functions discussed in Chapter 3) by iteratively updating the cluster parameters. For such iterative updating there are basically four categories of approaches, which are discussed in this chapter:

Gradient methods try to find a (local) optimum by iteratively computing the gradient of the objective function at the current point in the parameter space and making a (small) step in (or against) the direction of this gradient, thus reaching a new point in the parameter space. The size of the step can be controlled by a parameter that is often called "learning rate".

In *alternating optimization* the parameters of the objective function are split into (usually, but not necessarily) two parts, so that the optimum for each part can be computed directly, provided all other parameters are fixed. The sets of parameters are then worked on in an alternating fashion, optimizing one set of parameters while the other is fixed.

In *competitive learning* the cluster prototypes compete over the data points. Depending on the number of competitions they win or the degree to which they can acquire the data points (similar to a membership degree), the cluster parameters are adapted. An advantage of this approach is that it can also be executed "online", updating after each data point.

Guided random search exploits ideas from statistical optimization and evolution theory. The cluster parameters are randomly modified or randomly combined from two candidate solutions. Then the objective function is recomputed with the new set of parameters. Based on the resulting value and a random component the new parameters replace the old (simulated annealing) or have a higher probability of survival (genetic algorithms).

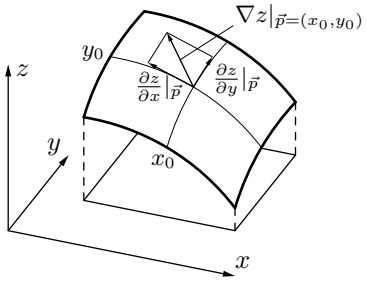

Figure 5.1: Illustration of the gradient vector of a real-valued function $z = f(x, y)$ at a point $\vec{p} = (x_0, y_0)$. It is $\nabla z|_{\vec{p}} = \left(\frac{\partial z}{\partial x}|_{\vec{p}}, \frac{\partial z}{\partial y}|_{\vec{p}}\right)$.

5.1 Gradient Methods

Gradient ascent or descent is among the oldest and best-known methods for function optimization. In the context of prototype-based clustering and classification it is most often found in connection with a linear function classifier and the sum of squared errors as the objective function. That is, gradient descent is used for training a *radial basis function neural network (RBF network)* [Rojas 1993, Haykin 1994, Zell 1994, Anderson 1995, Bishop 1995, Nauck *et al.* 2003]. Furthermore, it has also been applied in other areas as an alternative to the *expectation maximization algorithm* (see Section 5.2) [Russel *et al.* 1995, Bauer *et al.* 1997]. However, since the approach is fully general and requires only that the objective function is differentiable, it can, in principle, be applied to almost any objective function.

5.1.1 General Approach

The rationale underlying gradient methods is that we can attempt to optimize an objective function if we can find the direction in which we have to change a given set of parameters in order to improve the value of the objective function. This information is contained in the **gradient** of the function. The gradient is a differential operator (denoted by the symbol ∇, which is pronounced "nabla") that assigns a vector to each point of the parameter space. The elements of this vector are the partial derivatives of the function w.r.t. its parameters. Formally, the gradient turns a real-valued function into a **vector field**. Intuitively, the gradient of a function (at a given point) points into that direction in the parameter space, in which the function ascends most steeply. Its length specifies how steep the function is at this point. As an illustration Figure 5.1 shows the gradient vector of a real-valued function $f(x, y)$ at a point $\vec{p} = (x_0, y_0)$.

The general procedure of gradient methods is as follows: given initial values for the parameters (in the context of prototype-based classification and clustering: obtained with one of the initialization methods studied in the preceding chapter), we compute the gradient of the objective function and modify the parameters according to this gradient: the change is in the direction of the gradient if we want to maximize the function, and in the opposite direction if we want to minimize it. The length of the step depends on the length of the gradient vector at the current point and a factor by which it is multiplied in order to control the process. Due to its use in artificial neural networks (here in particular: radial basis function networks), this factor is known as the **learning rate**. It controls the "speed of learning", i.e., how quickly the parameters change their values.

The adapted parameters specify a new point in the parameter space, at which the gradient is recomputed (because it may be in a different direction compared to the initial point). Another step in or against the direction is carried out, and again the gradient is recomputed. The procedure is iterated until the gradient is sufficiently small, so that we can be reasonably sure that a (local) optimum of the objective function has been reached. In practice, a user specifies a threshold ε for the change of any parameter. If all changes are below this threshold, the procedure is terminated.

Formally, a gradient method can be described as follows: let $f(\vec{\theta})$ be an objective function that has a parameter vector $\vec{\theta}$ as its argument. In order to find a (local) optimum of the function f w.r.t. $\vec{\theta}$, we determine an initial parameter vector $\vec{\theta}^{(0)}$. Then we compute the sequence of vectors

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} \pm \eta \cdot \nabla_{\vec{\theta}} f\big(\vec{\theta}^{(t)}\big), \qquad t = 0, 1, 2, \ldots,$$

with the plus sign corresponding to gradient ascent (to maximize f) and the minus sign corresponding to gradient descent (to minimize f). It is

$$\nabla_{\vec{\theta}} = \left(\frac{\partial}{\partial\theta_1}, \ldots, \frac{\partial}{\partial\theta_k}\right)^\top$$

for $\vec{\theta} = \big(\theta_1, \ldots, \theta_k\big)^\top$ and η is the learning rate. As a consequence, all we have to do now is to compute the specific gradient of a given objective function, which is exactly what is done in the following sections.

Note that we may also take the update formula apart into one formula for each parameter, or that we may refer to subvectors of $\vec{\theta}$. This is important, because in the following it is convenient to split the set of parameters. For example, we may want to consider the update of the cluster centers separate from the update of the associated covariance matrices.

Note also that several of the objective functions that were considered in Chapter 3 consist of a sum of certain expressions over the data points. As a consequence the gradients of these functions are also computed as sums of expressions over the individual data points. That is, they consist of one term for each data point. Thus the idea suggests itself to take the update apart into individual steps, one for each data point. That is, the gradient ascent or descent is carried out as follows: one computes the term of the gradient that corresponds to one data point (that is, one term of the sum) and immediately adds the resulting change to the current parameter vector (instead of first aggregating the changes over all data points).

This approach to gradient descent is well known as **stochastic gradient descent** [Robbins and Monro 1951, Duda and Hart 1973, Spall 2003]. Here the term "stochastic" refers to the recommendable practice to shuffle the data points randomly between two traversal of the data set. In this way one tries to rule out or at least to mitigate effects that result from the order in which the data points are processed. In the area of neural networks stochastic gradient descent is known as **online training**, because the parameters are updated online with the data points that get available. In contrast to this **batch training** is equivalent to standard gradient descent, in which the changes are applied to the parameters only after all data points have been processed and the changes have been aggregated [Rojas 1993, Haykin 1994, Anderson 1995, Nauck *et al.* 2003]. Online training has been found to be often much faster than batch training, especially on large data sets. Its disadvantage is a reduced robustness of the training procedure.

The above considerations are all we need to know if we want to apply gradient ascent or descent to an objective function without constraints. However, all of the objective functions considered in Chapter 3 constrain at least some of the parameters appearing in them. For example, in the sum of (squared) distances the membership degrees u_{ij}, $1 \leq i \leq c$, $1 \leq j \leq n$, must lie in the unit interval (cf. page 51 in Section 3.1). In addition, they must sum to 1 for each value of j (i.e. for each data point). Analogous constraints hold for the probabilities appearing in a likelihood function.

Unfortunately, the gradient of an objective function does not respect these constraints (since they are not part of the objective function). Therefore a gradient ascent or descent step can leave the region of allowed parameter values. The standard solution to this problem is to **"repair" the parameters** if they lie outside the allowed region. The general idea underlying such a repair is to change the parameters to the closest acceptable point in the parameter space. However, sometimes it is difficult to determine this closest point and then heuristics are applied.

5.1.2 Gradient Descent on Sum of Squared Distances

As pointed out in the preceding section, the objective of this and the following sections is to derive the gradients for the different objective functions studied in Chapter 3. I go through the different objective functions in the same order as they were considered in Chapter 3, starting with the sum of (squared) distances (cf. Section 3.1). Since this objective function is to be minimized, we have to carry out a gradient descent.

Formally, the task consists in the following: The objective function for the sum of (squared) distances is defined as (cf. page 51 in Section 3.1)

$$J(\mathbf{X}; \mathbf{U}, \mathbf{C}) = \sum_{j=1}^{n} \sum_{i=1}^{c} u_{ij}^{w} \, d_{ij}^{2}.$$

Therefore the update formulae for the different parameters (recall from the preceding section that they can be treated separately) are

$$\begin{aligned}
\mathbf{U}^{(t+1)} &= \mathbf{U}^{(t)} - \eta_{\mathbf{U}} \; \nabla_{\mathbf{U}} \; J(\mathbf{X}; \mathbf{U}^{(t)}, \mathbf{C}^{(t)}), \\
\vec{\mu}_i^{\,(t+1)} &= \vec{\mu}_i^{\,(t)} - \frac{\eta_{\vec{\mu}}}{2} \; \nabla_{\vec{\mu}_i} \; J(\mathbf{X}; \mathbf{U}^{(t)}, \mathbf{C}^{(t)}), \\
\left(\boldsymbol{\Sigma}_i^{(t+1)}\right)^{-1} &= \left(\boldsymbol{\Sigma}_i^{(t)}\right)^{-1} - \eta_{\boldsymbol{\Sigma}} \; \nabla_{\boldsymbol{\Sigma}_i^{-1}} J(\mathbf{X}; \mathbf{U}^{(t)}, \mathbf{C}^{(t)}).
\end{aligned}$$

(The additional factor $\frac{1}{2}$ in the second formula and the use of the inverse $\boldsymbol{\Sigma}_i^{-1}$ instead of $\boldsymbol{\Sigma}_i$ in the third formula are explained below.)

What we have to do is to compute the different gradients appearing in these formulae. Before we do so, however, note that the learning rates η in these formulae carry indices, which indicate to which parameter set they belong. The reason is that it is usually advisable to use different learning rates for these parameters, because the objective function is not equally sensitive to changes, for example, of the cluster centers compared to changes of the covariance matrices. In particular, the learning rate for the cluster weights ϱ_i and the covariance matrices $\boldsymbol{\Sigma}_i$ should be considerably smaller (about a factor of 5 to 10) than the learning rates for the cluster centers.

For deriving the gradient of the objective function J w.r.t. the fuzzy partition matrix $\mathbf{U}$ it is most convenient to consider the different membership degrees u_{ij}, $1 \leq i \leq c$, $1 \leq j \leq n$, individually. That is, we exploit

$$\nabla_{\mathbf{U}} = \begin{pmatrix} \frac{\partial}{\partial u_{11}} & \cdots & \frac{\partial}{\partial u_{1c}} \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial u_{n1}} & \cdots & \frac{\partial}{\partial u_{nc}} \end{pmatrix}.$$

For an individual membership degree u_{kl}, $1 \leq k \leq c$, $1 \leq l \leq n$, we get

$$\frac{\partial}{\partial u_{kl}} J(\mathbf{X}; \mathbf{U}, \mathbf{C}) \;=\; \frac{\partial}{\partial u_{kl}} \sum_{j=1}^{n} \sum_{i=1}^{c} u_{ij}^{w}\, d_{ij}^2 \;=\; w\, u_{kl}^{w-1}\, d_{kl}^2$$

and thus $\forall i; 1 \leq i \leq c : \forall j : 1 \leq j \leq n$:

$$u_{ij}^{(t+1)} = u_{ij}^{(t)} - \eta_{\mathbf{U}}\, w \left(u_{ij}^{(t)}\right)^{w-1} \left(d_{ij}^{(t)}\right)^2.$$

However, this update formula is not enough. As already pointed out in the preceding section the objective function J comes with constraints on the membership degrees, namely $\forall i; 1 \leq i \leq c : \forall j : 1 \leq j \leq n$:

$$u_{ij} \in [0, 1], \qquad \sum_{i=1}^{c} u_{ij} = 1, \qquad \text{and} \qquad \sum_{j=1}^{n} u_{ij} > 0.$$

To satisfy these constraints the new membership degrees may have to be adapted. The simplest (heuristic) scheme for this is:

- If $u_{ij}^{(t+1)} > 1$, set $u_{ij}^{(t+1)} = 1$ and
  if $u_{ij}^{(t+1)} < 0$, set $u_{ij}^{(t+1)} = 0$ (to satisfy the first constraint).
- Finally normalize the new membership degrees for each value of j (i.e. for each data point) to sum 1 (to satisfy the second constraint).

Although this scheme is plausible, it may be advisable in practice not to force the membership degrees into the unit interval, but actually into an even smaller interval $[\varepsilon, 1 - \varepsilon]$, with a small positive number ε. The reason is that as soon as a membership value u_{ij} gets zero, it is not updated anymore. This can easily be seen if one writes the update formula as

$$u_{ij}^{(t+1)} = u_{ij}^{(t)} \cdot \left(1 - \eta_{\mathbf{U}}\, w \left(u_{ij}^{(t)}\right)^{w-2} \left(d_{ij}^{(t)}\right)^2\right).$$

Finally, the third constraint can be handled in different ways. In the first place, one may check (after the adaptations in the first point of the scheme) whether there is an empty cluster, that is, a cluster to which all data points have a vanishing membership degree. If there is, all data points are assigned with the same, very small membership degree $\varepsilon > 0$ to this cluster. As an alternative, one may correct all membership degrees that are less than or equal to 0 to some small value ε, regardless of whether there is an empty

cluster or not. (Of course, these adaptations have to be carried out before the renormalization.) The value of this ε should be very small (for example, $\varepsilon = 10^{-6}$), to prevent that the clustering result gets distorted.

For the update of the cluster centers $\vec{\mu}_k$, $1 \leq k \leq c$, we have

$$\nabla_{\vec{\mu}_k} J(\mathbf{X}; \mathbf{U}, \mathbf{C}) \;=\; \nabla_{\vec{\mu}_k} \sum_{j=1}^{n} \sum_{i=1}^{c} u_{ij}^{w}\, d_{ij}^2 \;=\; \sum_{j=1}^{n} u_{kj}^{w}\, \nabla_{\vec{\mu}_k} d_{kj}^2.$$

To proceed, we need the derivative of the squared distance. This derivative can be found in Section A.1 in the appendix, in which some basic vector and matrix derivatives are collected. It is (cf. page 268)

$$\nabla_{\vec{\mu}_k} d_{kj}^2 \;=\; \nabla_{\vec{\mu}_k} (\vec{x} - \vec{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\vec{x} - \vec{\mu}_k) \;=\; -2\boldsymbol{\Sigma}_k^{-1}(\vec{x}_j - \vec{\mu}_k).$$

Therefore we have $\forall i; 1 \leq i \leq c$:

$$\begin{aligned} \vec{\mu}_i^{(t+1)} &= \vec{\mu}_i^{(t)} - \frac{\eta_{\vec{\mu}}}{2} \nabla_{\vec{\mu}_k} J(\mathbf{X}; \mathbf{U}^{(t)}, \mathbf{C}^{(t)}) \\ &= \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}} \left(\boldsymbol{\Sigma}_i^{(t)}\right)^{-1} \sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^w \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right). \end{aligned}$$

This explains the additional factor $\frac{1}{2}$ in the original statement of the update formulae: this factor cancels against the 2 that results from the derivative of the distance. Alternatively one can argue at this point that the factor of 2 appearing here may be incorporated into the learning rate and thus may be canceled. It is simply convenient to eliminate such constants.

Note that the above formula simplifies to

$$\begin{aligned} \vec{\mu}_i^{(t+1)} &= \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}} \sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^w \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right) \\ &= \left(1 - \eta_{\vec{\mu}} \sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^w\right) \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}} \sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^w \vec{x}_j \end{aligned}$$

if the Euclidean distance is used.[1] This update formula is very intuitive: Suppose we choose a cluster-specific and time-dependent learning rate

$$\eta_{\vec{\mu}_i}^{(t)} = \frac{1}{\sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^w}.$$

[1] For the Euclidean distance it is $\boldsymbol{\Sigma} = \mathbf{1}$, cf. page 14 in Section 2.1.

Then the old center $\vec{\mu}_i^{(t)}$ is canceled and the new cluster center is simply set to the center of gravity of the weighted data points. That is, we get

$$\vec{\mu}_i^{(t+1)} = \frac{\sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^w \vec{x}_j}{\sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^w},$$

which is exactly the update formula of standard fuzzy clustering, which is obtained in an alternating optimization approach (cf. Section 5.2.3).

If we choose a smaller, but still cluster-specific and time-dependent learning rate, we make a step from the old cluster center towards this center of gravity. In this case the formula describes a convex combination of two points in the data space and the result must be on the line connecting the two points. This can easily be seen by writing the learning rate as a product of the learning rate chosen above and a factor $\xi < 1$, i.e. as

$$\eta_{\vec{\mu}_i}^{(t)} = \frac{\xi}{\sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^w}, \qquad \xi \in (0, 1).$$

With this learning rate we have

$$\vec{\mu}_i^{(t+1)} = (1 - \xi) \cdot \vec{\mu}_i^{(t)} + \xi \cdot \frac{\sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^w \vec{x}_j}{\sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^w}.$$

This way of rewriting the update formula is also important for another reason: as we will see in Section 5.3, we obtain almost the same update formula for a specific batch version of fuzzy learning vector quantization, again using a cluster-specific and time-dependent learning rate. However, the derivation of this formula in the context of fuzzy learning vector quantization does not refer to a gradient. It is pleasing to see that completely different approaches can lead to the same intuitive procedure.

Next we turn to the update of the covariance matrices $\mathbf{\Sigma}_i$, $1 \leq i \leq c$. As is also pointed out in Section A.2.2, it is advantageous not to consider the covariance matrix itself, but its inverse. That is, instead of

$$\mathbf{\Sigma}_i^{(t+1)} = \mathbf{\Sigma}_i^{(t)} - \eta_{\mathbf{\Sigma}} \nabla_{\mathbf{\Sigma}_i} J(\mathbf{X}; \mathbf{U}^{(t)}, \mathbf{C}^{(t)}),$$

which is difficult to compute due to the fact that only the inverse $\mathbf{\Sigma}_i^{-1}$ appears in J, we consider (as pointed out at the beginning of this section)

$$\left(\mathbf{\Sigma}_i^{(t+1)}\right)^{-1} = \left(\mathbf{\Sigma}_i^{(t)}\right)^{-1} - \eta_{\mathbf{\Sigma}} \nabla_{\mathbf{\Sigma}_i^{-1}} J(\mathbf{X}; \mathbf{U}^{(t)}, \mathbf{C}^{(t)}).$$

In this way the computations become much simpler. It is actually a standard trick when having to compute the derivative of some function w.r.t. an inverse matrix, which we will meet again several times in the following sections. Note that there is no additional cost involved in obtaining the inverse matrix, because we need it for the computation of the Mahalanobis distance anyway. Actually one could just as well use the inverse covariance matrix as a parameter in the first place, as the only reason for using the covariance matrix is that it is commonly used in statistics.

For the gradient in the above formula we obtain

$$\begin{aligned}
\nabla_{\boldsymbol{\Sigma}_k^{-1}} J(\mathbf{X}; \mathbf{U}^{(t)}, \mathbf{C}^{(t)}) &= \nabla_{\boldsymbol{\Sigma}_k^{-1}} \sum_{j=1}^{n} \sum_{i=1}^{c} u_{ij}^{w}\, d_{ij}^2 = \sum_{j=1}^{n} u_{kj}^{w}\, \nabla_{\boldsymbol{\Sigma}_k^{-1}} d_{kj}^2 \\
&= \sum_{j=1}^{n} u_{kj}^{w}\, \nabla_{\boldsymbol{\Sigma}_k^{-1}}\, (\vec{x}_j - \vec{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\vec{x}_j - \vec{\mu}_k) \\
&= \sum_{j=1}^{n} u_{kj}^{w}\, (\vec{x}_j - \vec{\mu}_k)(\vec{x} - \vec{\mu})^\top.
\end{aligned}$$

Again the justification for the last step can be found in Section A.1 in the appendix, in which some basic vector and matrix derivatives are collected (cf. page 268). As a consequence we arrive at the update rule $\forall i; 1 \leq i \leq c$:

$$\left(\boldsymbol{\Sigma}_i^{(t+1)}\right)^{-1} = \left(\boldsymbol{\Sigma}_i^{(t)}\right)^{-1} - \eta_{\boldsymbol{\Sigma}} \sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^{w} \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right) \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)^\top.$$

With this formula there is some similarity, but not such a direct equivalence to the corresponding update formula for learning vector quantization, as there is for the cluster centers $\vec{\mu}_i$ (see above and cf. Section 5.3). Nevertheless it shows how closely related these two approaches are.

It should be noted that there may be a constraint on the covariance matrix, namely that its determinant must be 1, so that all clusters have the same size. Of course, a gradient descent according to the above formula does not respect this constraint and thus unacceptable covariance matrices may result. The simplest solution to this problem is to normalize the resulting covariance matrix (or, equivalently, its inverse) to determinant 1.

This completes my discussion of gradient descent on the sum of (squared) distances. I do not consider explicitely the various special cases and extensions of this objective function as they were discussed in Section 3.1. The derivations of the update formulae for them follow basically the same lines and are easy to obtain in analogy to the derivations carried out above.

5.1.3 Gradient Descent on Sum of Squared Errors

In the area of classifier construction, gradient descent may be best known as a training method for artificial neural networks. As already pointed out several times above, I consider here the specific neural network type that is known as **radial basis function network** (RBF network) [Rojas 1993, Haykin 1994, Anderson 1995, Nauck *et al.* 2003]. Such a network is equivalent to a cluster model together with a set of (linear) classification functions as it was studied in Section 2.5: the hidden neurons represent the clusters and the output neurons represent the classification functions. Gradient descent is used to optimize the weights of the classification functions as well as the locations of the clusters and their shape and size parameters.

Formally, the task consists in the following: for each data point $\vec{x}_j$, $1 \leq j \leq n$, we are given a class $z_j \in \{1, \ldots, s\}$. For easier reference these classes are combined into a vector $\vec{z} = (z_1, \ldots, z_n)^\top$. The quality of the classification is measured by how well the classifier produces these desired outputs, usually based on a binary encoding of the classes (cf. Section 3.2). For a gradient descent approach it is best to measure this quality by the sum of squared errors, because obtaining the gradient of this objective function is technically easiest. That is, we use the error function

$$e_{\text{sqr}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) = \sum_{j=1}^{n} \sum_{k=1}^{s} (\delta_{k,z_j} - g_k(\vec{x}_j))^2,$$

where the g_k, $1 \leq k \leq s$, are the (linear) classification functions, the weights of which are contained in the matrix $\mathbf{W}$. $\delta_{k,z}$ is the Kronecker symbol, which describes the binary encoding of the classes (cf. Section 3.2). As a consequence the update formulae for the different parameters are

$$\begin{aligned}
\mathbf{W}^{(t+1)} &= \mathbf{W}^{(t)} - \frac{\eta_{\mathbf{W}}}{2} \nabla_{\mathbf{W}} e_{\text{sqr}}(\mathbf{X}, \vec{z}; \mathbf{C}^{(t)}, \mathbf{W}^{(t)}), \\
\vec{\mu}_i^{\,(t+1)} &= \vec{\mu}_i^{\,(t)} - \frac{\eta_{\vec{\mu}}}{2} \nabla_{\vec{\mu}_i} e_{\text{sqr}}(\mathbf{X}, \vec{z}; \mathbf{C}^{(t)}, \mathbf{W}^{(t)}), \\
\left(\boldsymbol{\Sigma}_i^{(t+1)}\right)^{-1} &= \left(\boldsymbol{\Sigma}_i^{(t)}\right)^{-1} - \frac{\eta_{\boldsymbol{\Sigma}}}{2} \nabla_{\boldsymbol{\Sigma}_i^{-1}} e_{\text{sqr}}(\mathbf{X}, \vec{z}; \mathbf{C}^{(t)}, \mathbf{W}^{(t)}).
\end{aligned}$$

(Recall from Section 5.1.1 that the different parameters can be treated separately—due to the partial derivatives—and thus we are allowed to split the update into separate formulae for each set of parameters.) Why the learning rates are written as $\frac{\eta}{2}$ will become clear below. The reason is basically the same as for the update formula for the centers in the preceding

section: the factor $\frac{1}{2}$ cancels against a factor of 2 that results from taking the derivative of the square in the objective function.

As in the preceding section for the fuzzy partition matrix $\mathbf{U}$, it is most convenient to derive the derivative of e_{sqr} w.r.t. the weight matrix $\mathbf{W}$ by considering the weights w_{ki}, $1 \leq k \leq s$, $0 \leq i \leq c$, individually. For such an individual weight we get (for $1 \leq q \leq s$ and $0 \leq r \leq c$)

$$\begin{aligned}
\frac{\partial}{\partial w_{qr}} e_{\text{sqr}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) &= \frac{\partial}{\partial w_{qr}} \sum_{j=1}^{n} \sum_{k=1}^{s} (\delta_{k,z_j} - g_k(\vec{x}_j))^2 \\
&= \frac{\partial}{\partial w_{qr}} \sum_{j=1}^{n} \sum_{k=1}^{s} (\delta_{k,z_j} - \vec{w}_k^{\top} \vec{u}(\vec{x}_j))^2 \\
&= \sum_{j=1}^{n} \frac{\partial}{\partial w_{qr}} (\delta_{q,z_j} - \vec{w}_q^{\top} \vec{u}(\vec{x}_j))^2 \\
&= -2 \sum_{j=1}^{n} (\delta_{q,z_j} - \vec{w}_q^{\top} \vec{u}(\vec{x}_j)) \cdot u_r(\vec{x}_j).
\end{aligned}$$

Here $\vec{w}_q = (w_{q0}, w_{q1}, \ldots, w_{qc})^{\top}$ is a weight vector containing the elements of the q-th row of the matrix $\mathbf{W}$. The vector $\vec{u}(\vec{x}) = (1, u_1(\vec{x}), \ldots, u_c(\vec{x}))^{\top}$ holds the membership degrees of the data point $\vec{x}$ to the c different clusters (cf. page 38 in Section 2.5).[2] Note that the last step follows from

$$\frac{\partial}{\partial w_{qr}} \vec{w}_q^{\top} \vec{u}(\vec{x}_j) = \frac{\partial}{\partial w_{qr}} \sum_{i=0}^{c} w_{qi}\, u_i(\vec{x}) = u_r(\vec{x}).$$

As a consequence we have $\forall k; 1 \leq k \leq s : \forall i; 1 \leq i \leq c :$

$$w_{ki}^{(t+1)} = w_{ki}^{(t)} + \eta_{\mathbf{W}} \sum_{j=1}^{n} \left(\delta_{k,z_j} - \left(\vec{w}_k^{(t)} \right)^{\top} \vec{u}^{(t)}(\vec{x}_j) \right) \cdot u_i^{(t)}(\vec{x}_j).$$

Note how the factor -2 cancels against the factor $\frac{1}{2}$ and the minus sign signifying gradient *descent* in the general statement of the update formula. Note also that the scalar product in the first factor of each term as well as the second factor are computed in the process of executing the classifier and are thus readily available, without any additional costs.

[2] Note that the additional 1 in the first element of the vector $\vec{u}(\vec{x})$ is combined with the bias weight of the classification function. In addition, recall that for a linear function classifier it is usually $\vec{u}(\vec{x}) = \vec{u}^{\circ}(\vec{x})$. That is, one uses the unnormalized and untransformed membership degrees (again cf. Section 2.5).

To obtain the update formula for the cluster centers, we have to compute

$$\begin{aligned}\nabla_{\vec{\mu}_r} e_{\mathrm{sqr}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) &= \nabla_{\vec{\mu}_r} \sum_{j=1}^{n} (\delta_{k,z_j} - g_k(\vec{x}_j))^2 \\ &= \sum_{j=1}^{n} \sum_{k=1}^{s} \nabla_{\vec{\mu}_r} (\delta_{k,z_j} - \vec{w}_k^\top \vec{u}(\vec{x}_j))^2.\end{aligned}$$

Similarly, for the update of the covariance matrices we have to compute

$$\begin{aligned}\nabla_{\mathbf{\Sigma}_r} e_{\mathrm{sqr}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) &= \nabla_{\mathbf{\Sigma}_r} \sum_{j=1}^{n} (\delta_{k,z_j} - g_k(\vec{x}_j))^2 \\ &= \sum_{j=1}^{n} \sum_{k=1}^{s} \nabla_{\mathbf{\Sigma}_r} (\delta_{k,z_j} - \vec{w}_k^\top \vec{u}(\vec{x}_j))^2.\end{aligned}$$

Since the centers as well as the covariance matrices enter the terms of these sums only through the classification functions g_k, $1 \leq k \leq s$, and in particular only through the membership degrees $u_i(\vec{x})$ in the vector $\vec{u}(\vec{x})$, it is most convenient to compute these gradients by applying the chain rule. That is, we compute them by exploiting for the cluster centers

$$\begin{aligned}\nabla_{\vec{\mu}_r} (\delta_{k,z_j} - g_k(\vec{x}_j))^2 &= \frac{\partial}{\partial u_r(\vec{x}_j)} (\delta_{k,z_j} - \vec{w}_k^\top \vec{u}(\vec{x}_j))^2 \cdot \nabla_{\vec{\mu}_r} u_r(\vec{x}_j) \\ &= -2 \left(\delta_{k,z_j} - \vec{w}_k^\top \vec{u}(\vec{x}_j)\right) \cdot w_{kr} \cdot \nabla_{\vec{\mu}_r} u_r(\vec{x}_j)\end{aligned}$$

and for the covariance matrices

$$\begin{aligned}\nabla_{\mathbf{\Sigma}_r} (\delta_{k,z_j} - g_k(\vec{x}_j))^2 &= \frac{\partial}{\partial u_r(\vec{x}_j)} (\delta_{k,z_j} - \vec{w}_k^\top \vec{u}(\vec{x}_j))^2 \cdot \nabla_{\mathbf{\Sigma}_r} u_r(\vec{x}_j) \\ &= -2 \left(\delta_{k,z_j} - \vec{w}_k^\top \vec{u}(\vec{x}_j)\right) \cdot w_{kr} \cdot \nabla_{\mathbf{\Sigma}_r} u_r(\vec{x}_j).\end{aligned}$$

With these formulae the task is reduced to computing the gradients of the membership degrees w.r.t. the cluster centers and the covariance matrices.

If $\vec{u}(x) = \vec{u}^\circ(\vec{x})$, that is, if we use unnormalized and untransformed membership degrees (as it is usually the case in a linear function classifier as we consider it here), expressions for these gradients can be found in Section A.2.2 in the appendix. In this section the most common choices for radial functions yielding the membership degrees, namely the generalized Cauchy function as well as the generalized Gaussian function (cf. Section 2.2), are treated in detail. In particular, their derivatives w.r.t. the center vector μ and the covariance matrix $\mathbf{\Sigma}$ are computed.

Inserting these derivatives into the formulae derived above, we obtain for the generalized Cauchy function as the radial function $\forall i; 1 \leq i \leq c$:

$$\begin{aligned}\vec{\mu}_i^{(t+1)} \;=\; \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}}\, a \sum_{j=1}^{n} \sum_{k=1}^{s} \Big(\delta_{k,z_j} - \Big(\vec{w}_k^{(t)}\Big)^\top \vec{u}^{(t)}(\vec{x}_j)\Big) \cdot w_{ki}^{(t)} \\ \cdot \Big(u_i^{\circ(t)}(\vec{x}_j)\Big)^2 \cdot \Big(d\Big(\vec{x}_j, \vec{\mu}_i^{(t)}; \boldsymbol{\Sigma}_i^{(t)}\Big)\Big)^{a-2} \\ \cdot \Big(\boldsymbol{\Sigma}_i^{(t)}\Big)^{-1} \Big(\vec{x}_j - \vec{\mu}_i^{(t)}\Big)\end{aligned}$$

as an update rule for the cluster centers. Note that the distance factor vanishes for the important special case $a = 2$. Note also that several terms in this formula are already known from the execution of the classifier. Finally, note that the factor a can be incorporated into the learning rate $\eta_{\vec{\mu}}$.

Analogously, we get for the covariance matrices $\forall i; 1 \leq i \leq c$:

$$\begin{aligned}\Big(\boldsymbol{\Sigma}_i^{(t+1)}\Big)^{-1} \;=\; \Big(\boldsymbol{\Sigma}_i^{(t)}\Big)^{-1} - \eta_{\boldsymbol{\Sigma}} \frac{a}{2} \sum_{j=1}^{n} \sum_{k=1}^{s} \Big(\delta_{k,z_j} - \Big(\vec{w}_k^{(t)}\Big)^\top \vec{u}^{(t)}(\vec{x}_j)\Big) \cdot w_{ki}^{(t)} \\ \cdot \Big(u_i^{\circ(t)}(\vec{x}_j)\Big)^2 \cdot \Big(d\Big(\vec{x}_j, \vec{\mu}_i^{(t)}; \boldsymbol{\Sigma}_i^{(t)}\Big)\Big)^{a-2} \\ \cdot \Big(\vec{x}_j - \vec{\mu}_i^{(t)}\Big) \Big(\vec{x}_j - \vec{\mu}_i^{(t)}\Big)^\top.\end{aligned}$$

On the other hand, if the generalized Gaussian function is used as the radial function, we obtain for the cluster centers $\forall i; 1 \leq i \leq c$:

$$\begin{aligned}\vec{\mu}_i^{(t+1)} \;=\; \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}}\, \frac{a}{2} \sum_{j=1}^{n} \sum_{k=1}^{s} \Big(\delta_{k,z_j} - \Big(\vec{w}_k^{(t)}\Big)^\top \vec{u}^{(t)}(\vec{x}_j)\Big) \cdot w_{ki}^{(t)} \\ \cdot u_i^{\circ(t)}(\vec{x}_j) \cdot \Big(d\Big(\vec{x}_j, \vec{\mu}_i^{(t)}; \boldsymbol{\Sigma}_i^{(t)}\Big)\Big)^{a-2} \\ \cdot \Big(\boldsymbol{\Sigma}_i^{(t)}\Big)^{-1} \Big(\vec{x}_j - \vec{\mu}_i^{(t)}\Big).\end{aligned}$$

Note that this formula is very similar to the one above for the generalized Cauchy function. The only differences are an additional factor $\frac{1}{2}$ and the missing square at the membership degree $u_i^{\circ}(\vec{x}_j)$. (Note, however, that these membership degrees are different, because they are computed with the generalized Cauchy function or the generalized Gaussian function, depending on which function is used as the radial function.)

For the covariance matrices we get $\forall i; 1 \leq i \leq c$:

$$\left(\boldsymbol{\Sigma}_i^{(t+1)}\right)^{-1} = \left(\boldsymbol{\Sigma}_i^{(t)}\right)^{-1} - \eta_{\boldsymbol{\Sigma}} \frac{a}{4} \sum_{j=1}^{n} \sum_{k=1}^{s} \left(\delta_{k,z_j} - \left(\vec{w}_k^{(t)}\right)^\top \vec{u}^{(t)}(\vec{x}_j)\right) \cdot w_{ki}^{(t)} \cdot u_i^{\circ(t)}(\vec{x}_j) \cdot \left(d\left(\vec{x}_j, \vec{\mu}_i^{(t)}; \boldsymbol{\Sigma}_i^{(t)}\right)\right)^{a-2} \cdot \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)\left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)^\top.$$

Again this formula is very similar to the corresponding one for the generalized Cauchy function derived above. The differences are the same as for the cluster centers: an additional factor $\frac{1}{2}$ and a missing square.

The above considerations were restricted to the sum of *squared* errors, for which the technical task of computing the gradient is easiest. However, gradient descent may also be carried out for the sum of *absolute* errors, provided we complete the derivative of the absolute value, which does not exist at 0, in an appropriate way. For example, we may use the definition

$$\frac{\mathrm{d}}{\mathrm{d}x}|x| \stackrel{\text{def}}{=} \operatorname{sgn}(x) = \begin{cases} 1, & \text{if } x > 0, \\ -1, & \text{if } x < 0, \\ 0, & \text{otherwise.} \end{cases}$$

This approach is closely related to a training method that is known as **Manhattan training** in neural networks. Manhattan training uses only the sign of the gradient and a fixed step width for the update. Here, however, some information about the length of the gradient is preserved.

With this definition, we get for the cluster centers (in analogy to the considerations for squared errors above)

$$\begin{aligned} \nabla_{\vec{\mu}_r} |\delta_{k,z_j} - g_k(\vec{x}_j)| &= \frac{\partial}{\partial u_r(\vec{x}_j)} |\delta_{k,z_j} - \vec{w}_k^\top \vec{u}(\vec{x}_j)| \cdot \nabla_{\vec{\mu}_r} u_r(\vec{x}_j) \\ &= -\operatorname{sgn}(\delta_{k,z_j} - \vec{w}_k^\top \vec{u}(\vec{x}_j)) \cdot w_{kr} \cdot \nabla_{\vec{\mu}_r} u_r(\vec{x}_j) \end{aligned}$$

and for the covariance matrices

$$\begin{aligned} \nabla_{\boldsymbol{\Sigma}_r} |\delta_{k,z_j} - g_k(\vec{x}_j)| &= \frac{\partial}{\partial u_r(\vec{x}_j)} |\delta_{k,z_j} - \vec{w}_k^\top \vec{u}(\vec{x}_j)| \cdot \nabla_{\boldsymbol{\Sigma}_r} u_r(\vec{x}_j) \\ &= -\operatorname{sgn}(\delta_{k,z_j} - \vec{w}_k^\top \vec{u}(\vec{x}_j)) \cdot w_{kr} \cdot \nabla_{\boldsymbol{\Sigma}_r} u_r(\vec{x}_j). \end{aligned}$$

Inserting into these formulae the same expressions as above for the gradients of the membership degrees $\vec{u}(\vec{x}_j)$ then yields the update rules.

5.1.4 Gradient Ascent on Likelihood Function

Although the standard method for optimizing a likelihood function is the *expectation maximization algorithm*, which is discussed in detail in Section 5.2.4, gradient ascent may also be applied. Even though it has the drawback of being less robust (in part this is due to the parameter repairs that may be necessary, cf. Section 5.1.1), it can be faster than the expectation maximization algorithm, in particular close to the convergence point and if specific versions of gradient descent are used [Jamshidian and Jennrich 1993, Russel *et al.* 1995, Bauer *et al.* 1997, Salakhutdinov *et al.* 2003].

Formally, the task consists in the following: The likelihood function describes the likelihood of the data set w.r.t. the cluster parameters in $\mathbf{C}$. That is,

$$L(\mathbf{X}; \mathbf{C}) = f_{\mathcal{X}}(\mathbf{X}; \mathbf{C}) = \prod_{j=1}^{n} f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}).$$

As explained in Section 3.3, the probability density function $f_{\vec{X}}(\vec{x}; \mathbf{C})$ can be written as a sum of membership degrees. As a consequence we have

$$\begin{aligned} L(\mathbf{X}; \mathbf{C}) &= \prod_{j=1}^{n} \sum_{i=1}^{c} u_i^*(\vec{x}_j) = \prod_{j=1}^{n} \sum_{i=1}^{c} \varrho_i \cdot u_i^\circ(\vec{x}_j) \\ &= \prod_{j=1}^{n} \sum_{i=1}^{c} \varrho_i \cdot \gamma(a, b, m, \boldsymbol{\Sigma}_i) \cdot f_{\text{radial}}(d(\vec{x}_j, \vec{\mu}_i; \boldsymbol{\Sigma}_i); a, b). \end{aligned}$$

Here ϱ_i is the prior probability of the i-th cluster. $\gamma(a, b, m, \boldsymbol{\Sigma})$ is a normalization factor that scales the radial function in such a way that it can be interpreted as a probability density function (cf. page 19 in Section 2.2). That is, its value is defined by the requirement to satisfy the constraints

$$\forall i; 1 \leq i \leq c: \quad \int_{\mathbb{R}^m} \gamma(a, b, m, \boldsymbol{\Sigma}_i) \cdot f_{\text{radial}}(d(\vec{x}, \vec{\mu}_i; \boldsymbol{\Sigma}_i); a, b) \, \mathrm{d}\vec{x} = 1.$$

Since it simplifies the technical task of computing the gradients, one usually does not consider the above likelihood function, but its natural logarithm. That is, one tries to optimize the **log-likelihood** (cf. Section 3.3)

$$\ln L(\mathbf{X}; \mathbf{C}) = \sum_{j=1}^{n} \ln \left(\sum_{i=1}^{c} \varrho_i \cdot u_i^\circ(\vec{x}_j) \right).$$

This function has to be maximized, so we have to carry out a gradient *ascent*, which explains the plus signs in the following list of generic update

formulae for the cluster parameters (compare Sections 5.1.2 and 5.1.3):

$$\begin{aligned}
\varrho_i^{(t+1)} &= \varrho_i^{(t)} + \eta_\varrho \frac{\partial}{\partial \varrho_i} \ln L(\mathbf{X}; \mathbf{C}^{(t)}), \\
\vec{\mu}_i^{(t+1)} &= \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}} \nabla_{\vec{\mu}_i} \ln L(\mathbf{X}; \mathbf{C}^{(t)}), \\
\left(\mathbf{\Sigma}_i^{(t+1)}\right)^{-1} &= \left(\mathbf{\Sigma}_i^{(t)}\right)^{-1} + \eta_{\mathbf{\Sigma}} \nabla_{\mathbf{\Sigma}_i^{-1}} \ln L(\mathbf{X}; \mathbf{C}^{(t)}).
\end{aligned}$$

To obtain the exact update formulae, we have to compute—as usual—the different gradients appearing in these generic formulae. For the prior probabilities ϱ_i, $1 \leq i \leq c$, of the clusters we get

$$\frac{\partial}{\partial \varrho_r} \ln L(\mathbf{X}; \mathbf{C}) = \frac{\partial}{\partial \varrho_r} \sum_{j=1}^{n} \ln \left(\sum_{i=1}^{c} \varrho_i \cdot u_i^\circ(\vec{x}_j) \right) = \sum_{j=1}^{n} \frac{u_r^\circ(\vec{x}_j)}{\sum_{i=1}^{c} \varrho_i \cdot u_i^\circ(\vec{x}_j)}.$$

Therefore the update rule for the prior probabilities is $\forall i; 1 \leq i \leq c:$

$$\varrho_i^{(t+1)} = \varrho_i^{(t)} + \eta_\varrho \sum_{j=1}^{n} \frac{u_i^{\circ(t)}(\vec{x}_j)}{\sum_{k=1}^{c} \varrho_k^{(t)} \cdot u_k^{\circ(t)}(\vec{x}_j)}.$$

However, as for the elements of the fuzzy partition matrix in Section 5.1.2 (which must sum to 1 for each data point), we have to take the constraints

$$\forall i; 1 \leq i \leq c: \quad \varrho_i \in [0, 1] \qquad \text{and} \qquad \sum_{i=1}^{c} \varrho_i = 1$$

into account. To ensure that these constraints are satisfied, we may have to repair the result of a gradient ascent step. We do so in basically the same way as in Section 5.1.2, that is, we adapt negative prior probabilities to 0 or some small ε and prior probabilities greater than 1 to 1 or $1 - \varepsilon$. In addition, we may have to renormalize them to sum 1.

For the center vectors $\vec{\mu}_i$ we obtain generally

$$\begin{aligned}
\nabla_{\vec{\mu}_r} \ln L(\mathbf{X}; \mathbf{C}) &= \nabla_{\vec{\mu}_r} \sum_{j=1}^{n} \ln \left(\sum_{i=1}^{c} \varrho_i \cdot u_i^\circ(\vec{x}_j) \right) \\
&= \sum_{j=1}^{n} \frac{\varrho_r}{\sum_{i=1}^{c} \varrho_i \cdot u_i^\circ(\vec{x}_j)} . \nabla_{\vec{\mu}_r} u_r^\circ(\vec{x}_j) \\
&= \sum_{j=1}^{n} \frac{\varrho_r}{\sum_{i=1}^{c} \varrho_i \cdot u_i^\circ(\vec{x}_j)} \\
&\qquad \cdot \gamma(a, b, m, \mathbf{\Sigma}_r) \cdot \nabla_{\vec{\mu}_r} f_{\text{radial}}(d(\vec{x}_j, \vec{\mu}_r; \mathbf{\Sigma}_r); a, b).
\end{aligned}$$

Note that the normalization factor does not depend on any center vector and thus is simply reproduced. This is different for the derivatives w.r.t. the (inverse) covariance matrices:

$$\begin{aligned}
\nabla_{\mathbf{\Sigma}_r} \ln L(\mathbf{X}; \mathbf{C}) &= \nabla_{\mathbf{\Sigma}_r} \sum_{j=1}^{n} \ln \left(\sum_{i=1}^{c} \varrho_i \cdot u_i^{\circ}(\vec{x}_j) \right) \\
&= \sum_{j=1}^{n} \frac{\varrho_r}{\sum_{i=1}^{c} \varrho_i \cdot u_i^{\circ}(\vec{x}_j)} . \nabla_{\mathbf{\Sigma}_r} u_r^{\circ}(\vec{x}_j) \\
&= \sum_{j=1}^{n} \frac{\varrho_r}{\sum_{i=1}^{c} \varrho_i \cdot u_i^{\circ}(\vec{x}_j)} \\
&\quad \cdot \nabla_{\mathbf{\Sigma}_r^{-1}} \gamma(a, b, m, \mathbf{\Sigma}_r) \cdot f_{\text{radial}}(d(\vec{x}_j, \vec{\mu}_r; \mathbf{\Sigma}_r); a, b).
\end{aligned}$$

Here we have to apply the product rule to obtain

$$\begin{aligned}
&\nabla_{\mathbf{\Sigma}_r^{-1}} \gamma(a, b, m, \mathbf{\Sigma}_r) \cdot f_{\text{radial}}(d(\vec{x}_j, \vec{\mu}_r; \mathbf{\Sigma}_r); a, b) \\
&= \gamma(a, b, m, \mathbf{\Sigma}_r) \cdot \left(\nabla_{\mathbf{\Sigma}_r^{-1}} f_{\text{radial}}(d(\vec{x}_j, \vec{\mu}_r; \mathbf{\Sigma}_r); a, b) \right) \\
&+ \left(\nabla_{\mathbf{\Sigma}_r^{-1}} \gamma(a, b, m, \mathbf{\Sigma}_r) \right) \cdot f_{\text{radial}}(d(\vec{x}_j, \vec{\mu}_r; \mathbf{\Sigma}_r); a, b) \\
&= \gamma(a, b, m, \mathbf{\Sigma}_r) \cdot \left(\nabla_{\mathbf{\Sigma}_r^{-1}} f_{\text{radial}}(d(\vec{x}_j, \vec{\mu}_r; \mathbf{\Sigma}_r); a, b) \right) \\
&- \frac{1}{2} \underbrace{\gamma(a, b, m, \mathbf{\Sigma}_r) \cdot f_{\text{radial}}(d(\vec{x}_j, \vec{\mu}_r; \mathbf{\Sigma}_r); a, b)}_{= u_r^{\circ}(\vec{x}_j)} \cdot \mathbf{\Sigma}.
\end{aligned}$$

The derivative of the normalization factor used in this formula can be found in Section A.2.2 in the appendix. For the gradients of the radial function we also use again (as in the preceding section) the formulae derived in Section A.2.2 in the appendix. This yields as the update rule for the cluster centers for the generalized Cauchy function $\forall i; 1 \leq i \leq c$:

$$\begin{aligned}
\vec{\mu}_i^{(t+1)} &= \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}}\, a \sum_{j=1}^{n} \frac{\varrho_r}{\sum_{i=1}^{c} \varrho_i \cdot u_i^{\circ(t)}(\vec{x}_j)} \cdot \frac{1}{\gamma(a, b, m, \mathbf{\Sigma}_r)} \\
&\quad \cdot \left(u_i^{\circ(t)}(\vec{x}_j) \right)^2 \cdot \left(d\left(\vec{x}_j, \vec{\mu}_i^{(t)}; \mathbf{\Sigma}_i^{(t)} \right) \right)^{a-2} \\
&\quad \cdot \left(\mathbf{\Sigma}_i^{(t)} \right)^{-1} \left(\vec{x}_j - \vec{\mu}_i^{(t)} \right).
\end{aligned}$$

Note that here $u_i^{\circ}(\vec{x}_j)$ contains the normalization factor $\gamma(a, b, m, \mathbf{\Sigma}_r)$, in contrast to the preceding section, where this normalization factor was 1.

For the covariance matrices we have $\forall i; 1 \leq i \leq c$:

$$\left(\boldsymbol{\Sigma}_i^{(t+1)}\right)^{-1} = \left(\boldsymbol{\Sigma}_i^{(t)}\right)^{-1} - \eta_{\boldsymbol{\Sigma}} \frac{a}{2} \sum_{j=1}^{n} \frac{\varrho_r}{\sum_{i=1}^{c} \varrho_i \cdot u_i^{\circ(t)}(\vec{x}_j)} \cdot u_i^{\circ}(\vec{x}_j) \cdot \left(\frac{u_i^{\circ(t)}(\vec{x}_j)}{\gamma(a,b,m,\boldsymbol{\Sigma}_r)} \cdot \left(d\left(\vec{x}_j, \vec{\mu}_i^{(t)}; \boldsymbol{\Sigma}_i^{(t)}\right)\right)^{a-2} \cdot \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)\left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)^{\top} - \frac{1}{2}\boldsymbol{\Sigma} \right).$$

Note here the additional term $-\frac{1}{2}\boldsymbol{\Sigma}$ that results from the application of the product rule and then from the normalization factor.

On the other hand, if the generalized Gaussian function is used as the radial function, we obtain for the cluster centers $\forall i; 1 \leq i \leq c$:

$$\vec{\mu}_i^{(t+1)} = \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}} \frac{a}{2} \sum_{j=1}^{n} \frac{\varrho_r}{\sum_{i=1}^{c} \varrho_i \cdot u_i^{\circ(t)}(\vec{x}_j)} \cdot u_i^{\circ(t)}(\vec{x}_j) \cdot \left(d\left(\vec{x}_j, \vec{\mu}_i^{(t)}; \boldsymbol{\Sigma}_i^{(t)}\right)\right)^{a-2} \cdot \left(\boldsymbol{\Sigma}_i^{(t)}\right)^{-1} \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right).$$

This formula is very similar to the one above for the generalized Cauchy function. The only differences are an additional factor $\frac{1}{2}$ and the missing square at the membership degree $u_i^{\circ}(\vec{x}_j)$. (Note, however, that these membership degrees are different, because they are computed with the generalized Cauchy function or the generalized Gaussian function, depending on which function is used as the radial function.)

For the covariance matrices we get $\forall i; 1 \leq i \leq c$:

$$\left(\boldsymbol{\Sigma}_i^{(t+1)}\right)^{-1} = \left(\boldsymbol{\Sigma}_i^{(t)}\right)^{-1} - \eta_{\boldsymbol{\Sigma}} \frac{a}{4} \sum_{j=1}^{n} \frac{\varrho_r}{\sum_{i=1}^{c} \varrho_i \cdot u_i^{\circ(t)}(\vec{x}_j)} \cdot u_i^{\circ(t)}(\vec{x}_j) \cdot \left(\left(d\left(\vec{x}_j, \vec{\mu}_i^{(t)}; \boldsymbol{\Sigma}_i^{(t)}\right)\right)^{a-2} \cdot \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)\left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)^{\top} - \frac{1}{2}\boldsymbol{\Sigma} \right).$$

Again this formula is very similar to the corresponding one for the generalized Cauchy function derived above. The differences are the same as for the cluster centers: an additional factor $\frac{1}{2}$ and a missing square.

I skip a detailed consideration of a gradient ascent on the two versions of the **likelihood ratio** that were considered in Section 3.4. The reason is that by applying the same trick as for the likelihood function, namely by taking the natural logarithm, these objective functions become

$$\ln L^{(1)}_{\text{ratio}}(\mathbf{X},\vec{z};\mathbf{C},\mathbf{W}) = \sum_{j=1}^{n} \ln f_{\vec{X}_j,Z_j}(\vec{x}_j,z_j;\mathbf{C},\mathbf{W}) - \sum_{j=1}^{n} \ln f_{\vec{X}_j,Z_j}(\vec{x}_j,\overline{z_j};\mathbf{C},\mathbf{W})$$

and

$$\ln L^{(2)}_{\text{ratio}}(\mathbf{X},\vec{z};\mathbf{C},\mathbf{W}) = \sum_{j=1}^{n} \ln f_{\vec{X}_j,Z_j}(\vec{x}_j,z_j;\mathbf{C},\mathbf{W}) - \sum_{j=1}^{n} \ln f_{\vec{X}_j}(\vec{x}_j;\mathbf{C},\mathbf{W}),$$

respectively (cf. Section 3.4). Since the class indicators z_j or $\overline{z_j}$ only specify a restriction to a subset of the clusters (cf. Section 3.4), these log-likelihood ratios can be handled with basically the same formulae as the likelihood function itself (cf. also Section 5.3.4 and [Seo and Obermayer 2003]).

5.1.5 Problems of Gradient Methods

There are two main problems of gradient methods. The first is the **choice of the learning rate**. This choice is critical as is illustrated in Figures 5.2 and 5.3, which show a simple gradient descent on the polynomial[3]

$$f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6.$$

To carry out a gradient descent on this function, we compute its derivative

$$f'(x) = \frac{10}{3}x^3 - 21x^2 + \frac{115}{3}x - 18,$$

which corresponds to the gradient (the sign of this function indicates the direction of the steepest ascent, its value how steep the ascent is). Then a gradient descent is carried out according to

$$x_{i+1} = x_i + \Delta x_i \qquad \text{with} \qquad \Delta x_i = -\eta f'(x_i),$$

where x_0 is the chosen starting point (initialization) and η the learning rate. Figures 5.2 and 5.3 show this gradient descent with different starting points and different learning rates.

[3]Note that this function has no connection to any of the objective functions discussed here. It is used here only to illustrate the general problems of gradient descent.

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	0.200	3.112	−11.147	0.011
1	0.211	2.990	−10.811	0.011
2	0.222	2.874	−10.490	0.010
3	0.232	2.766	−10.182	0.010
4	0.243	2.664	−9.888	0.010
5	0.253	2.568	−9.606	0.010
6	0.262	2.477	−9.335	0.009
7	0.271	2.391	−9.075	0.009
8	0.281	2.309	−8.825	0.009
9	0.289	2.233	−8.585	0.009
10	0.298	2.160		

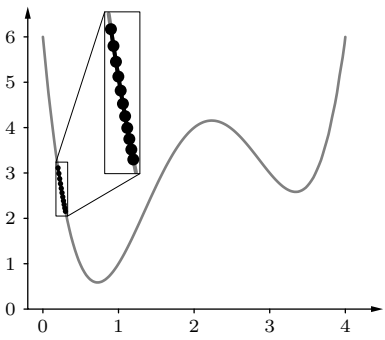

Figure 5.2: Gradient descent with initial value 0.2 and learning rate 0.001.

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	1.500	2.719	3.500	−0.875
1	0.625	0.655	−1.431	0.358
2	0.983	0.955	2.554	−0.639
3	0.344	1.801	−7.157	1.789
4	2.134	4.127	0.567	−0.142
5	1.992	3.989	1.380	−0.345
6	1.647	3.203	3.063	−0.766
7	0.881	0.734	1.753	−0.438
8	0.443	1.211	−4.851	1.213
9	1.656	3.231	3.029	−0.757
10	0.898	0.766		

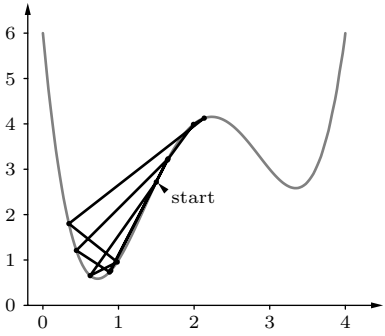

Figure 5.3: Gradient descent with initial value 1.5 and learning rate 0.25.

In Figure 5.2 the learning rate is too small and thus the minimum is approached very slowly. Even though it is clear that it will be reached finally, it takes a (too) large number of steps. On the other hand, in Figure 5.3 the learning rate is chosen too large, so that the current point in the parameter space (here the set $\mathbb{R}$ of real numbers) "jumps around," leading to oscillatory behavior. If some more steps are carried out than those shown in the figure, the process even leaves the region of the global minimum on the left and updates the parameter value to the region of the local minimum on the right. This is definitely not an acceptable behavior.

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	2.600	3.816	−1.707	0.085
1	2.685	3.660	−1.947	0.097
2	2.783	3.461	−2.116	0.106
3	2.888	3.233	−2.153	0.108
4	2.996	3.008	−2.009	0.100
5	3.097	2.820	−1.688	0.084
6	3.181	2.695	−1.263	0.063
7	3.244	2.628	−0.845	0.042
8	3.286	2.599	−0.515	0.026
9	3.312	2.589	−0.293	0.015
10	3.327	2.585		

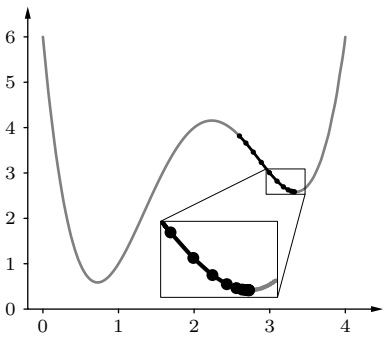

Figure 5.4: Gradient descent with initial value 2.6 and learning rate 0.05.

As a consequence the learning rate should be chosen with care, so that the (local) minimum is approached rapidly without the danger of running into instabilities like oscillations. This choice is difficult, because the optimal value of the learning rate depends on the particular objective function to be optimized and often enough also on the region in the parameter space in which the current parameter vector lies. To cope with this problem, adaptive methods have been devised, which try to adapt the learning rate (or directly the length of the step that is carried out in parameter space) depending on the preceding steps and the current gradient of the objective function. These methods are reviewed in Section 6.2. They are discussed in a separate chapter, because they are not only applicable to gradient methods, but to basically all update methods discussed in this chapter.

The second fundamental problem connected with gradient ascent or descent is that they **may get stuck in a local optimum**. This is demonstrated in Figure 5.4: the learning rate is chosen well so that the minimum is approached quickly. However, due to the unfortunate location of the starting point this minimum is only a local minimum.

No general method (in the sense of a modification of gradient ascent or descent) is known that ensures that this does not happen and thus guarantees that the global optimum of the objective function is found. As already mentioned in Section 4.3, the most common way of trying to cope with this problem is to run the clustering algorithm several times with different initializations. This yields several clustering results, of which the one having the best value for the objective function is chosen as the final result [Duda

and Hart 1973]. Of course, this approach does not rule out the possibility of getting stuck in a local optimum. It only improves the chances of finding the global optimum or at least of finding a sufficiently good local optimum. In practice, however, this is good enough and thus it is not surprising that the method has been used successfully in several applications.

5.2 Alternating Optimization

Alternating optimization is the standard parameter update scheme in the area of (fuzzy) c-means clustering. In addition, the expectation maximization algorithm for estimating the parameters of a mixture of Gaussians is very closely related to alternating optimization. Hence it is also treated in this section, even though it deviates slightly from the general scheme.

Alternating optimization has the advantage that it is an extremely robust update method. In several of its variants it is guaranteed to converge to a (local) optimum of the objective function. However, gradient descent is sometimes preferable, because it can be faster, especially close to the convergence point, where alternating optimization tends to be slow.

5.2.1 General Approach

The general idea of alternating optimization is to split the set of parameters to optimize into two (or more) subsets. This is done in such a way that the optimum for one set of parameters can be computed directly, provided the other set(s) are held constant. This property is exploited to optimize the full set of parameters by alternatingly optimizing (hence the name) the parameters in each set while the others are fixed.

Formally, alternating optimization can be described as follows for two sets of parameters: ket $f(\vec{\theta}_1, \vec{\theta}_2)$ be an objective function that has two parameter vectors $\vec{\theta}_1$ and $\vec{\theta}_2$ as its arguments. Together these two vectors should contain all parameters of the objective function and they must not have any parameters in common. In order to find a (local) optimum of the function f w.r.t. $\vec{\theta}_1$ and $\vec{\theta}_2$, we determine an initial parameter vector $\vec{\theta}_2^{(0)}$. Then we compute the sequence of vectors

$$\begin{aligned} \vec{\theta}_1^{(t+1)} &= \underset{\vec{\theta}_1}{\operatorname{argopt}} f\left(\vec{\theta}_1, \vec{\theta}_2^{(t)}\right), \\ \vec{\theta}_2^{(t+1)} &= \underset{\vec{\theta}_2}{\operatorname{argopt}} f\left(\vec{\theta}_1^{(t+1)}, \vec{\theta}_2\right), \qquad t = 0, 1, 2, \ldots, \end{aligned}$$

where argopt is either argmax (if the objective function f is to be maximized) or argmin (if it is to be minimized). Of course, this scheme presupposes that both optima can be computed directly.

As for gradient descent, situations may arise in which the objective function has its optimum at a point that does not satisfy the given constraints on the parameters. In such a case alternating optimization has the advantage that we need not apply repair mechanisms in order to force the parameter vector back into the allowed region, but have a much more elegant method at hand. It consists in the well-known method of **Lagrange multipliers**, which is reviewed in some detail in Section A.8 in the appendix. The general idea is that constraints of a certain type can be incorporated into the objective function and thus the optimization process can be guaranteed to yield a solution that respects the constraints.

5.2.2 Classical Crisp Clustering

Classical crisp clustering is best known under the name of c-means (or k-means) clustering, where c (or k) is the number of clusters to be found. It is based on the objective function of the sum of (squared) distances defined in Section 3.1, with a hard assignment of the data points to the clusters. That is, each data point is assigned to one cluster and one cluster only.

At first sight this objective function may seem to be easy to optimize. However, a careful analysis reveals that the problem of hard c-means clustering is NP-hard even for $c = 2$ [Drineas *et al.* 2004]. That is, there is no known algorithm that finds the optimal partition in a time that is polynomial in the size of the input (that is, the number of data points). The proof is carried out by a reduction from the problem of minimum bisection, that is, the problem of partitioning a given graph into two parts of equal size so that the number of edges between these two parts is minimized [Drineas *et al.* 2004]. For larger values of c the same result can be obtained by a reduction from the problem of minimum c-section.

However, as is also shown in [Drineas *et al.* 2004], a 2-approximation of the optimum can be achieved in polynomial time, that is, a result for which the value of the objective function is at most twice as large as the optimum. The corresponding algorithm, which I do not discuss in detail here, starts by projecting the given data points to a cleverly chosen subspace of dimensionality c (the number of clusters). This subspace is determined by a singular value decomposition of the data matrix and selecting the singular vectors corresponding to the largest singular values. [Drineas *et al.* 2004] show that there is a linear time Monte Carlo method for approximating the largest

singular values, which renders this step efficient. The algorithm then finds the solution of the crisp clustering problem in this subspace by enumerating and checking all relevant partitions of the projected data points. However, for practical purposes this is usually not a feasible algorithm. One reason is that it seems to require $c \leq m$, where m is the number of dimensions of the data space, another is that the time complexity of the enumeration of all relevant partitions is too high for larger values of c.

As a consequence approaches like alternating optimization are the methods of choice. Even though they cannot be guaranteed to find the optimal solution as they may get stuck in a local optimum, they work very well in practice. The alternating optimization scheme of c-means clustering works as follows: after initial cluster centers have been chosen (see Chapter 4 for a discussion of different methods), the data points are first partitioned by assigning each data point to the closest cluster center. In a second step new cluster centers are computed as the centers of gravity of the data points in each cluster, that is, the data points that have been assigned to the same cluster center. These two steps are iterated until the partition of the data points does not change anymore. This iterative procedure, which is guaranteed to converge, is also known as the **ISODATA algorithm** [Ball and Hall 1966, Ball and Hall 1967] or as **Lloyd's algorithm** [Lloyd 1982].

As an example Figure 5.5 shows the execution of c-means clustering on a very simple two-dimensional data set. The data set itself is shown in the top left diagram. Obviously, it contains three clusters. Hence the algorithm is initialized in the middle diagram in the top row by randomly selecting three data points as the initial cluster centers. In this diagram the colors of the data points as well as the grey lines show the initial partition of the data points. The grey lines actually form a **Voronoi diagram** [Preparata and Shamos 1985, Aurenhammer 1991] of the three initial cluster centers. It is a partition of the data space into three regions, called **Voronoi cells**, with one cluster center per cell. Each region consists of those points that are closer to the cluster center contained in the region than to any other cluster center and thus also indicates the partition of the data points.[4]

[4]The Voronoi diagram of a set of points (e.g., a set of cluster centers) can easily be computed from a so-called *Delaunay triangulation* [Preparata and Shamos 1985] of this set of points. A **triangulation** is a maximal set of non-intersecting edges with end points in the given set of points. It structures the point set into triangles. A **Delaunay triangulation** has the special property that the circle through the corners of each triangle does not contain another point from the set. The Voronoi diagram is formed by the mid-perpendiculars of the edges of the Delaunay triangulation. Both Voronoi diagrams and Delaunay triangulations can be generalized to more than two dimensions by replacing the terms "triangle" and "circle" with "simplex" and "(hyper-)sphere", respectively.

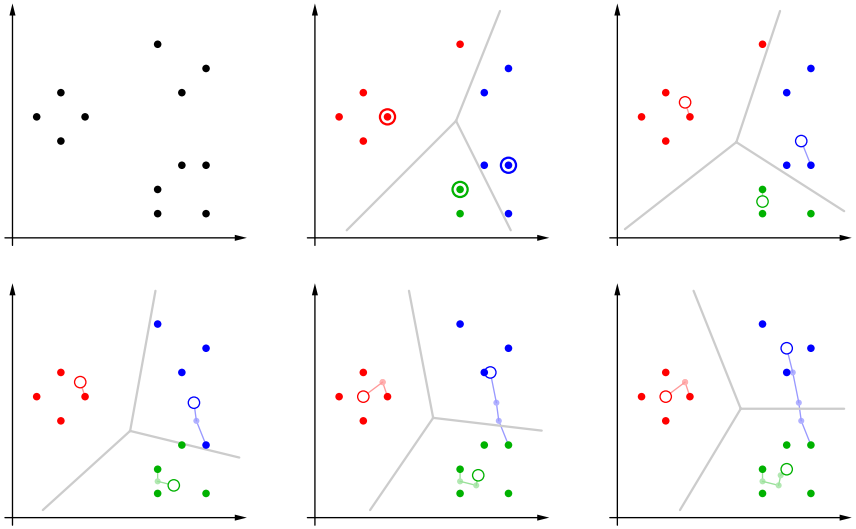

Figure 5.5: Hard c-means clustering of a simple example data set.

In the right diagram in the top row new cluster centers have been computed as the centers of gravity of those data points that were assigned to a cluster center. The colors of the data points and the grey lines have been adapted to show the new partition of the data points. The bottom row of Figure 5.5 shows the next steps, each consisting of a recomputation of the cluster centers and a new assignment of the data points. The rightmost diagram in the bottom row shows the convergence point: here a recomputation of the cluster centers and a new assignment of the data points does not lead to any (further) change. Obviously the desired result is obtained: each cluster is captured by one cluster center.

Unfortunately, hard c-means does not always work as smoothly as in this example. This is demonstrated in Figure 5.6: the diagrams in the top row show different initialization of the algorithm on the same data set, the corresponding diagram in the bottom row shows the convergence point reached from this initialization. (The second and third initialization differ only in the assignment of the data point in the bottom right corner, which lies exactly on the border between two Voronoi cells, so that the tie has to be broken randomly.) As can be seen, the desired result is reached in none of these cases, because the algorithm gets stuck in a local optimum.

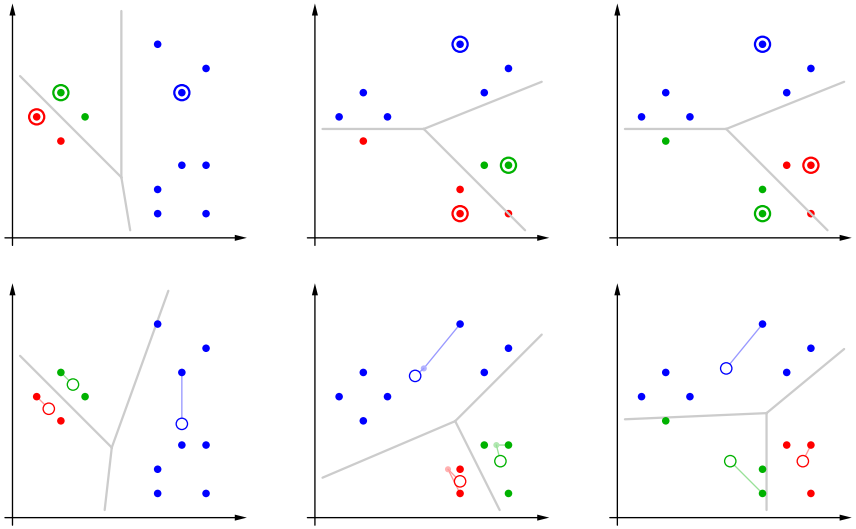

Figure 5.6: Hard c-means clustering can easily get stuck in local minima.

The tendency of hard c-means clustering to get stuck in a local optimum is very annoying, as it makes it often necessary to carry out several clustering runs, from which the best result is then selected [Duda and Hart 1973]. As a consequence, fuzzy c-means approaches—as they are discussed in the next section—are usually preferable, because they tend to behave much nicer w.r.t. local optima and also possess some other favorable characteristics. Although there is no formal proof of a superior robustness of fuzzy clustering algorithms yet, there are plausible conjectures that the objective function of fuzzy c-means clustering has fewer local optima, which are also "flattened out" by the fuzzifier (i.e., the weighting exponent for the membership degrees). Furthermore, experimental results clearly speak in favor of fuzzy clustering: there are several data sets on which hard c-means clustering does not succeed in a certain percentage of all cases, while fuzzy c-means clustering has never been observed to fail on them.

As a final remark I would like to point out that I introduced the two alternating steps of hard c-means clustering simply as a computation rule without deriving it from the objective function. Such a derivation is, of course, possible. However, it is most conveniently treated as a special case of fuzzy c-means clustering and thus it is postponed to the next section.

5.2.3 Fuzzy Clustering

The two sets of parameters that are optimized alternatingly in fuzzy clustering are the degrees of membership on the one hand (which corresponds to the assignment of the data points in hard clustering) and the cluster parameters (center vectors, covariance matrices) on the other. In the following I consider the corresponding two update steps in this order. In doing so I do not treat—as it is commonly done—fuzzy c-means clustering separately from more sophisticated approaches like the Gustafson–Kessel algorithm, either in its general form [Gustafson and Kessel 1979] or in its axes-parallel version [Klawonn and Kruse 1997]. The reason is that fuzzy c-means clustering can be seen as a special case of a more general approach, in which the covariance matrices are all fixed to the unit matrix. Similarly, axes-parallel versions restrict the covariance matrices to diagonal matrices, as explained in detail in Section 2.1. However, I consider explicitely possibilistic clustering (in order to show how it is easily obtained by simply parameterizing the standard scheme) as well as the modified update formulae that result from an alternative to the standard fuzzifier [Klawonn and Höppner 2003].

Update of the Membership Degrees

When deriving the update rule for the membership degrees, we have to take into account that for each data point they should sum to one over the different clusters. Following the general approach as it was explained in Section 5.2.1, we do so with the help of Lagrange multipliers (cf. Section A.8 in the appendix for a fairly detailed review of this mathematical technique). For the task at hand, we have to introduce in all n Lagrange multipliers λ_j, $1 \leq j \leq n$, that is, one multiplier per data point for the constraint referring to that data point. This yields the Lagrange function

$$\mathcal{L}(\mathbf{X}, \mathbf{C}, \mathbf{U}, \Lambda) = \underbrace{\sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{w} \, d_{ij}^2}_{=J(\mathbf{X},\mathbf{C},\mathbf{U})} + \sum_{j=1}^{n} \lambda_j \left(1 - \sum_{i=1}^{c} u_{ij} \right),$$

which is to be minimized. (Note that the function $J(\mathbf{X}, \mathbf{C}, \mathbf{U})$ is the objective function of fuzzy clustering as it was defined on page 51 in Section 3.1.) A necessary condition for a minimum of the Lagrange function is that the partial derivatives w.r.t. the membership degrees vanish, i.e.,

$$\frac{\partial}{\partial u_{kl}} \mathcal{L}(\mathbf{X}, \mathbf{C}, \mathbf{U}, \Lambda) = w \, u_{kl}^{w-1} \, d_{kl}^2 - \lambda_l \stackrel{!}{=} 0,$$

which leads to $\forall i; 1 \leq i \leq c$: $\forall j; 1 \leq j \leq n$:

$$u_{ij} = \left(\frac{\lambda_j}{w\, d_{ij}^2} \right)^{\frac{1}{w-1}}.$$

Summing these equations over the clusters (in order to be able to exploit the corresponding constraints on the membership degrees), we get

$$1 = \sum_{i=1}^{c} u_{ij} = \sum_{i=1}^{c} \left(\frac{\lambda_j}{w\, d_{ij}^2} \right)^{\frac{1}{w-1}}.$$

Consequently the λ_j, $1 \leq j \leq n$, are

$$\lambda_j = \left(\sum_{i=1}^{c} \left(w\, d_{ij}^2 \right)^{\frac{1}{1-w}} \right)^{1-w}.$$

Inserting this into the equation for the membership degrees yields
$\forall i; 1 \leq i \leq c$: $\forall j; 1 \leq j \leq n$:

$$u_{ij}^{(t+1)} = \frac{\left(d_{ij}^{(t)} \right)^{\frac{2}{1-w}}}{\sum_{k=1}^{c} \left(d_{kj}^{(t)} \right)^{\frac{2}{1-w}}},$$

with the special (and most common) case $w = 2$ leading to

$$u_{ij}^{(t+1)} = \frac{\left(d_{ij}^{(t)} \right)^{-2}}{\sum_{k=1}^{c} \left(d_{kj}^{(t)} \right)^{-2}}.$$

Note that this update formula results regardless of the distance measure $d(\vec{x}, \vec{\mu}; \mathbf{\Sigma})$. Note also that it is actually a very intuitive result: the membership degrees reflect the relative (squared) distances of the data point to the different cluster centers, which is what one would expect.

In the above derivation w may not be set to 1, because then we get

$$\frac{\partial}{\partial u_{kl}} \mathcal{L}(\mathbf{X}, \mathbf{C}, \mathbf{U}, \Lambda) = d_{kl}^2 - \lambda_l \stackrel{!}{=} 0.$$

This set of equations cannot be satisfied in general (with the exception of the special case in which all distances are equal). However, that the formal approach fails is actually not surprising, because the objective function does not have a minimum in this case. Even respecting the constraint, the value

of the objective function can be made arbitrarily small by choosing negative membership degrees for all distances except the minimal one. We would have to incorporate the constraint $u_{ij} \in [0,1]$ in order to get a result. However, we may also get a reasonable result by using $|u_{ij}|$ instead of u_{ij} in the objective function, which does not make a difference for the allowed range of values for the u_{ij}, but ensures that the objective function has a minimum in the allowed region of the parameter space. Using the convenient definition $\frac{\mathrm{d}}{\mathrm{d}x}|x| = \mathrm{sgn}(x)$ (cf. page 61), we obtain in this case

$$\frac{\partial}{\partial u_{kl}} \mathcal{L}(\mathbf{X}, \mathbf{C}, \mathbf{U}, \Lambda) = \mathrm{sgn}(u_{kl}) \left(d_{kl}^2 - \lambda_l\right) \stackrel{!}{=} 0.$$

From this set of equations it follows that for a given data point $\vec{x}_j$ all u_{ij}, $1 \leq i \leq c$, have to be zero, except one. Drawing on the considerations on page 49 in Section 3.1, we can even conclude that it should be $u_{kj} = 1$ for $k = \mathrm{argmin}_{i=1}^{c} d_{ij}^2$. That is, the data point should be assigned exclusively to the closest cluster center. However, this is exactly the assignment rule as it is used for hard c-means clustering, which shows that hard c-means results from fuzzy clustering in the limit for $w \to 1_+$.

If we consider the update rule that results for $w > 1$ (and in particular the special case $w = 2$), we observe that this rule may also be written as

$$u_{ij}^{(t+1)} = \frac{f_{\text{Cauchy}}\left(d_{ij}^{(t)}; \frac{2}{w-1}, 0\right)}{\sum_{k=1}^{c} f_{\text{Cauchy}}\left(d_{kj}^{(t)}; \frac{2}{w-1}, 0\right)} = \frac{u_i^{\circ(t)}(\vec{x}_j)}{\sum_{k=1}^{c} u_k^{\circ(t)}(\vec{x}_j)} = u_i^{(t)}(\vec{x}_j).$$

Even though this formula presupposes a specific choice of the radial function, it suggests the idea to generalize the update rule to

$$u_{ij}^{(t+1)} = u_i^{(t)}(\vec{x}_j),$$

independent of the chosen radial function and whether the membership degrees are weighted or not and transformed or not. (Note that the factor $\frac{1}{w-1}$ in the exponent of the distance may also be shifted to the transformation of the membership degrees as it was described in Section 2.4.)

Such a generalization is indeed possible: as pointed out above, the update rule results regardless of the chosen distance measure. Therefore we may define the objective function as (cf. page 47 in Section 3.1)

$$J(\mathbf{X}, \mathbf{C}, \mathbf{U}) = \sum_{i=1}^{c} \sum_{j=1}^{n} \frac{u_{ij}^w}{u_i^\bullet(\vec{x}_j)}$$

and interpret $(u_i^\bullet(\vec{x}_j))^{-1}$ as a (squared) distance measure.

Practical tests show that such a generalization is indeed feasible and leads to reasonable results that are often preferable to the results obtained with the standard method (which relies on relative inverse squared distances). In particular, in can be observed that choosing the Gaussian radial function (instead of the Cauchy radial function) in the above update formula makes the clustering algorithm much more robust w.r.t. informationless attributes[5] [Döring *et al.* 2005]. While the Cauchy radial function leads to a pronounced tendency of clusters merging into one if there are too many informationless attributes, no such tendency can be observed with the Gaussian radial function. The reasons for this behavior were already given in Section 2.4, in which the effects of a normalization of the membership degrees to sum 1 were studied in detail. With the Cauchy function larger distances lead to a less extreme assignment of a data point to a cluster (in the limit the membership degrees are all equal), while it gets more pronounced with the Gaussian function (in the limit we get a hard assignment to the closest cluster). As a consequence additional informationless dimensions, which mainly have the effect of increasing the average distance to the cluster centers, are no problem for the Gaussian function, but a severe problem for the Cauchy function. Although this effect can be mitigated by reducing the fuzzifier to values closer to 1, it cannot be eliminated completely.

Up to now I considered only the standard membership transformation $h(u) = u^w$ in the objective function. However, as discussed in Section 3.1, this transformation can be generalized to any convex function $[0, 1] \to [0, 1]$. A very useful special case of such a convex transformation function was suggested in [Klawonn and Höppner 2003], namely

$$h(u_{ij}) = \alpha u_{ij}^2 + (1 - \alpha) u_{ij}$$

or equivalently, but easier to interpret,

$$h(u_{ij}) = \frac{1 - \beta}{1 + \beta} u_{ij}^2 + \frac{2\beta}{1 + \beta} u_{ij},$$

where $\beta = \frac{1-\alpha}{1+\alpha}$ or $\alpha = \frac{1-\beta}{1+\beta}$. With this transformation the objective function reads (cf. page 52 in Section 3.1)

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} \left(\frac{1 - \beta}{1 + \beta} u_{ij}^2 + \frac{2\beta}{1 + \beta} u_{ij} \right) d_{ij}^2.$$

[5] An attribute is considered to be informationless if it does not provide any information about the split of a set of data points into clusters, that is, if the distribution of its values is basically the same for all clusters, so that they cannot be distinguished.

Introducing Lagrange multipliers as above in order to incorporate the constraint that the membership degrees must sum to 1 for each data point, we obtain the Lagrange function

$$\mathcal{L}(\mathbf{X}, \mathbf{U}, \mathbf{C}, \Lambda) = \underbrace{\sum_{i=1}^{c} \sum_{j=1}^{n} \left(\frac{1-\beta}{1+\beta} u_{ij}^2 + \frac{2\beta}{1+\beta} u_{ij} \right) d_{ij}^2}_{J(\mathbf{X}, \mathbf{U}, \mathbf{C})} + \sum_{j=1}^{n} \lambda_j \left(1 - \sum_{i=1}^{c} u_{ij} \right).$$

Again it is a necessary condition for a minimum that the partial derivatives w.r.t. the membership degrees vanish, that is, $\forall k; 1 \leq k \leq c : \forall l; 1 \leq l \leq n :$

$$\frac{\partial}{\partial u_{kl}} \mathcal{L}(\mathbf{X}, \mathbf{U}, \mathbf{C}, \Lambda) = \left(\frac{1-\beta}{1+\beta} 2u_{kl} + \frac{2\beta}{1+\beta} \right) d_{kl}^2 - \lambda_l \stackrel{!}{=} 0.$$

As a consequence we have $\forall i; 1 \leq i \leq c : \forall j; 1 \leq j \leq n :$

$$\lambda_j = \frac{(2(1-\beta)u_{ij} + 2\beta)\, d_{ij}^2}{1+\beta}$$

and therefore $\forall i; 1 \leq i \leq c : \forall j; 1 \leq j \leq n :$

$$u_{ij} = \frac{(1+\beta)\lambda_j d_{ij}^{-2} - 2\beta}{2(1-\beta)}.$$

At this point we have to pay attention to the fact that this expression for the u_{ij} may be negative for large distances d_{ij}, because then it may be $(1+\beta)\lambda_j d_{ij}^{-2} < 2\beta$. However, by definition, a membership degree u_{ij} cannot be negative. Hence one has to bear in mind that this formula may only be used if is yields a positive value, and $u_{ij} = 0$ otherwise. This is expressed in the following by restricting certain sums to terms for which $u_{ij} > 0$.

As in the derivation for the membership transformation $h(u) = u^w$, we sum these equations over all clusters in order to be able to exploit the constraint on the membership degrees. This yields

$$1 = \sum_{i=1}^{c} u_{ij} = \sum_{i:u_{ij}=0} 0 + \sum_{i:u_{ij}>0} \frac{(1+\beta)\lambda_j d_{ij}^{-2} - 2\beta}{2(1-\beta)}.$$

From this equation follows $\forall j; 1 \leq j \leq n :$

$$\lambda_j = \frac{2(1+\beta(\check{c}_j - 1))}{(1+\beta)\sum_{i=1;u_{ij}>0}^{c} d_{ij}^{-2}},$$

where $\check{c}_j$ is the number of clusters for which $u_{ij} > 0$, $1 \leq i \leq c$. Therefore

$$\begin{aligned} u_{ij}^{(t+1)} &= \frac{1}{1-\beta} \left(\frac{1+\beta\left(\check{c}_j^{(t)}-1\right)}{\sum_{k:u_{kj}^{(t+1)}>0} \left(d_{kj}^{(t)}\right)^{-2}} \left(d_{ij}^{(t)}\right)^{-2} - \beta \right) \\ &= \frac{1+\beta\left(\check{c}_j^{(t)}-1\right)}{(1-\beta)\sum_{k:u_{kj}^{(t+1)}>0} \left(d_{kj}^{(t)}\right)^{-2}} \\ &\quad \cdot \left(\left(d_{ij}^{(t)}\right)^{-2} - \frac{\beta}{1+\beta\left(\check{c}_j^{(t)}-1\right)} \sum_{k:u_{kj}^{(t+1)}>0} \left(d_{kj}^{(t)}\right)^{-2} \right), \end{aligned}$$

provided, of course, that this expression yields a positive membership degree (see above). Based on the fact that for such positive membership degrees the right factor must be positive, we can now determine $\check{c}_j$: the distances d_{ij} are sorted ascendingly, which may be described formally by a mapping function $\varsigma : \{1, \ldots, c\} \to \{1, \ldots c\}$ for the cluster indices such that

$$\forall i; 1 \leq i < c : \quad d_{\varsigma(i)j} \leq d_{\varsigma(i+1)j}.$$

With this function ς we can define $\check{c}_j$, $1 \leq j \leq n$, as

$$\check{c}_j = \max \left\{ k \;\middle|\; d_{\varsigma(k)j}^{-2} > \frac{\beta}{1+\beta(k-1)} \sum_{i=1}^{k} d_{\varsigma(i)j}^{-2} \right\}.$$

The membership degrees are thus $\forall i; 1 \leq i \leq c : \forall j; 1 \leq j \leq n :$

$$u_{ij}^{(t+1)} = \frac{u_{ij}'^{(t+1)}}{\sum_{k=1}^{c} u_{kj}'^{(t+1)}}$$

where

$$u_{ij}'^{(t+1)} = \max \left\{ 0, \; \left(d_{ij}^{(t)}\right)^{-2} - \frac{\beta}{1+\beta(\check{c}_j^{(t)}-1)} \sum_{k=1}^{\check{c}_j^{(t)}} \left(d_{\varsigma(k)j}^{(t)}\right)^{-2} \right\}.$$

This formula for the u_{ij}', which was already mentioned in Section 2.4 (cf. page 26), justifies the alternative transformation of the membership degrees described in Section 2.4, even though it is slightly different (cf. page 23).

To check that the above computation of the membership degrees in two steps (first computing the u'_{ij} and then normalizing them to sum 1) leads to the same result as the formula stated earlier, we compute

$$\begin{aligned}
\sum_{i=1}^{c} u'_{ij} &= \sum_{i=1}^{\check{c}_j} u'_{\varsigma(i)j} + \sum_{i=\check{c}_j+1}^{c} 0 \\
&= \sum_{i=1}^{\check{c}_j} \left(d_{s(i)j}^{-2} - \frac{\beta}{1+\beta(\check{c}_j-1)} \sum_{k=1}^{\check{c}_j} d_{\varsigma(k)j}^{-2} \right) \\
&= \sum_{i=1}^{\check{c}_j} d_{\varsigma(i)j}^{-2} - \frac{\beta\,\check{c}_j}{1+\beta(\check{c}_j-1)} \sum_{k=1}^{\check{c}_j} d_{\varsigma(k)j}^{-2} \\
&= \left(1 - \frac{\beta\,\check{c}_j}{1+\beta(\check{c}_j-1)}\right) \sum_{k=1}^{\check{c}_j} d_{s(k)j}^{-2} \\
&= \frac{1-\beta}{1+\beta(\check{c}_j-1)} \sum_{k=1}^{\check{c}_j} d_{\varsigma(k)j}^{-2}.
\end{aligned}$$

Obviously, this expression is the reciprocal of the first factor in the original formula for the $u_{ij}^{(t)}$ and thus the two steps yield the same result.

Finally, we have to consider the update of the membership degrees in possibilistic fuzzy clustering, which drops the constraint on the membership degrees and uses the objective function (cf. page 53 in Section 3.1)

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{w} d_{ij}^{2} + \sum_{i=1}^{c} \nu_i \sum_{j=1}^{n} (1-u_{ij})^{w},$$

where the ν_i, $1 \leq i \leq c$, are (constant) cluster-specific penalty weights. In this case we do not need any Lagrange multipliers (as there are no constraints) and thus optimize the objective function directly. As usual we exploit the fact that it is a necessary condition for a minimum that the partial derivatives vanish. This leads to $\forall k; 1 \leq k \leq c : \forall l; 1 \leq l \leq n :$

$$\frac{\partial}{\partial u_{kl}} J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = w u_{kl}^{w-1} d_{kl}^{2} - \nu_k w (1-u_{kl})^{w-1} \stackrel{!}{=} 0.$$

From these equations it follows $\forall i; 1 \leq i \leq c : \forall j; 1 \leq j \leq n :$

$$u_{ij} = \left(\frac{\nu_i}{d_{ij}^2 + \nu_i} \right)^{\frac{1}{w-1}} = \left(\frac{d_{ij}^2}{\nu_i} + 1 \right)^{\frac{1}{1-w}}$$

with the special case $w = 2$ yielding

$$u_{ij} = \frac{1}{\frac{d_{ij}^2}{\nu_i} + 1}.$$

From this special case we see two things: in the first place, the penalty weights ν_i, $1 \leq i \leq c$, turn out to be the squares of the distances from the cluster centers at which the degree of membership equals $\frac{1}{2}$. Therefore they can be interpreted as a kind of reference radius. This is also supported by the second observation, which consists in the insight that the above formula for the special case $w = 2$ may also be written as

$$u_{ij} = f_{\text{Cauchy}}(d(\vec{\mu}_i, \vec{x}_j; \nu_i \mathbf{\Sigma}_i); 2, 1).$$

Here ν_i is combined with the covariance matrix of the Mahalanobis distance, which has the same effect as scaling the (squared) distance with its reciprocal value. The general form of the update rule (i.e., with the exponent $\frac{1}{w-1}$) may also be accommodated by exploiting the membership transformation as it was described in Section 2.4 with $\alpha = \frac{1}{w-1}$ and $\beta = 0$. This shows that possibilistic fuzzy clustering needs no special treatment, but can be achieved by properly parameterizing the general cluster model.

Update of the Cluster Parameters

For the derivation of the update rule for the cluster parameters we consider the most general case of a cluster-specific Mahalanobis distance measure. That is, we consider $\forall i; 1 \leq i \leq c$: $\forall j; 1 \leq j \leq n$:

$$d_{ij} = d(\vec{x}_j, \vec{\mu}_i; \mathbf{\Sigma}_i) = \sqrt{(\vec{x}_j - \vec{\mu}_i)^\top \mathbf{\Sigma}_i^{-1} (\vec{x}_j - \vec{\mu}_i)},$$

where each $\mathbf{\Sigma}_i$, $1 \leq i \leq c$, is a symmetric and positive definite matrix. Note that in this very general case I do not even require these matrices to have determinant 1, as it is usually the case for so-called Gustafson–Kessel clustering [Gustafson and Kessel 1979]. Although this leads to minor technical complications (see below), it has the advantage of being very general and demonstrates that all special approaches result from the general case by simply introducing particular constraints.

With a general Mahalanobis distance the objective function reads

$$J(\mathbf{X}, \mathbf{C}, \mathbf{U}) = \sum_{i=1}^{c} \sum_{j=1}^{n} h(u_{ij}) \, (\vec{x}_j - \vec{\mu}_i)^\top \mathbf{\Sigma}^{-1} (\vec{x}_j - \vec{\mu}_i).$$

Note that this objective function is written with a general membership transformation function $h(u_{ij})$ instead of the special case u_{ij}^w. This is possible, because the membership degrees are fixed in this step and thus the transformation function does not have any influence on the result.

To obtain the update rule for the cluster parameters we exploit that a necessary condition for a minimum is that the partial derivatives w.r.t. the parameters vanish. For the cluster centers we thus get $\forall k; 1 \leq k \leq c$:

$$\begin{aligned}
\nabla_{\vec{\mu}_k} J(\mathbf{X}, \mathbf{C}, \mathbf{U}) &= \nabla_{\vec{\mu}_k} \sum_{i=1}^{c} \sum_{j=1}^{n} h(u_{kj}) \, (\vec{x}_j - \vec{\mu}_i)^\top \boldsymbol{\Sigma}_i^{-1} (\vec{x}_j - \vec{\mu}_i) \\
&= \sum_{j=1}^{n} h(u_{kj}) \, \nabla_{\vec{\mu}_k} (\vec{x}_j - \vec{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\vec{x}_j - \vec{\mu}_k) \\
&= -2\boldsymbol{\Sigma}_k^{-1} \sum_{j=1}^{n} h(u_{kj}) \, (\vec{x}_j - \vec{\mu}_k) \stackrel{!}{=} \vec{0}
\end{aligned}$$

Note that the minus sign results from the inner derivative $\nabla_{\vec{\mu}_k} (\vec{x}_j - \vec{\mu}_k)$ (also cf. Sections A.1 and A.2.2 in the appendix for details about computing the gradient of the vector-matrix expression). It follows $\forall i; 1 \leq i \leq c$:

$$\vec{\mu}_i^{(t+1)} = \frac{\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right) \vec{x}_j}{\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right)}.$$

Like the update rule for the membership degrees this expression is fairly intuitive. However, one has to concede that it would be even more intuitive without the membership transformation function $h(u_{ij})$. Then the u_{ij} could be interpreted as fractional weights of the data points, which add up to one over the clusters, so that each data point has unit total influence on the clusters. Unfortunately, such a more intuitive update rule cannot be obtained as a result of modifying the objective function.[6] Such a rule results, though, in the related approach of expectation maximization for mixture models, which is discussed in the next section. However, there it is derived in a completely different way.

[6] It is possible to get such an update rule formally by using $h(u_{ij}) = u_{ij}$ and changing the constraint to $\forall j; 1 \leq j \leq n : \sum_{i=1}^{c} u_{ij}^2 = 1$. However, this only shifts the standard fuzzifier $w = 2$ of $h(u_{ij}) = u_{ij}^2$ into the constraint, where it has basically the same effect (although the membership degrees are different now). Due to the changed constraint, the membership degrees again do not add up to one for each data point and thus cannot be interpreted as a distribution of a unit data point weight.

Next we have to find the update formula for the covariance matrices $\mathbf{\Sigma}_i$, $1 \leq i \leq c$. Here we face the complication that the objective function does not possess a minimum w.r.t. unconstrained covariance matrices. The reason is that a Mahalanobis distance is the smaller, the larger the determinant of the covariance matrix it is parameterized with. This is so, simply because the larger the determinant of the covariance matrix, the smaller the determinant of its inverse and thus the smaller the value of $(\vec{x}-\vec{\mu}_i)^\top \mathbf{\Sigma}_i^{-1}(\vec{x}-\vec{\mu}_i)$. Hence the minimum of the objective function would be obtained for covariance matrices with infinite determinant (if this were possible).

To handle this (actually merely technical and not fundamental) problem, we draw on insights about the properties of cluster prototypes derived in Section 2.3. In particular, we exploit that a covariance matrix can be split into size and shape parameters by writing it as

$$\mathbf{\Sigma} = \sigma^2 \mathbf{S},$$

where $\sigma = \sqrt[2m]{|\mathbf{\Sigma}|}$ (with m being the number of dimensions of the data space) and $\mathbf{S}$ is a symmetric and positive definite matrix satisfying $|\mathbf{S}| = 1$. At first sight such an approach may seem to be unreasonable, because we now have to handle the additional constraints $|\mathbf{S}_i| = 1$, $1 \leq i \leq c$. However, a careful analysis shows that in some special cases we need these constraints anyway in order to obtain a proper result, so we may just as well introduce them generally. We also have to cope with the technical problem that it is cumbersome to compute the derivative of $(\vec{x}_j - \vec{\mu}_i)^\top \mathbf{\Sigma}_i^{-1}(\vec{x}_j - \vec{\mu}_i)$ w.r.t. $\mathbf{S}_i$. Fortunately, however, this problem is removed without any effort by the convenient fact that taking the derivative w.r.t. the inverse $\mathbf{S}_i^{-1}$ leads to an update formula for $\mathbf{S}_i$. Therefore we can proceed in a fairly straightforward manner, treating the size and shape parameters separately.

We start by deriving an update formula for the shape parameters $\mathbf{S}_i$. To do so, we incorporate the constraints $|\mathbf{S}_i| = |\mathbf{S}_i^{-1}| = 1$, $i = 1, \ldots, c$, with the help of Lagrange multipliers λ_i. Thus we get the Lagrange function

$$\begin{aligned} \mathcal{L}(\mathbf{X}, \mathbf{C}, \mathbf{U}, \Lambda) &= \underbrace{\sum_{i=1}^{c} \sum_{j=1}^{n} h(u_{ij})(\vec{x}_j - \vec{\mu}_i)^\top \sigma_i^{-2} \mathbf{S}_i^{-1} (\vec{x}_j - \vec{\mu}_i)}_{= J(\mathbf{X}, \mathbf{C}, \mathbf{U})} \\ &+ \sum_{i=1}^{c} \lambda_i (1 - |\mathbf{S}_i^{-1}|), \end{aligned}$$

which is to be minimized. A necessary condition for a minimum is that the derivatives w.r.t. $\mathbf{S}_k^{-1}$, $1 \leq k \leq c$, all vanish. Thus we get (cf. the general

considerations in Section A.1 in the appendix) $\forall k; 1 \leq k \leq c$:

$$\begin{aligned}
&\nabla_{\mathbf{S}_k^{-1}} \mathcal{L}(\mathbf{X}, \mathbf{C}, \mathbf{U}, \Lambda) \\
&= \nabla_{\mathbf{S}_k^{-1}} \left(\sum_{i=1}^{c} \sum_{j=1}^{n} h(u_{ij}) \, (\vec{x}_j - \vec{\mu}_i)^\top \sigma_i^{-2} \mathbf{S}_i^{-1} (\vec{x}_j - \vec{\mu}_i) + \sum_{i=1}^{c} \lambda_i (1 - |\mathbf{S}_i^{-1}|) \right) \\
&= \sum_{j=1}^{n} h(u_{kj}) \, \sigma_i^{-2} \, (\vec{x}_j - \vec{\mu}_k)(\vec{x}_j - \vec{\mu}_k)^\top - \lambda_k \, |\mathbf{S}_k^{-1}| \mathbf{S}_k \quad \stackrel{!}{=} \quad \mathbf{0},
\end{aligned}$$

where $\mathbf{0}$ denotes an $m \times m$ null matrix (i.e., all its elements are zero). For the derivative $\nabla_{\mathbf{S}_k^{-1}} |\mathbf{S}_k^{-1}|$ the formula computed in Section A.1 in the appendix was used with $\mathbf{A} = \mathbf{S}_k^{-1}$. It follows $\forall i; 1 \leq i \leq c$:

$$\sum_{j=1}^{n} h(u_{ij}) \, \sigma_i^{-2} (\vec{x}_j - \vec{\mu}_i)(\vec{x}_j - \vec{\mu}_i)^\top = \lambda_i \, |\mathbf{S}_i^{-1}| \mathbf{S}_i = \lambda_i \mathbf{S}_i,$$

since we have to respect the constraint $|\mathbf{S}_i| = |\mathbf{S}_i^{-1}| = 1$. Hence we have

$$|\lambda_i \mathbf{S}_i| = \lambda_i^m |\mathbf{S}_i| = \lambda_i^m = \left| \sum_{j=1}^{n} h(u_{ij}) \, \sigma_i^{-2} (\vec{x}_j - \vec{\mu}_i)(\vec{x}_j - \vec{\mu}_i)^\top \right|,$$

where we exploited again the constraints $|\mathbf{S}_i| = |\mathbf{S}_i^{-1}| = 1$. Inserting the value for λ_i that results from this equation into the preceding one, we obtain

$$\mathbf{S}_i^{(t+1)} = \left| \widehat{\mathbf{S}}_i^{(t+1)} \right|^{-\frac{1}{m}} \widehat{\mathbf{S}}_i^{(t+1)}$$

where

$$\widehat{\mathbf{S}}_i^{(t+1)} = \sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right) \left(\sigma_i^{(t)}\right)^{-2} \left(\vec{x}_j - \vec{\mu}_i^{(t+1)}\right) \left(\vec{x}_j - \vec{\mu}_i^{(t+1)}\right)^\top.$$

Obviously the factor $|\widehat{\mathbf{S}}_i|^{-\frac{1}{m}}$ normalizes $\widehat{\mathbf{S}}_i$ to a unit determinant, as desired. If the sizes of the clusters (as they are described by the σ_i^2) are fixed, the above update formulae are all we need. They define a slightly generalized version of the algorithm by [Gustafson and Kessel 1979] (generalized to arbitrary, but fixed cluster sizes). Note that in the above formula one may also use $\vec{\mu}_i^{(t)}$ instead of $\vec{\mu}_i^{(t+1)}$, depending on which version is more convenient to implement (this depends on how the covariance matrix is computed—from the data points or from the difference vectors).

It is worth noting that the algorithm by [Gustafson and Kessel 1979] is actually defined in terms of so-called **fuzzy covariance matrices**[7]

$$\widehat{\mathbf{\Sigma}}_i^{(t+1)} = \frac{\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right) \left(\vec{x}_j - \vec{\mu}_i^{(t+1)}\right) \left(\vec{x}_j - \vec{\mu}_i^{(t+1)}\right)^\top}{\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right)}$$

instead of the matrices $\widehat{\mathbf{S}}_i$, $1 \leq i \leq c$, as defined above. (Note that the factor σ_i^{-2} is missing, because the algorithm by [Gustafson and Kessel 1979] fixes this factor to 1. It may, however, be introduced here without problems.) This leads to the same result for the $\mathbf{S}_i$, because the additional factors $\left(\sum_{j=1}^{n} h(u_{ij})\right)^{-1}$ have no influence due to the normalization to a unit determinant that is involved in computing the new size parameters (the new matrices $\mathbf{S}_i$). However, using $\widehat{\mathbf{\Sigma}}_i$ instead of $\widehat{\mathbf{S}}_i$ is clearly more intuitive, when compared to the well-known statistical estimator for a covariance matrix.

Furthermore, with a fuzzy covariance matrix as defined above, the most natural and intuitive way of adapting the cluster size would be to set

$$\mathbf{\Sigma}_i^{(t+1)} = \widehat{\mathbf{\Sigma}}_i^{(t+1)}, \qquad \text{or equivalently} \qquad \sigma_i^{(t+1)} = \sqrt[2m]{\left|\widehat{\mathbf{\Sigma}}_i^{(t+1)}\right|},$$

that is, to simply skip the normalization to a unit determinant. Such an approach is indeed feasible and leads to good results in practical tests. Its only drawback is that this update rule cannot be derived from the objective function due to the complications mentioned above.

If one desires to derive an update rule for the cluster sizes from the objective function, one has to introduce a constraint that prevents the clusters sizes from becoming infinite (see above). Such a constraint could be that the sum of the cluster sizes is fixed. For example, one may constrain the sum of the cluster sizes to the number of clusters, so that on average the clusters have unit size. Of course, if one does so, one has to take into account that the size of a cluster can be measured in different ways, as discussed in Section 2.3. Drawing on the considerations in that section, I measure the size of a cluster generally as σ_i^κ, where σ_i is defined as above, that is, as $\sigma_i = \sqrt[2m]{|\mathbf{\Sigma}_i|}$. Hence I introduce the size constraint

$$\sum_{i=1}^{c} \sigma_i^\kappa = c$$

[7] The original paper uses, of course, $h(u_{ij}) = u_{ij}^w$, but it is immediately clear that a generalization to an arbitrary membership transformation is possible.

and incorporate it into the objective function by a Lagrange multiplier λ. This approach—which is very similar to the approach suggested in [Keller 2002], but somewhat more principled—yields the Lagrange function

$$\mathcal{L}(\mathbf{X}, \mathbf{C}, \mathbf{U}, \Lambda) = \underbrace{\sum_{i=1}^{c} \sum_{j=1}^{n} h(u_{ij})(\vec{x}_j - \vec{\mu}_i)^\top \sigma_i^{-2} \mathbf{S}_i^{-1} (\vec{x}_j - \vec{\mu}_i)}_{=J(\mathbf{X},\mathbf{C},\mathbf{U})} + \lambda \left(\sum_{i=1}^{c} \sigma_i^\kappa - c \right).$$

Exploiting as usual that it is a necessary condition for a minimum that the partial derivatives vanish, we obtain $\forall k; 1 \leq k \leq c$:

$$\begin{aligned} &\frac{\partial}{\partial \sigma_k} \mathcal{L}(\mathbf{X}, \mathbf{C}, \mathbf{U}, \Lambda) \\ &= -2\sigma_k^{-3} \sum_{j=1}^{n} h(u_{kj}) \, (\vec{x}_j - \vec{\mu}_k)^\top \mathbf{S}_k^{-1} (\vec{x}_j - \vec{\mu}_k) + \lambda\kappa \, \sigma_k^{\kappa-1} \stackrel{!}{=} 0. \end{aligned}$$

As a consequence we have $\forall i; 1 \leq i \leq c$:

$$\sigma_i^{\kappa+2} = \frac{2}{\lambda\kappa} \sum_{j=1}^{n} h(u_{ij}) \, (\vec{x}_j - \vec{\mu}_i)^\top \mathbf{S}_i^{-1} (\vec{x}_j - \vec{\mu}_i).$$

In order to be able to exploit the constraint on the cluster sizes, which refers to σ_i^κ, $1 \leq i \leq c$, we transform this equation into

$$\sigma_i^\kappa = \left(\frac{2}{\lambda\kappa} \sum_{j=1}^{n} h(u_{ij}) \, (\vec{x}_j - \vec{\mu}_i)^\top \mathbf{S}_i^{-1} (\vec{x}_j - \vec{\mu}_i) \right)^{\frac{\kappa}{\kappa+2}}$$

and sum the result over all clusters. Thus we obtain

$$c = \sum_{i=1}^{c} \sigma_i^\kappa = \sum_{i=1}^{c} \left(\frac{2}{\lambda\kappa} \sum_{j=1}^{n} h(u_{ij}) \, (\vec{x}_j - \vec{\mu}_i)^\top \mathbf{S}_i^{-1} (\vec{x}_j - \vec{\mu}_i) \right)^{\frac{\kappa}{\kappa+2}},$$

which entails

$$\lambda = \frac{2}{\kappa} \left(\frac{1}{c} \sum_{i=1}^{c} \left(\sum_{j=1}^{n} h(u_{ij}) \, (\vec{x}_j - \vec{\mu}_i)^\top \mathbf{S}_i^{-1} (\vec{x}_j - \vec{\mu}_i) \right)^{\frac{\kappa}{\kappa+2}} \right)^{\frac{\kappa+2}{\kappa}}.$$

Inserting this expression for λ into the above formula for σ_i^κ yields

$$\sigma_i^{\kappa(t+1)} = \frac{c \cdot \left(\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right)\left(\vec{x}_j - \vec{\mu}_i^{(t+1)}\right)^\top \left(\mathbf{S}_i^{(t+1)}\right)^{-1}\left(\vec{x}_j - \vec{\mu}_i^{(t+1)}\right)\right)^{\frac{\kappa}{\kappa+2}}}{\sum_{k=1}^{c}\left(\sum_{j=1}^{n} h\left(u_{kj}^{(t+1)}\right)\left(\vec{x}_j - \vec{\mu}_k^{(t+1)}\right)^\top \left(\mathbf{S}_k^{(t+1)}\right)^{-1}\left(\vec{x}_j - \vec{\mu}_k^{(t+1)}\right)\right)^{\frac{\kappa}{\kappa+2}}}.$$

On closer inspection this formula is actually not too unintuitive. If there were no exponents, the new sizes of the clusters would be computed from the weighted sum of squared distances as they result from a covariance matrix with unit determinant, by relating it to the total weighted sum of squared distances for all clusters. This would be a very natural way of updating the sizes. The exponents modify this a little, so that the results deviate somewhat from the most natural case. Unfortunately, such an effect is often met in fuzzy clustering. We already observed it when we derived the update rules for the cluster centers, which contain the fuzzifier as an unnatural modification. Here, however, the effect is that a tendency towards equal sizes is introduced (since the exponents are less than 1), which may be desirable in some cases (cf. also Section 6.1.3).

This completes the derivation of the update rules for the cluster parameters. Nevertheless, it is often recommendable to introduce further constraints and modifications into the update process, which can enhance the stability and usefulness of fuzzy clustering. However, since these constraints and modifications are more generally applicable (for example, they may also be used with gradient descent), they are considered in the next chapter.

Note that for the derivation of the update rules for the cluster parameters it was not necessary to consider possibilistic fuzzy clustering as a special case, while it was necessary to treat it as a special case for the derivation of the update rules for the membership degrees. The reason is that the additional terms in the objective function of possibilistic fuzzy clustering refer only to the membership degrees and not to any cluster parameters. Therefore they are eliminated by taking derivatives w.r.t. cluster parameters.

What needs explicit consideration, though, is the generalized form of the objective function that uses membership functions instead of distances, i.e.

$$J(\mathbf{X}, \mathbf{C}, \mathbf{U}) = \sum_{i=1}^{c}\sum_{j=1}^{n} \frac{u_{ij}^w}{u_i^\bullet(\vec{x}_j)}.$$

In this case the membership functions $u_i^\bullet$, $1 \leq i \leq c$, may make it impossible to find the optimum w.r.t. the cluster parameters (for fixed membership degrees) by simply setting the partial derivatives equal to zero.

The most convenient way to solve this problem is to abandon the strict requirement that the update formulae have to be derived from the objective function. The general idea is to keep the alternating scheme, but to specify update rules for the two steps that are inspired by the objective function and are analogous to the results for cases with just a distance measure, but may not be derivable in a strict way. This approach has been suggested by [Runkler and Bezdek 1999] under the name of **alternating cluster estimation** (ACE). Here I use the same idea by keeping the update rules for the cluster parameters even though the membership to a cluster may be computed with the generalized Cauchy or Gaussian function. Experiments show that this approach is feasible and leads to good results.

As a final remark I would like to mention the reformulation approach to the update rules that was suggested in [Hathaway and Bezdek 1995]. The basic idea is to insert the update rule for the membership degrees into the objective function in order to obtain a one step update scheme. Although this is surely an interesting approach from a mathematical point of view, I do not discuss it in detail here, because its practical relevance is limited. In addition, it works smoothly only for the standard membership transformation $h(u_{ij}) = u_{ij}^w$, while one encounters technical problems for the more general case and in particular for $h(u_{ij}) = \frac{1-\beta}{1+\beta} u_{ij}^2 + \frac{2\beta}{1+\beta} u_{ij}$.

5.2.4 Expectation Maximization

The expectation maximization (EM) algorithm is a standard method in statistics for computing maximum likelihood estimates of distribution parameters in the presence of missing values or hidden variables (also called *latent* variables), that is, variables the values of which cannot be observed. In the presence of missing values or hidden variables it is often impossible to maximize the likelihood function directly, because it is (partially) unknown which data points may be used for estimating which parameters.

The general idea underlying the expectation maximization algorithm is to describe a value that is missing by a random variable. The domain of this random variable is the set of values that could be the actual, but unknown value. As a consequence, the likelihood of the data set becomes a random variable, because it is a function of the random variables describing the missing values. (We may also view the situation like this: we do not have a fixed data set, but a random variable that has different data sets, though from a limited range, as possible values.) This, of course, makes it impossible to maximize the likelihood directly, as it does not have a unique value anymore. However, since it is a random variable, we can compute its

expected value and choose the parameters in such a way that this expected value is maximized. Hence the name *expectation maximization* for this approach: the parameters are estimated such that the expected value of the likelihood function is maximized.

In the following I consider the expectation maximization approach in two flavors. In the first place I study the *standard approach for mixture models*, in which I focus in particular on the most common case of Gaussian mixture models [Everitt and Hand 1981, Bilmes 1997]. The second approach is what has become known as the *fuzzy maximum likelihood estimation algorithm* [Gath and Geva 1989]. This algorithms draws on ideas from both expectation maximization for mixture models and fuzzy clustering as it was discussed in the preceding section.

Expectation Maximization for Mixture Models

We assume that the given data was generated by sampling from a mixture of c probability density functions, each of which is represented by a cluster and its parameters (cf. Section 3.3). That is, we assume that the probability density function of the data generation process can be described as

$$f_{\vec{X}}(\vec{x}; \mathbf{C}) = \sum_{y=1}^{c} f_{\vec{X},Y}(\vec{x}, y; \mathbf{C}) = \sum_{y=1}^{c} p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}|y; \mathbf{C}).$$

Here $\mathbf{C}$ is, as usual, the set of cluster parameters. $\vec{X}$ is a random vector that has the data space as its domain (i.e. $\mathrm{dom}(\vec{X}) = \mathbb{R}^m$). Y is a random variable that has cluster indices as possible values (i.e. $\mathrm{dom}(Y) = \{1, \ldots, c\}$). $p_Y(y; \mathbf{C})$ is the probability that a data point belongs to (is generated by) the y-th component of the mixture (the y-th cluster) and $f_{\vec{X}|Y}(\vec{x}|y; \mathbf{C})$ is the conditional probability density function of a data point given the mixture component (as specified by the cluster index y).

The task is to estimate the cluster parameters in such a way that the likelihood of a given data set, described by the **likelihood function**

$$L(\mathbf{X}; \mathbf{C}) = \prod_{j=1}^{n} f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}) = \prod_{j=1}^{n} \sum_{y=1}^{c} p_Y(y; \mathbf{C}) \cdot f_{\vec{X}_j|Y}(\vec{x}_j|y; \mathbf{C})$$

(cf. page 62 in Section 3.3) is maximized. Alternatively, one may choose to maximize the so-called **log-likelihood**, because this can ease the technical task[8], in particular, if a Gaussian mixture model is used.

[8] Note that taking the logarithm of the likelihood function changes the *value*, but not the *location* of the maximum in the parameter space, which is all we are interested in.

Unfortunately, the likelihood function is difficult to optimize, even if one takes its natural logarithm, because

$$\ln L(\mathbf{X}; \mathbf{C}) = \sum_{j=1}^{n} \ln \sum_{y=1}^{c} p_Y(y; \mathbf{C}) \cdot f_{\vec{X}_j|Y}(\vec{x}_j|y; \mathbf{C})$$

contains the product of the natural logarithms of complex sums. (Note that this was no problem in a gradient descent approach, because we only desired the derivative of the log-likelihood function. Here, however, we want to compute the maximum directly, in a non-iterative fashion.)

To handle this problem, we assume that there are "hidden" variables Y_j, $1 \leq j \leq n$, which state for each data point $\vec{x}_j$ the cluster that generated it (i.e. $\mathrm{dom}(Y_i) = \{1, \ldots, c\}$). With such variables, the (inner) sums reduce to one term and the task of computing the maximum becomes mathematically feasible. Formally, we now maximize the likelihood of the "completed" data set $(\mathbf{X}, \vec{y})$, where $\vec{y} = (y_1, \ldots, y_n)$ combines the values of the variables Y_j, $1 \leq j \leq n$. That is, we consider the extended likelihood function

$$L(\mathbf{X}, \vec{y}; \mathbf{C}) = \prod_{j=1}^{n} f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) = \prod_{j=1}^{n} p_{Y_j}(y_j; \mathbf{C}) \cdot f_{\vec{X}_j|Y_j}(\vec{x}_j|y_j; \mathbf{C})$$

(or, equivalently, the natural logarithm of this function). The problem with this approach is, of course, that the Y_j are hidden, so we do not know their values. However, we need to know these values in order to compute the factors $p_{Y_j}(y_j; \mathbf{C})$. Otherwise we are unable to maximize the likelihood function, because it is incompletely specified.

To find a solution nevertheless, we see the Y_j as random variables (i.e., the values y_j are not fixed), as already indicated at the beginning of this section. Then we consider a probability distribution over the possible values, that is over the set $\mathrm{dom}(Y_j) = \{1, \ldots, c\}$. As a consequence the value of the likelihood $L(\mathbf{X}, \vec{y}; \mathbf{C})$ becomes a random variable, even for a fixed data set $\mathbf{X}$ and fixed cluster parameters $\mathbf{C}$, simply because the third parameter $\vec{y}$ of the likelihood function is a random vector. Since $L(\mathbf{X}, \vec{y}; \mathbf{C})$ or $\ln L(\mathbf{X}, \vec{y}; \mathbf{C})$ are random variables, they have an expected value and we may now try to *maximize this expected value* (hence the name *expectation maximization* for this approach, as already pointed out above).

Formally, this means to try to find the cluster parameters as

$$\hat{\mathbf{C}} = \operatorname*{argmax}_{\mathbf{C}} E([\ln]L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}),$$

where the hat symbol (ˆ) is standard statistical notation for an estimate and [ln] indicates the optional use of the log-likelihood. That is, we compute the parameters by maximizing the expected likelihood

$$E(L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}) = \sum_{\vec{y} \in \{1,\ldots,c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y}\,|\mathbf{X}; \mathbf{C}) \cdot \prod_{j=1}^{n} f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C})$$

or, alternatively, by maximizing the expected log-likelihood

$$E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}) = \sum_{\vec{y} \in \{1,\ldots,c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y}\,|\mathbf{X}; \mathbf{C}) \cdot \sum_{j=1}^{n} \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}).$$

Here $\mathcal{X}$ is a random variable that has data sets as possible values. $\vec{Y}$ is a random variable that has the possible assignments of the data points to the clusters as possible values, which are described by a vector $\vec{y} \in \{1, \ldots, c\}^n$. The expected value is computed in the standard way: the different values of the likelihood function of the completed data set, which are distinguished by the different possible vectors $\vec{y}$ of cluster indices, are summed weighted with the probability of their occurrence.

Unfortunately, these functionals are still difficult to optimize directly. However, they can nicely be turned into iterative schemes. This is done by fixing $\mathbf{C}$ in some terms. Then the maximization operation (now limited to terms in which $\mathbf{C}$ is not fixed) is used iteratively to compute increasingly better approximations of the maximum, until a convergence point is reached.

This principle is very similar to the idea underlying **Heron's algorithm** for computing the square root of a given real number, which is reviewed in Section A.9 in the appendix. In Heron's algorithm the defining equation for the square root is rewritten, so that the quantity to compute appears several times in the resulting equation. The resulting equation is then written in such a way that the desired quantity (the square root) appears isolated on the left hand side. The algorithm itself works by initializing the desired quantity (the square root) to an arbitrary value and inserting it into the right hand side. Evaluating the right hand side provides us with a new approximation of the desired quantity, which is inserted again into the right hand side and so on, until a chosen precision is reached.

Analogously, the iterative scheme for expectation maximization works as follows: we choose an initial set $\mathbf{C}^{(0)}$ of cluster parameters (see Chapter 4 for initialization methods). With this initial parameter set we can compute the posterior probability of the i-th cluster given a data point. This enables

us to carry out the maximization, since the first factors in the outer sum are now fixed values, which can even be simplified considerably (see below). The result is a new approximation $\mathbf{C}^{(1)}$ of the cluster parameters, which is used again to compute the posterior probabilities of the clusters given a data point and so on, until a convergence point is reached.

The posterior probabilities, which I abbreviate by $u_{ij}^{(t+1)}$ in analogy to the membership degrees of fuzzy clustering, are computed as [Bilmes 1997]

$$u_{ij}^{(t+1)} = p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}^{(t)}) = \frac{f_{\vec{X}_j,Y_j}(\vec{x}_j, i; \mathbf{C}^{(t)})}{f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}^{(t)})} = \frac{f_{\vec{X}_j|Y_j}(\vec{x}_j|i; \mathbf{C}^{(t)}) \cdot p_{Y_j}(i; \mathbf{C}^{(t)})}{\sum_{k=1}^{c} f_{\vec{X}_j|Y_j}(\vec{x}_j|k; \mathbf{C}^{(t)}) \cdot p_{Y_j}(k; \mathbf{C}^{(t)})},$$

that is, as the relative probability densities of the different clusters at the location of the data points $\vec{x}_j$. Note that these $u_{ij}^{(t+1)}$ may also be written in terms of the cluster membership functions (parameterized with $\mathbf{C}^{(t)}$) as

$$u_{ij}^{(t+1)} = u_i^{(\mathrm{sum1})\,(t)}(\vec{x}_j) = \frac{u_i^{\bullet(t)}(\vec{x}_j)}{\sum_{k=1}^{c} u_k^{\bullet(t)}(\vec{x}_j)}.$$

This provides another justification for using $u_{ij}^{(t+1)}$ as an abbreviation for the posterior probabilities $p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}^{(t)})$.

The computation of the posterior probabilities is called the **expectation step**, which is followed by the **maximization step** [Bilmes 1997]:

$$\begin{aligned}
\mathbf{C}^{(t+1)} &= \operatorname*{argmax}_{\mathbf{C}} E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}^{(t)}) \\
&= \operatorname*{argmax}_{\mathbf{C}} \sum_{\vec{y} \in \{1,\ldots,c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y}\,|\mathbf{X}; \mathbf{C}^{(t)}) \sum_{j=1}^{n} \ln f_{\vec{X}_j,Y_j}(\vec{x}_j, y_j; \mathbf{C}) \\
&= \operatorname*{argmax}_{\mathbf{C}} \sum_{\vec{y} \in \{1,\ldots,c\}^n} \left(\prod_{l=1}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l; \mathbf{C}^{(t)}) \right) \sum_{j=1}^{n} \ln f_{\vec{X}_j,Y_j}(\vec{x}_j, y_j; \mathbf{C}) \\
&= \operatorname*{argmax}_{\mathbf{C}} \sum_{i=1}^{c} \sum_{j=1}^{n} p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}^{(t)}) \cdot \ln f_{\vec{X}_j,Y_j}(\vec{x}_j, i; \mathbf{C}).
\end{aligned}$$

The last step of this transformation, which replaces the complex sum over all possible vectors of cluster indices by a simple sum over the clusters, is

justified as follows [Bilmes 1997] ($\delta_{i,y}$ is the Kronecker symbol, cf. page 60):

$$
\begin{aligned}
&\sum_{\vec{y}\in\{1,\ldots,c\}^n} \left(\prod_{l=1}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}^{(t)})\right) \sum_{j=1}^{n} \ln f_{\vec{X}_j,Y_j}(\vec{x}_j,y_j;\mathbf{C}) \\
&= \sum_{y_1=1}^{c}\cdots\sum_{y_n=1}^{c}\left(\prod_{l=1}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}^{(t)})\right)\sum_{j=1}^{n}\sum_{i=1}^{c}\delta_{i,y_j}\ln f_{\vec{X}_j,Y_j}(\vec{x}_j,i;\mathbf{C}) \\
&= \sum_{i=1}^{c}\sum_{j=1}^{n}\ln f_{\vec{X}_j,Y_j}(\vec{x}_j,i;\mathbf{C})\sum_{y_1=1}^{c}\cdots\sum_{y_n=1}^{c}\delta_{i,y_j}\prod_{l=1}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}^{(t)}) \\
&= \sum_{i=1}^{c}\sum_{j=1}^{n}\ln f_{\vec{X}_j,Y_j}(\vec{x}_j,i;\mathbf{C}) \\
&\quad \sum_{y_1=1}^{c}\cdots\sum_{y_n=1}^{c}\delta_{i,y_j}\cdot p_{Y_j|\vec{X}_j}(y_j|\vec{x}_j;\mathbf{C}^{(t)})\prod_{l=1;l\neq j}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}^{(t)}) \\
&= \sum_{i=1}^{c}\sum_{j=1}^{n}\ln f_{\vec{X}_j,Y_j}(\vec{x}_j,i;\mathbf{C})\sum_{y_j=1}^{c}\delta_{i,y_j}\cdot p_{Y_j|\vec{X}_j}(y_j|\vec{x}_j;\mathbf{C}^{(t)}) \\
&\quad \sum_{y_1=1}^{c}\cdots\sum_{y_{j-1}=1}^{c}\sum_{y_{j+1}=1}^{c}\cdots\sum_{y_n=1}^{c}\prod_{l=1;l\neq j}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}^{(t)}) \\
&= \sum_{i=1}^{c}\sum_{j=1}^{n}\ln f_{\vec{X}_j,Y_j}(\vec{x}_j,i;\mathbf{C})\cdot p_{Y_j|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}^{(t)}) \\
&\quad \sum_{y_1=1}^{c}\cdots\sum_{y_{j-1}=1}^{c}\sum_{y_{j+1}=1}^{c}\cdots\sum_{y_n=1}^{c}\prod_{l=1;l\neq j}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}^{(t)}) \\
&= \sum_{i=1}^{c}\sum_{j=1}^{n} p_{Y_j|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}^{(t)})\cdot\ln f_{\vec{X}_j,Y_j}(\vec{x}_j,i;\mathbf{C}) \\
&\quad \prod_{l=1;l\neq j}^{n}\underbrace{\sum_{y_l=1}^{c} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}^{(t)})}_{=1} \\
&= \sum_{i=1}^{c}\sum_{j=1}^{n} p_{Y_j|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}^{(t)})\cdot\ln f_{\vec{X}_j,Y_j}(\vec{x}_j,i;\mathbf{C}).
\end{aligned}
$$

Intuitively, the outlined procedure can be interpreted as follows: In the **expectation step**, with the posterior probabilities of the clusters given a data point, we compute case weights of a "completed" data set. That is,

- we split each data point $\vec{x}_j$ into c data points $(\vec{x}_j, i)$, $1 \leq i \leq c$;
- we distribute the unit weight of the data point $\vec{x}_j$ according to the above probabilities, i.e., we assign to $(\vec{x}_j, i)$, $1 \leq i \leq c$, $1 \leq j \leq n$, the case weight $u_{ij}^{(t+1)} = p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k^{(t)})$.

This interpretation of the posterior probabilities as case weights is supported by the following consideration: suppose we sample a data set S from a cluster model $\mathbf{C}$, recording for each data point also the cluster i, so that the sample points have the form $(\vec{x}_j, i)$, $1 \leq i \leq c$, $1 \leq j \leq n$. In addition, let $u_{ij} \in \mathbb{N}_0$ state how often the corresponding data point $(\vec{x}_j, i)$ occurs in the sample S. (In principle, it may be $u_{ij} = 0$, namely if the data point does not occur in the sample, although this is only a theoretical possibility.) Then the probability of the sample S can be written as

$$P(S) = \prod_{i=1}^{c} \prod_{j=1}^{n} \Big(f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C}) \Big)^{u_{ij}}.$$

(Compare the considerations in Section 3.3, where I used exponents u_{ij} to write the probability of a data set in a very similar way.)

On the other hand, the expected log-likelihood can be written as

$$\begin{aligned} E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}^{(t)}) &= \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{(t)} \cdot \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C}) \\ &= \sum_{i=1}^{c} \sum_{j=1}^{n} \ln \Big(f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C}) \Big)^{u_{ij}^{(t)}} \\ &= \ln \prod_{i=1}^{c} \prod_{j=1}^{n} \Big(f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C}) \Big)^{u_{ij}^{(t)}}. \end{aligned}$$

Hence, if the exponents $u_{ij}^{(t)}$ were integer values, the expected log-likelihood would be the natural logarithm of the probability of a data set, in which a data point $(\vec{x}_j, i)$ occurred $u_{ij}^{(t)}$ times, $1 \leq i \leq c$, $1 \leq j \leq n$. All we have to do now is to drop the requirement for integer occurrence numbers and allow fractional occurrences. These fractional occurrences (or case weights) are given by the posterior probabilities.

After the data set has been "completed" by computing case weights for the extended data points, we estimate the cluster parameters in the **maximization step**. This estimation is trivial now, because we have a completely specified data set and thus can apply the standard statistical estimation formulae. Intuitively, we simple split the "completed" data set according to the value of i in the data points $(\vec{x}_j, i)$ into c subsets. From each of the subsets we have to estimate the parameters of one clusters. However, one cluster is described by a single density function—not by a mixture of density functions—and thus all mathematical problems that we encountered originally for the (log-)likelihood maximization vanished.

It can be shown that each iteration of the expectation maximization algorithm, consisting of one expectation step and one maximization step, increases the likelihood of the data. Even more, it can be shown that the algorithm converges to a local maximum of the likelihood function. In other words, expectation maximization is a safe way to maximize the likelihood function [Dempster *et al.* 1977, Wu 1983, Cheeseman and Stutz 1996].

In the above derivation the form of the conditional probability density function given the cluster was left unspecified. The most common case, however, is a normal distribution, i.e., a Gaussian mixture model (cf. page 62 in Section 3.3). In this case the maximization is particularly simple for the expected log-likelihood, because then we have for the maximization step (using again the abbreviation $u_{ij}^{(t)} = p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}^{(t)})$ introduced above)

$$
\begin{aligned}
\mathbf{C}^{(t+1)} &= \operatorname*{argmax}_{\mathbf{C}} E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}^{(t)}) \\
&= \operatorname*{argmax}_{\mathbf{C}} \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{(t+1)} \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C}) \\
&= \operatorname*{argmax}_{\mathbf{C}} \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{(t+1)} \\
&\qquad \ln \left(\frac{\varrho_i}{\sqrt{(2\pi)^m |\mathbf{\Sigma}_i|}} \exp\left(-\frac{1}{2} (\vec{x}_j - \vec{\mu}_i)^\top \mathbf{\Sigma}_i^{-1} (\vec{x}_j - \vec{\mu}_i) \right) \right) \\
&= \operatorname*{argmax}_{\mathbf{C}} \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{(t+1)} \frac{1}{2} \big(2 \ln \varrho_i - m \ln(2\pi) - \ln |\mathbf{\Sigma}_i| \\
&\qquad\qquad - (\vec{x}_j - \vec{\mu}_i)^\top \mathbf{\Sigma}_i^{-1} (\vec{x}_j - \vec{\mu}_i) \big).
\end{aligned}
$$

From this formula we can derive the update rules for the cluster parameters by simply taking the partial derivatives of the double sum and setting them

equal to zero (necessary condition for a maximum). That is, we compute

$$\begin{aligned}
&\nabla_{\vec{\mu}_k} E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}^{(t)}) \\
&= \nabla_{\vec{\mu}_k} \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{(t+1)} \frac{1}{2} \big(2 \ln \varrho_i - m \ln(2\pi) - \ln |\mathbf{\Sigma}_i| \\
&\qquad\qquad - (\vec{x}_j - \vec{\mu}_i)^\top \mathbf{\Sigma}_i^{-1} (\vec{x}_j - \vec{\mu}_i)\big) \\
&= -\frac{1}{2} \sum_{j=1}^{n} u_{kj}^{(t+1)} \nabla_{\vec{\mu}_k} (\vec{x}_j - \vec{\mu}_k)^\top \mathbf{\Sigma}_k^{-1} (\vec{x}_j - \vec{\mu}_k) \\
&= \mathbf{\Sigma}_k^{-1} \sum_{j=1}^{n} u_{kj}^{(t+1)} (\vec{x}_j - \vec{\mu}_k) \stackrel{!}{=} \vec{0}.
\end{aligned}$$

This leads to

$$\vec{\mu}_i^{(t+1)} = \frac{\sum_{j=1}^{n} u_{ij}^{(t+1)} \vec{x}_j}{\sum_{j=1}^{n} u_{ij}^{(t+1)}},$$

which is directly analogous to the standard maximum likelihood estimate for the expected value vector of a multivariate normal distribution. Note that it differs from the update rule for the cluster centers derived in fuzzy clustering by the missing transformation function $h(u_{ij})$ for the membership degrees/posterior probabilities. As a consequence it is more intuitive.

For the covariance matrices we obtain, by taking derivatives w.r.t. their inverses as we already did several times before to simplify the computations:

$$\begin{aligned}
&\nabla_{\mathbf{\Sigma}_k^{-1}} E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}^{(t)}) \\
&= \nabla_{\mathbf{\Sigma}_k^{-1}} \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{(t+1)} \frac{1}{2} \big(2 \ln \varrho_i - m \ln(2\pi) - \ln |\mathbf{\Sigma}_i| \\
&\qquad\qquad - (\vec{x}_j - \vec{\mu}_i)^\top \mathbf{\Sigma}_i^{-1} (\vec{x}_j - \vec{\mu}_i)\big) \\
&= \sum_{j=1}^{n} u_{ij}^{(t+1)} \frac{1}{2} \nabla_{\mathbf{\Sigma}_k^{-1}} \left(\ln |\mathbf{\Sigma}_k^{-1}| - (\vec{x}_j - \vec{\mu}_k)^\top \mathbf{\Sigma}_k^{-1} (\vec{x}_j - \vec{\mu}_k)\right) \\
&= \sum_{j=1}^{n} u_{kj}^{(t+1)} \frac{1}{2} \left(\mathbf{\Sigma}_k - (\vec{x}_j - \vec{\mu}_k)(\vec{x}_j - \vec{\mu}_k)^\top\right) \stackrel{!}{=} \mathbf{0}.
\end{aligned}$$

In this derivation it was exploited that $\nabla_{\mathbf{\Sigma}_k^{-1}} \ln |\mathbf{\Sigma}_k^{-1}| = |\mathbf{\Sigma}_k| |\mathbf{\Sigma}_k^{-1}| \, \mathbf{\Sigma}_k = \mathbf{\Sigma}_k$ (cf. Section A.1 in the appendix for the derivative of a determinant).

It follows as the update rule for the covariance matrices

$$\mathbf{\Sigma}_i^{(t+1)} = \frac{\sum_{j=1}^{n} u_{ij}^{(t+1)} \left(\vec{x}_j - \vec{\mu}_i^{(t+1)}\right)\left(\vec{x}_j - \vec{\mu}_i^{(t+1)}\right)^\top}{\sum_{j=1}^{n} u_{ij}^{(t+1)}}.$$

Again this result is directly analogous to the standard maximum likelihood estimate for a covariance matrix. Note that it differs, like the update rule for the expected value vectors, from the update rule derived in fuzzy clustering by the missing transformation function $h(u_{ij})$ for the membership degrees/posterior probabilities. As a consequence it is again more intuitive. Note also that for implementations it may be more convenient to use $\vec{\mu}_i^{(t)}$ instead of $\vec{\mu}_i^{(t+1)}$, depending on how the covariance matrix is computed (directly from the data points or from the difference vectors).

In a final step we have to consider the update rule for the prior probabilities ϱ_i, $1 \leq i \leq c$, of the different clusters, as the last cluster parameter. For these prior probability we have to take into account that they have to sum to 1 over the clusters. That is, we have to respect the constraint

$$\sum_{i=1}^{c} \varrho_i = 1.$$

Relying on our standard practice, we incorporate this constraint with the help of a Lagrange multiplier λ and obtain the Lagrange function

$$\mathcal{L}(\mathbf{X}, \vec{y}; \mathbf{C}, \lambda) = E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}^{(t)}) + \lambda\left(1 - \sum_{i=1}^{c} \varrho_i\right),$$

which is to be maximized. Exploiting as usual that it is a necessary condition for a maximum that the partial derivatives vanish, we obtain

$$\begin{aligned}
\frac{\partial}{\partial \varrho_k}\mathcal{L}(\mathbf{X}, \vec{y}; \mathbf{C}, \lambda) &= \frac{\partial}{\partial \varrho_k}\left(\sum_{i=1}^{c}\sum_{j=1}^{n} u_{ij}^{(t+1)} \frac{1}{2}\big(2\ln \varrho_i - m\ln(2\pi) - \ln|\mathbf{\Sigma}_i| \right. \\
&\qquad \left. - (\vec{x}_j - \vec{\mu}_i)^\top \mathbf{\Sigma}_i^{-1}(\vec{x}_j - \vec{\mu}_i)\big) + \lambda\left(1 - \sum_{i=1}^{c} \varrho_i\right)\right) \\
&= \sum_{j=1}^{n} u_{kj}^{(t+1)} \frac{\partial}{\partial \varrho_k} \ln \varrho_k + \frac{\partial}{\partial \varrho_k}\lambda\left(1 - \sum_{i=1}^{c} \varrho_i\right) \\
&= \sum_{j=1}^{n} u_{kj}^{(t+1)} \frac{1}{\varrho_k} - \lambda \stackrel{!}{=} 0.
\end{aligned}$$

This leads to $\forall i; 1 \leq i \leq c$:

$$\varrho_i = \frac{1}{\lambda} \sum_{j=1}^{n} u_{ij}^{(t+1)}.$$

In order to exploit the constraint, we sum these equations over the clusters,

$$1 = \sum_{i=1}^{c} \varrho_i = \frac{1}{\lambda} \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{(t+1)}.$$

From this we can compute λ as

$$\lambda = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{(t+1)}.$$

Inserting this equation into the expression for the prior probabilities yields

$$\varrho_i^{(t+1)} = \frac{\sum_{j=1}^{n} u_{ij}^{(t+1)}}{\sum_{k=1}^{c} \sum_{j=1}^{n} u_{kj}^{(t+1)}} = \frac{\sum_{j=1}^{n} u_{ij}^{(t+1)}}{\sum_{j=1}^{n} \underbrace{\sum_{k=1}^{c} u_{kj}^{(t+1)}}_{=1}} = \frac{1}{n} \sum_{j=1}^{n} u_{ij}^{(t+1)}.$$

Like the other update rules this result is very intuitive and directly analogous to standard maximum likelihood estimates for corresponding situations (like, for example, the estimate of a probability in a multinomial distribution, which is the relative frequency of the corresponding value). Note that it was exploited here that the $u_{ij}^{(t+1)}$ describe, for each j, $1 \leq j \leq n$, a probability distribution over the clusters. Actually, we defined (see above)

$$u_{ij}^{(t+1)} = p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}^{(t)}).$$

Consequently their sum over the clusters must be 1 for any value of j.

With these update rules, together with the formula for the computation of the posterior probabilities $u_{ij}^{(t+1)}$ as it was stated at the beginning, we have the full set of update formulae for a standard Gaussian mixture model. Of course, it is also possible to derive analogous formulae for a mixture model that uses the generalized Gaussian radial function or even the generalized Cauchy radial function, provided they can be normalized to unit integral (cf. Section A.2.1 in the appendix). However, since these derivations follow the same lines, I do no consider them explicitely.

Fuzzy Maximum Likelihood Estimation

The fuzzy maximum likelihood estimation algorithm [Gath and Geva 1989], although it uses an objective function that is based on a sum of (squared) distances, is more closely related to the expectation maximization algorithm than to the fuzzy clustering approaches discussed in the preceding section. It uses a (squared) distance measure that is inversely proportional to the probability that the datum is a member of the cluster, based on a normal distribution assumption. That is, the (squared) distance is defined as

$$d_{ij}^2 = \left(\frac{\varrho_i}{\sqrt{(2\pi)^m |\boldsymbol{\Sigma}_i|}} \exp\left(-\frac{1}{2} (\vec{x}_j - \vec{\mu}_i)^\top \boldsymbol{\Sigma}_i^{-1} (\vec{x}_j - \vec{\mu}_i) \right) \right)^{-1}.$$

Therefore the objective function is

$$J(\mathbf{X}, \mathbf{C}, \mathbf{U}) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^w \frac{\sqrt{(2\pi)^m |\boldsymbol{\Sigma}_i|}}{\varrho_i} \exp\left(\frac{1}{2} (\vec{x}_j - \vec{\mu}_i)^\top \boldsymbol{\Sigma}_i^{-1} (\vec{x}_j - \vec{\mu}_i) \right).$$

As we saw in Section 5.2.3, the update rule for the membership degrees is, independent of the distance measure,

$$u_{ij}^{(t+1)} = \frac{\left(d_{ij}^{(t)} \right)^{\frac{2}{1-w}}}{\sum_{k=1}^{c} \left(d_{kj}^{(t)} \right)^{\frac{2}{1-w}}}.$$

Thus we have for the special case $w = 2$

$$\begin{aligned} u_{ij}^{(t+1)} &= \frac{\frac{\varrho_i^{(t)}}{\sqrt{(2\pi)^m |\boldsymbol{\Sigma}_i^{(t)}|}} \exp\left(-\frac{1}{2} \left(\vec{x}_j - \vec{\mu}_i^{(t)} \right)^\top \left(\boldsymbol{\Sigma}_i^{(t)} \right)^{-1} \left(\vec{x}_j - \vec{\mu}_i^{(t)} \right) \right)}{\sum_{k=1}^{c} \frac{\varrho_k^{(t)}}{\sqrt{(2\pi)^m |\boldsymbol{\Sigma}_k^{(t)}|}} \exp\left(-\frac{1}{2} \left(\vec{x}_j - \vec{\mu}_k^{(t)} \right)^\top \left(\boldsymbol{\Sigma}_k^{(t)} \right)^{-1} \left(\vec{x}_j - \vec{\mu}_k^{(t)} \right) \right)} \\ &= p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}^{(t)}). \end{aligned}$$

That is, we obtain a formula that is identical to the expectation step of the expectation maximization algorithm for a standard Gaussian mixture model as it was studied above.

On the other hand, deriving update rules for the cluster parameters from the above objective function is much more difficult. Therefore [Gath and Geva 1989] rely on analogical reasoning and draw on the update rules as they result from fuzzy clustering with normal distances and from the expectation maximization algorithm for a Gaussian mixture model.

The basic idea is that the parameter update rules in the expectation maximization algorithm closely resemble those for fuzzy clustering based on a Mahalanobis distance (Gustafson–Kessel algorithm). The main difference between these update rules consists in the fact that in fuzzy clustering the data point weights are computed from the membership degrees through a membership transformation function. In expectation maximization, however, the posterior probabilities, which are analogous to the membership degrees in fuzzy clustering, are used directly as case weights. Hence the idea suggests itself to apply such a membership transformation function here too, in order to arrive at a "fuzzified" maximum likelihood estimation algorithm [Gath and Geva 1989, Döring *et al.* 2004].

For the cluster centers we thus get the same update rule as in fuzzy clustering based on a Mahalanobis distance, namely $\forall i; 1 \leq i \leq c$:

$$\vec{\mu}_i^{(t+1)} = \frac{\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right) \vec{x}_j}{\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right)}.$$

For the covariance matrices, we take the update rule of the expectation maximization algorithm for a Gaussian mixture model and introduce membership transformations. This is equivalent to computing fuzzy covariance matrices as they were defined on page 136 and refraining from normalizing them to unit determinant. That is, $\forall i; 1 \leq i \leq c$:

$$\boldsymbol{\Sigma}_i^{(t+1)} = \widehat{\boldsymbol{\Sigma}}_i^{(t+1)} = \frac{\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right) \left(\vec{x}_j - \vec{\mu}_i^{(t+1)}\right) \left(\vec{x}_j - \vec{\mu}_i^{(t+1)}\right)^{\top}}{\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right)}.$$

(Note that this matrix need not have unit determinant, and thus differs from the standard approach in fuzzy clustering. At least it cannot be derived from the objective function—see the discussion in Section 5.2.3.)

In addition, we need an update rule for the prior probability, which is also obtained by introducing a membership transformation into the corresponding update rule of the expectation maximization algorithm. That is,

$$\varrho_i^{(t+1)} = \frac{\sum_{j=1}^{n} h\left(u_{ij}^{(t+1)}\right)}{\sum_{k=1}^{c} \sum_{j=1}^{n} h\left(u_{kj}^{(t+1)}\right)}.$$

Unfortunately, in contrast to the original rule it was derived from, this rule cannot be simplified, since no constraint holds for the transformed membership degrees, but only for the untransformed ones.

5.3 Competitive Learning

Competitive learning, likely to be even better known under the name of **learning vector quantization**, was developed mainly in the area of artificial neural networks [Gersho 1982, Gray 1984, Kohonen 1986, DeSieno 1988, Kohonen 1990, Kohonen 1995, NNRC 2002]. It has the advantage that it can be applied for supervised as well as for unsupervised learning, that is, for classification as well as for clustering tasks. The training approach is closely related to gradient descent (as already pointed out in Section 5.1.2), although it has a much more heuristic and *ad hoc* basis. Furthermore, it is closely related to classical c-means clustering, which is very similar to a "batch version" of learning vector quantization.

The general idea of competitive learning is that a set of reference vectors (or cluster centers) in the data space compete over each data point. The reference vector that is most similar to a data point wins the competition and is adapted subsequently. The adaptation is done in such a way that the reference vector becomes even more similar to the data point. In order to enforce convergence, the adaptation is reduced over time.

In the following sections I consider first classical winner-takes-all learning vector quantization, in which only the winning reference vector is adapted. Then I turn to a fuzzy counterpart, in which multiple reference vectors may be adapted according to their relative similarity to a data point. Next I study how shape and size parameters, which are missing from the original approach, can be introduced into learning vector quantization in a very natural way. Finally, I discuss a statistical justification of learning vector quantization for classification purposes that is based on the objective function of maximum likelihood ratio. As it turns out, learning vector quantization can be seen as limiting case, a view that also gives rise to a better version, which works without a "window rule" for the adaptation.

5.3.1 Classical Learning Vector Quantization

As already outlined above, classical learning vector quantization works with a set a reference vectors that are adapted in such a way that each reference vector captures a set of similar data points. These reference vectors are initialized randomly (see Chapter 4 for some methods) and then adapted in an "online" fashion. That is, the data points are considered one after the other and for each of them one reference vector is adapted. This reference vector is determined by a "competition" between all reference vectors, which is won by the one that is most similar to the currently processed data point.

The similarity of a reference vector and a data point is measured in the same way as we have always done it up to now, namely with the help of a similarity function that is defined on a distance measure. Since a similarity function is usually required to be monotonous (see Section 2.2), we may also say that the competition is won by the reference vector that is closest to the currently processed data point. For classical learning this formulation is surely preferable. However, when we introduce fuzzy learning vector quantization in the next section, the similarity functions enter the consideration again and therefore I prefer the first statement.

After the winning reference vector has been determined, it is updated immediately (that is, before the next data point is processed). If the task is clustering, there is only one update rule, namely

$$\vec{\mu}_k^{(t+1)} = \vec{\mu}_k^{(t)} + \eta_{\vec{\mu}} \left(\vec{x} - \vec{\mu}_k^{(t)} \right),$$

where k, $1 \leq k \leq c$, is the index of the winning reference vector $\vec{\mu}_k$, $\vec{x}$ is the processed data point, and $\eta_{\vec{\mu}}$ is a (positive) learning rate[9] less than 1. The idea of this rule is to update the reference vector in such a way that it becomes even more similar to the processed data point. It is, so to say, "attracted" by the data point, which is why I call the above rule the **attraction rule**.[10] This rule is illustrated in the left diagram in Figure 5.7.

With such a rule it is plausible that each reference vector will end up in the middle of a group of similar data points, for which it wins the competition. This becomes even clearer if we rewrite the update rule as

$$\vec{\mu}_k^{(t+1)} = (1 - \eta_{\vec{\mu}})\, \vec{\mu}_k^{(t)} + \eta_{\vec{\mu}}\, \vec{x}.$$

This equation makes it obvious that the update models an **exponential decay** of information incorporated earlier into the reference vector $\vec{\mu}$ (as it is multiplied with a factor less than 1—actually the new reference vector is a convex combination of the old reference vector and the data point). Hence, after sufficiently many updates, the reference vectors will depend almost exclusively on the data points closely surrounding them.

However, in the above simple form, the update rule is susceptible to oscillations and cyclic updates. To understand this, consider the extreme case that is depicted in the left diagram in Figure 5.8: a single reference vector is repeatedly updated with four data points that are placed at the

[9] The learning rate η has an index $\vec{\mu}$ here, because later we will also consider learning rates for other cluster parameters (like size and shape, see Section 5.3.3).

[10] Later I will also introduce a *repulsion rule*, namely in the context of classification.

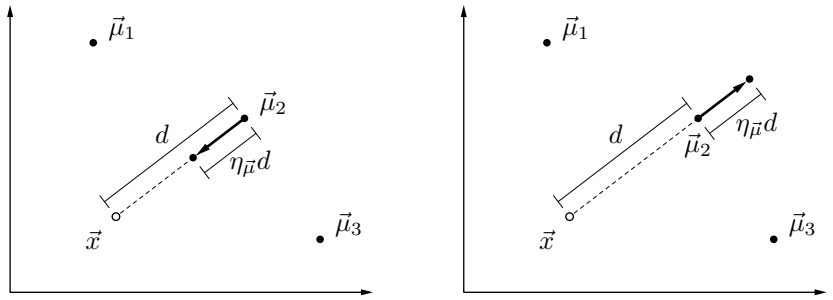

Figure 5.7: Adaptation of reference vectors (•) with a single data point (○), $\eta_{\vec{\mu}} = 0.4$. Left: attraction rule, right: repulsion rule.

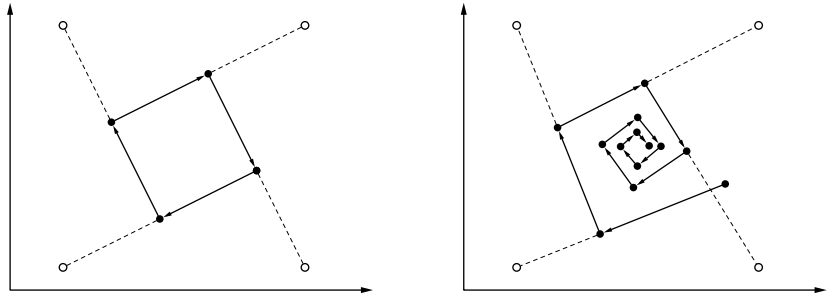

Figure 5.8: Adaptation of a single reference vector with four data points. Left: constant learning rate $\eta_{\vec{\mu}}(t) = 0.5$, right: continuously decreasing learning rate $\eta_{\vec{\mu}}(t) = 0.6 \cdot 0.85^t$. In the first step it is $t = 0$.

four corners of a square. Due to the relation of the initial position of the reference vector and the learning rate, the reference vector moves in a cycle of length four. Although situations of such high symmetry are unlikely in practice, it is clear that similar cyclic updates (though with more irregular cycles, which may also not be so perfectly closed) are likely to occur. In such situations the center of the group of data points, which would be the desired final position of the reference vector, is never reached.

The standard approach to enforce convergence in such a case is to make the update steps smaller and smaller over time, so that a cycle turns into a spiral. That is, while up to now I considered a constant learning rate $\eta_{\vec{\mu}}$, I

now introduce a **time-dependent learning rate**, for instance,

$$\eta_{\vec{\mu}}^{(t)} = \eta_{\vec{\mu}}^{(0)} \alpha^t, \quad 0 < \alpha < 1, \qquad \text{or} \qquad \eta_{\vec{\mu}}^{(t)} = \eta_{\vec{\mu}}^{(0)} t^{\kappa}, \quad \kappa < 0,$$

for $t = 1, 2, \ldots$ As an illustration of the resulting effect, the right diagram in Figure 5.8 shows the update of a single reference vector with the same four data points as considered above. However, this time the learning rate is reduced over time. Consequently, the update cycle turns into a spiral and thus the reference vector will finally come to a rest in the center of the square that is formed by the four data points—as desired.

However, although a time-dependent learning rate guarantees convergence, one has to pay attention not to let the learning rate decrease too quickly over time. Otherwise one may observe a "starvation" effect, that is, the update steps get very quickly very small, so that the reference vectors cannot reach or even come close to their natural destination. On the other hand, letting it decrease too slowly can lead to slow convergence. As usual, the proper choice of the learning rate is a difficult issue.

Although the update rule as it was stated above already describes the update process fully, it is technically convenient to write it as a rule that refers to all reference vectors, that is, as $\forall i; 1 \leq i \leq c$: $\forall j; 1 \leq j \leq n$:

$$\vec{\mu}_i^{(t+1)} = \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}}\, u_{ij}^{(t)} \left(\vec{x}_j - \vec{\mu}_i^{(t)} \right),$$

where $u_{ij}^{(t)} = \delta_{i,k(j,t)}$ with $k(j,t) = \operatorname{argmin}_{l=1}^{c} d(\vec{x}_j, \vec{\mu}_l^{(t)})$. Here $\delta_{i,k}$ is the Kronecker symbol (cf. page 60 in Section 3.2 for a definition), which has the effect that only the winning reference vector is adapted, while all others stay unchanged (because $u_{ij}^{(t)} = 1$ only for the closest reference vector and $u_{ij}^{(t)} = 0$ otherwise). Note that the u_{ij}, $1 \leq i \leq c$, $1 \leq j \leq n$, in this rule correspond directly to the u_{ij} in classical (crisp) c-means clustering (cf. page 46 in Section 3.1), which justifies the use of this symbol.

The correspondence to c-means clustering becomes even more obvious if we consider a *batch update* of the reference vectors. In a batch update the winning reference vector is not updated immediately after each data point that is processed, but the changes are aggregated in separate variables and applied only at the end of an **epoch**[11], that is, after all data points have been processed. The corresponding update rule is therefore

$$\vec{\mu}_i^{(t+1)} = \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}} \sum_{j=1}^{n} u_{ij}^{(t)} \left(\vec{x}_j - \vec{\mu}_i^{(t)} \right).$$

[11] In neural network training one traversal of the whole data set is called an *epoch*.

(Note that the time steps are different in this formula compared to the one above for an *online update*. While for an online update the time indicator is incremented with each data point that is processed, in batch update it is incremented only with each epoch and thus much more slowly.) Rewriting the above batch update rule in the same way as the online update rule yields

$$\vec{\mu}_i^{(t+1)} = \left(1 - \eta_{\vec{\mu}} \sum_{j=1}^{n} u_{ij}^{(t)}\right) \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}} \sum_{j=1}^{n} u_{ij}^{(t)} \vec{x}_j,$$

which again reveals learning vector quantization as a method of exponential decay of information. Furthermore, this formula suggests to choose a time- and cluster-dependent learning rate

$$\eta_{\vec{\mu}_i}^{(t)} = \frac{1}{\sum_{j=1}^{n} u_{ij}^{(t)}}.$$

With such a learning rate the update rule turns into

$$\vec{\mu}_i^{(t+1)} = \frac{\sum_{j=1}^{n} u_{ij}^{(t)} \vec{x}_j}{\sum_{j=1}^{n} u_{ij}^{(t)}},$$

that is, the standard update rule for (crisp) c-means clustering. Note that a similar connection was pointed out for gradient descent on the sum of squared errors for fuzzy clustering, which led to almost the same update rule (cf. page 106 in Section 5.1.2). The only differences are the restriction of the u_{ij} to the set $\{0, 1\}$ (which I will drop in the next section, though) and the missing fuzzifier w. As a consequence, gradient descent, alternating optimization, and learning vector quantization turn out to be very similar approaches (at least w.r.t. the update of cluster centers/reference vectors), even though their standard forms exhibit, of course, certain differences.

As an illustration of both online and batch training for learning vector quantization, Figure 5.9 shows both update methods applied to a very simple data set with 15 data points. The left diagram shows the online update, as can be seen from the zig-zag shape of the trajectories of the reference vectors, which is particularly pronounced for the middle one. As long as this reference vector has not been attracted far enough to the data point group in the lower right corner, it wins over the uppermost reference vector for the data points in the upper middle of the diagram and thus is temporarily dragged back and forth between two groups of data points. In contrast to this the batch update, which is depicted in the right diagram, is much smoother, but, as usual, also somewhat slower than the online update.

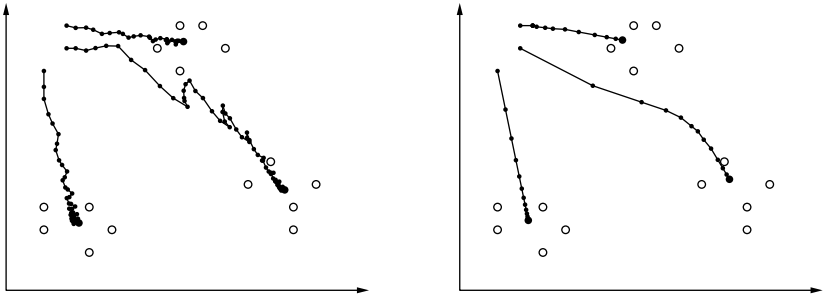

Figure 5.9: Learning vector quantization with 3 reference vectors for a simple set of 15 data points. Left: online training with (constant) learning rate $\eta_{\vec{\mu}} = 0.1$, right: batch training with (constant) learning rate $\eta_{\vec{\mu}} = 0.05$.

Up to now I considered learning vector quantization only as a method for clustering. However, it may just as well be applied to solve classification tasks, namely by building a *nearest prototype classifier* (cf. page 38 in Section 2.5 for a definition of this type of classifier). To this end each reference vectors is endowed not only with an initial position, but also with a class label, which, however, is never changed in the update process.

Based on the class labels, the update rule is modified as follows: if the winning reference vector for a data point has the same class label as the data point, it is adapted in the same way as above, that is, using the *attraction rule*. If, however, the class labels differ, the **repulsion rule** is applied, which is distinguished from the attraction rule only by the sign of the change:

$$\vec{\mu}_k^{(t+1)} = \vec{\mu}_k^{(t)} - \eta_{\vec{\mu}} \left(\vec{x} - \vec{\mu}_k^{(t)} \right)$$

(see the right diagram in Figure 5.7 for an illustration). Thus it can easily be merged with the attraction rule if we use indicators u_{ij} in the same way as we did above, that is, if we define the update rule as

$$\vec{\mu}_i^{(t+1)} = \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}}\, u_{ij}^{(t)} \left(\vec{x} - \vec{\mu}_i^{(t)} \right).$$

Then we simply set the indicators u_{ij}, $1 \leq i \leq c$, $1 \leq j \leq n$, according to

$$u_{ij}^{(t)} = \begin{cases} +1, & i = \mathrm{argmin}_{l=1}^{c}\, d\big(\vec{x}_j, \vec{\mu}_l^{(t)}\big) \wedge z_j = \zeta_i, \\ -1, & i = \mathrm{argmin}_{l=1}^{c}\, d\big(\vec{x}_j, \vec{\mu}_l^{(t)}\big) \wedge z_j \neq \zeta_i, \\ 0, & \text{otherwise.} \end{cases}$$

Here z_j, $1 \leq j \leq n$, is the class associated with the data point $\vec{x}_j$ and ζ_i, $1 \leq i \leq c$, is the class associated with the reference vector $\vec{\mu}_i$. Like for learning vector quantization for clustering, the learning rate is made time-dependent in order to enforce convergence. This approach is a simple, yet fairly effective way of learning nearest prototype classifiers.

Improved versions of learning vector quantization for classification update not only the one reference vector that is closest to the currently processed data point, but the two closest ones [Kohonen 1990, Kohonen 1995]. Let $\vec{\mu}_k$ and $\vec{\mu}_l$ be these two closest reference vectors. Then an update is carried out if the classes ζ_k and ζ_l assigned to these reference vectors differ, but one of them coincides with the class z_j assigned to the data point $\vec{x}_j$. W.l.o.g. we assume that $\zeta_k = z_j$. Then the update rules are

$$\vec{\mu}_k^{(t+1)} = \vec{\mu}_k^{(t)} + \eta_{\vec{\mu}} \left(\vec{x} - \vec{\mu}_k^{(t)} \right) \qquad \text{and}$$
$$\vec{\mu}_l^{(t+1)} = \vec{\mu}_l^{(t)} - \eta_{\vec{\mu}} \left(\vec{x} - \vec{\mu}_l^{(t)} \right),$$

while all other reference vectors remain unchanged. If, on the other hand, the classes of the two closest reference vectors coincide with each other, regardless of whether they coincide with the class of the data point or not, no reference vector is adapted at all. These rules have been found to yield good nearest prototype classifiers [Kohonen 1990].

Unfortunately, it was also observed in practical tests that this version of learning vector quantization may drive the reference vectors further and further apart on certain data sets, instead of leading to a stable convergence. To counteract this obviously undesired behavior, [Kohonen 1990] introduced a so-called **window rule** into the update: the reference vectors are updated only if the data point $\vec{x}_j$ is close to the classification boundary, where "close" is made formally precise by requiring

$$\min \left(\frac{d(\vec{x}_j, \vec{\mu}_k)}{d(\vec{x}_j, \vec{\mu}_l)}, \frac{d(\vec{x}_j, \vec{\mu}_l)}{d(\vec{x}_j, \vec{\mu}_k)} \right) > \theta, \qquad \text{where} \qquad \theta = \frac{1-\xi}{1+\xi}.$$

Here ξ is a parameter that has to be specified by a user. Intuitively, ξ describes the "width" of the window around the classification boundary, in which the data point has to lie in order to lead to an update. Using it prevents divergence, because the update ceases for a data point once the classification boundary has been moved far enough away.

Nevertheless, this window rule is a bit counterintuitive and it would be preferable to do without it. This is actually possible, as will be shown in Section 5.3.3, in which a method closely related to learning vector quantization is derived as a limiting case of optimizing the likelihood ratio.

5.3.2 Fuzzy Learning Vector Quantization

Classical learning vector quantization employs a winner-takes-all principle: only the winning reference vector, that is, the closest or most similar reference vector, is adapted. However, in analogy to the extension of classical crisp c-means clustering to fuzzy c-means clustering, one may consider a fuzzified version, in which multiple reference vectors are adapted according to their respective distance or similarity to a data point.

A straightforward way to introduce membership degrees into learning vector quantization is to employ a similarity function like, for instance, the Cauchy or Gaussian radial functions, which I generally use for this purpose in this thesis. In neural network terms this function yields an **activation** of the neuron that represents a reference vector. The idea is to weight the update of a reference vector with this activation (in addition to a time-dependent learning rate). However, this scheme, which is closely related to *possibilistic fuzzy clustering*, usually leads to unsatisfactory results, since there is no dependence between the clusters/reference vectors, so that they tend to end up at the center of gravity of all data points (cf. the extensive discussion of possibilistic fuzzy clustering in Section 3.1).

An alternative is to rely on a normalization scheme as in *probabilistic fuzzy clustering* (cf. also Section 3.1). That is, one computes the weight for the update of a reference vector as the *relative* inverse (squared) distance from this vector, or generally as the *relative* activation of a neuron. This is the approach I employ here, which I presented in [Borgelt and Nürnberger 2004a, Borgelt *et al.* 2005]. It is closely related to the approach by [Tsao *et al.* 1994], but differs from it in as far as it uses a fixed instead of a varying fuzzifier (which is used to obtain a smooth transition from fuzzy to hard clustering). Other discussions of related aspects of fuzzy learning vector quantization can be found in [Yair *et al.* 1992, Bezdek and Pal 1995, Zhang *et al.* 2004]. However, one should note that basically all of these approaches (including the one by [Tsao *et al.* 1994]) are restricted to batch learning vector quantization, while I strive here to develop an online version in order to exploit the usually higher speed of such an approach.

The update rule for fuzzy learning vector quantization is basically the same as above, only that it necessarily refers to all vectors, i.e. $\forall i; 1 \leq i \leq c$:

$$\vec{\mu}_i^{(t+1)} = \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}} \left(u_{ij}^{(t)}\right)^w \left(\vec{x} - \vec{\mu}_i^{(t)}\right).$$

The main differences consist in the range of values for the u_{ij}, which is now the real interval $[0, 1]$, and that a *fuzzifier* or *weighting exponent* w

is introduced as in fuzzy clustering. (Note that such an exponent has no effect in classical (crisp) learning vector quantization due to the restriction of the u_{ij} to the set $\{0, 1\}$.) The *membership degrees* or *activations* u_{ij} are computed as $\forall i; 1 \leq i \leq c$: $\forall j; 1 \leq j \leq n$:

$$u_{ij} = u_i(\vec{x}_j),$$

where it is important that they are normalized.[12] A normalization to sum 1 is, of course, the most natural choice, but one may also consider a normalization to maximum 1 (cf. Section 2.4 about normalization modes).

In contrast to fuzzy clustering, where a fuzzifier $w = 1$ leads effectively to crisp clustering, this choice does not have such an effect in learning vector quantization, since the update rule is derived in a different way. Rather, $w = 1$ is the most natural choice, which keeps fuzzy learning vector quantization as close as possible to the classical approach. Hence, by rewriting the update rule (in its online or batch form) into a scheme of exponential decay and choosing a cluster- and time-dependent learning rate (see above), we can obtain from fuzzy learning vector quantization the update rule for the cluster centers/reference vectors as it results from expectation maximization (cf. Section 5.2.4), namely for $w = 1$, as well as the update rule for fuzzy clustering, namely for $w > 1$. Therefore fuzzy learning vector quantization can be seen as mediating between the two approaches.

Although the choice $w = 1$ is unproblematic now and thus we do not obtain crisp clustering in the limit for $w \to 1_+$ as in fuzzy clustering, classical (crisp) learning vector quantization can nevertheless be seen as a limiting case of fuzzy learning vector quantization. It results in the limit if we let the radius of the clusters, as it enters the radial similarity or activation function, go to zero. The reason is that decreasing the cluster radius effectively increases the distance of the data points to the cluster centers/reference vectors. Hence we get the effects that were studied in Section 2.4 in connection with the limiting behavior of the membership degrees for large distances w.r.t. a normalization to sum 1 or maximum 1 and their computation from the Cauchy and Gaussian radial function (cf. page 32f). The interesting case here is the Gaussian radial function. If we let the radius of the clusters go to zero, even the membership degrees of data points close to the reference vectors (close w.r.t. an unmodified distance) get more and more extreme. In the limit, for a vanishing cluster radius, we get a hard assignment to the closest cluster center/reference vector.

[12] Unnormalized membership degrees $u_{ij} = u_i^\circ(\vec{x}_j)$ have the disadvantage pointed out above, which are the same as those of possibilistic fuzzy clustering.

It is fairly obvious that with the described approach batch fuzzy learning vector quantization is very similar to fuzzy clustering and can be made equivalent by using the cluster- and time-dependent learning rate mentioned in the preceding section. If we choose a smaller, but still cluster-specific and time-dependent learning rate, we make a step from the position of the old reference vector towards the center of gravity of the weighted data points. This can easily be seen by writing the learning rate as

$$\eta_{\vec{\mu}_i}^{(t)} = \frac{\kappa}{\sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^w}, \qquad \kappa \in (0,1).$$

(cf. Section 5.1.2 for an analogous suggestion). With this learning rate we obtain for a batch version of fuzzy learning vector quantization

$$\vec{\mu}_i^{(t+1)} = (1-\kappa) \cdot \vec{\mu}_i^{(t)} + \kappa \cdot \frac{\sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^w \vec{x}_j}{\sum_{j=1}^{n} \left(u_{ij}^{(t)}\right)^w}.$$

However, I am more interested in an online version, updating the cluster parameters more often in order to achieve faster convergence. In particular, I desire a scheme that allows to update the reference vectors not only after single data points (this case does not pose any particular problems), but after a group of data points, which are processed in a batch manner. The objective of such an approach is to combine the higher speed of an online update with the superior robustness of a batch update. In addition, an online update of other cluster parameters (which will be studied in the next section), may be too expensive if carried out for every single data point. For example, it is not advisable to update a covariance matrix, which captures the size and shape of a cluster, after each individual data point.

Therefore I consider an approach, in which an update is performed after a user-specified number of data points, which may be adapted depending on the size of the data set. However, this makes it slightly difficult to choose a proper learning rate, especially since the weights (sum of membership degrees) may differ for each cluster/reference vector and in particular from the relative weights of a batch update. As a solution I propose a scheme that is inspired by the similarity of learning vector quantization to fuzzy clustering, as it was also exploited for the special batch update rule above. That is, I suggest to update a reference vector according to [Borgelt *et al.* 2005]

$$\vec{\mu}_i^{(t+1)} = \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}}^{*(t)} \sum_{j \in J} \left(u_{ij}^{(t)}\right)^w \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right),$$

where J is the index set of the processed subset of data points and

$$\eta_{\vec{\mu}}^{*(t)} = \begin{cases} \eta_{\vec{\mu}}^{(t)}, & \text{if } \sum_{j \in J} \left(u_{ij}^{(t)}\right)^w \leq 1 \\ \dfrac{\eta_{\vec{\mu}}^{(t)}}{\sum_{j \in J} \left(u_{ij}^{(t)}\right)^w}, & \text{otherwise.} \end{cases}$$

The idea of this rule is that the update step is limited by a basic learning rate even if the sum of membership degrees to a cluster is small. As a consequence, a reference vector is always moved only a fraction of the distance towards the center of gravity of the weighted data points, specified by the learning rate. Again this becomes obvious if the update rule is rewritten into a scheme of exponential decay (as above). Defining the learning rate in this way makes is fairly easy to choose a proper learning rate. In experiments this rule exhibited a very robust behavior (cf. Section 8.3).

5.3.3 Size and Shape Parameters

Up to now the clusters that are found with learning vector quantization were described only by reference vectors, which play exactly the same role as cluster centers. No information about the size or the shape of the clusters was included in the model. However, drawing on the analogy of (fuzzy) learning vector quantization to fuzzy clustering and expectation maximization, the idea suggests itself to introduce size and shape parameters into this approach as well. Of course, this is simpler in a batch version, but as we will see below, may also be employed in an online update scheme.

Early approaches to find clusters of different size are based on the relative weight of the assigned data points and thus try to find clusters that do not differ too much in the number of data points they cover. An example is so-called **frequency sensitive competitive learning** [DeSieno 1988], in which the distance to a reference vector is modified according to the number of data points that are assigned to this reference vector, i.e.

$$d_{ij}^{(t)} = d_{\text{mod}}^{(t)}\left(\vec{x}_j, \vec{\mu}_i^{(t)}\right) = \frac{n_i^{(t-1)}}{n} d\left(\vec{x}_j, \vec{\mu}_i^{(t)}\right),$$

where $n_i^{(t-1)}$, $1 \leq i \leq c$, is the number of data points that were assigned to the reference vector $\vec{\mu}_i$ in the preceding epoch and n is the total number of data points. That is, the distance of a data point to a reference vector is the smaller, the fewer data points that reference vector covers. Consequently there is a tendency to equalize the number of covered data points.

Note that this approach is equivalent to using a time-dependent cluster radius $\sigma^{(t)} = n/n_i^{(t-1)}$ to modify the distance function, or formally

$$d_{ij}^{(t)} = d\left(\vec{x}_j, \vec{\mu}_i^{(t)}; \frac{n}{n_i^{(t-1)}}\mathbf{1}\right).$$

Drawing on this idea, one may also state explicitely that the goal is to assign (roughly) the same number of data points to each cluster. That is, one sets the goal to cover $\frac{n}{c}$ data points with each cluster/reference vector, where n is the total number of data points and c the number of clusters. If a given (fixed or current) radius σ_i, $1 \leq i \leq c$, leads to an assignment of n_i data points, the **desired reference radius** is computed as [Borgelt *et al.* 2004]

$$\sigma_i^{(\text{desired})} = \sigma_i \cdot \frac{n}{c \cdot n_i}.$$

The rationale is to decrease the radius if the desired number of data points is less than the current number and to increase it if it is greater, thus balancing the number of data points that are assigned to each reference vector.

An alternative consists in computing a desired reference radius from the average distance of the data points to a reference vector (or, alternatively, from the square root of the average squared distance), where on average the data points are weighted with the activation/membership degree. Then the reference radius may simply be set to this desired radius (of course, this is only reasonable in batch update), or, more cautiously, changed in the direction of the desired radius using a learning rate as for the update of the reference vectors. A corresponding online learning rule is $\forall i; 1 \leq i \leq c$:

$$\sigma_i^{(t+1)} = \sigma_i^{(t)} + \eta_\sigma \left(d\left(\vec{x}, \vec{\mu}_i^{(t)}\right) - \sigma_i^{(t)}\right),$$

where η_σ may or may not be the same learning rate as the one that is used for the reference vectors themselves [Borgelt *et al.* 2004]. Intuitively, the cluster radius is increased if the data point lies outside the current radius and it is decreased if it lies inside. Thus, in the limit, the radius should converge to the average distance of the data points. In [Acciani *et al.* 1999] a slightly more complex update scheme is used, which distinguishes whether a data point is inside the (hyper-)sphere defined by the current radius (then only this radius, i.e., the one for the corresponding winning reference vector, is decreased) or outside the radius (hyper-)spheres of all clusters (then all radii, i.e. the radii of all reference vectors, are increased).

In a fuzzy approach to learning vector quantization it should be noted that if we do not normalize the activations/membership degrees, which we use for computing the average distance in the above scheme, size adaptation can be slightly problematic. The reason is that in this case the sum of the activations over all reference vectors and all data points will, in general, differ from the total number n of data points. Depending on the method to determine the desired radius and the activation function used, this can lead to collapsing clusters in some cases. To counteract this tendency, we may introduce a parameter by which we multiply the computed desired radius before we use it to adapt the current reference radius [Borgelt *et al.* 2004].

A more principled approach to introduce cluster sizes, which at the same time enables cluster-specific shape parameters, is based on using a **cluster-specific covariance matrix**, which is updated in each step [Borgelt and Nürnberger 2004a, Borgelt *et al.* 2005]. In order to find an update rule for the covariance matrix, we draw on the observation made in Section 5.3.1, namely that the update of the reference vector (in classical, crisp learning vector quantization) may also be written as

$$\vec{\mu}_i^{(t+1)} = (1 - \eta_{\vec{\mu}})\, \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}}\, \vec{x}_j.$$

This showed that the update can be seen as an exponential decay of information gained from data points processed earlier. In addition, we exploit that in c-means clustering with cluster-specific covariance matrices, these matrices can be estimated as $\forall i; 1 \leq i \leq c$:

$$\widehat{\mathbf{\Sigma}}_i^{(t+1)} = \frac{\sum_{j=1}^{n} u_{ij}^{(t+1)} \left(\vec{x}_j - \vec{\mu}_i^{(t+1)}\right) \left(\vec{x}_j - \vec{\mu}_i^{(t+1)}\right)^\top}{\sum_{j=1}^{n} u_{ij}^{(t+1)}}$$

(cf. Section 5.2.3), where the u_{ij}, $1 \leq i \leq c$, $1 \leq j \leq n$, describe the assignment of the data points to the clusters. This formula shows that each data point contributes to the covariance matrix through the outer product of the difference vector to the cluster center with itself. This insight and the transfer of the idea of exponential decay leads directly to

$$\mathbf{\Sigma}_i^{(t+1)} = (1 - \eta_{\mathbf{\Sigma}})\, \mathbf{\Sigma}_i^{(t)} + \eta_{\mathbf{\Sigma}} \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right) \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)^\top$$

for the update of the covariance matrix of the winning reference vector, where $\eta_{\mathbf{\Sigma}}$ is a learning rate. Usually $\eta_{\mathbf{\Sigma}}$ will differ from the learning rate $\eta_{\vec{\mu}}$ for the reference vectors (it should be chosen much smaller than $\eta_{\vec{\mu}}$). In the fuzzy case this update may be weighted, as the update of the reference vectors, by $h(u_{ij})$, that is, we use $\eta_{\mathbf{\Sigma}} \cdot h(u_{ij})$ instead of just $\eta_{\mathbf{\Sigma}}$.

Obviously, this update rule is very similar to Gustafson–Kessel fuzzy clustering (cf. Section 5.2.3), especially if we use batch update and introduce activations/membership degrees to arrive at a fuzzy approach. Note that we may or may not normalize the covariance matrices to determinant 1 after the update, so the rule may also update the cluster size. For an online update rule, it is important to note that it is too expensive to update the covariance matrices after each training pattern. Therefore I recommend to update them only after a user-specified number of data points, which may be adapted depending on the size of the data set (as already discussed for pure reference vectors in Section 5.3.2). In this case the scheme for modifying the learning rate that was proposed on page 161 in Section 5.2.3 should be used for the update of the covariance matrices as well. As already mentioned, this rule exhibited a very stable behavior in experiments (cf. Section 8.3).

5.3.4 Maximum Likelihood Ratio

At the end of Section 5.3.1 I mentioned an improved version of learning vector quantization for classification, which adapts the *two* closest reference vectors, but only if their classes differ and one of the classes coincides with the class of the data point [Kohonen 1990, Kohonen 1995]. In this section I review an approach to justify this update rule in a principled way from a gradient descent scheme [Seo and Obermayer 2003]. It also provides means to get rid of the *window rule*—a heuristic that was introduced to avoid divergence in the update of the reference vectors.

This approach is based on the assumption that the probability distribution of the data points for each class can be described well by a mixture of Gaussian distributions. However, we may just as well employ the generalized forms of the Gaussian or Cauchy function (normalized, of course, to integral 1) to describe the distributions. I confine myself here to standard Gaussian distributions merely for the sake of simplicity.

Furthermore, since with learning vector quantization one tries to build a *nearest prototype classifier* (cf. page 38 in Section 2.5 for a definition), the Gaussian distributions are assumed to have fixed and equal standard deviations σ (that is, the clusters have uniform size and (hyper-)spherical shape) as well as equal prior probabilities $\varrho = \frac{1}{c}$. With these constraints only the distance of a data point to the cluster center/reference vector decides about the classification of the data point.

The learning procedure is derived from the objective function of *maximum likelihood ratio*, with which one tries to maximize the odds of a correct classification for each data point (cf. Section 3.4). As we will see below, it

then turns out that the attraction rule is a result of maximizing the posterior probability of the correct class (i.e. the true class assigned to the data point under consideration), while the repulsion rule is a consequence of the minimization of the posterior probability of the wrong class [Seo and Obermayer 2003]. Formally, we perform a (stochastic) gradient descent on the maximum likelihood ratio, as it was considered briefly (though not in detail) in Section 3.4. However, as I already pointed out several times in this chapter, gradient descent and competitive learning are very closely related methods and thus it is not surprising that insights about the one can be gained by considering the other and transferring the result.

Based on the above assumption we get for the first likelihood ratio

$$\ln L^{(1)}_{\text{ratio}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) = \sum_{j=1}^{n} \ln \sum_{y \in I(z_j)} \exp\left(-\frac{(\vec{x}_j - \vec{\mu}_y)^\top (\vec{x}_j - \vec{\mu}_y)}{2\sigma^2}\right) - \sum_{j=1}^{n} \ln \sum_{y \in \{1,\ldots,c\} - I(z_j)} \exp\left(-\frac{(\vec{x}_j - \vec{\mu}_y)^\top (\vec{x}_j - \vec{\mu}_y)}{2\sigma^2}\right).$$

Note that the normalization factors, which scale the Gaussian function to integral 1, cancel, because all clusters are endowed with the same standard deviation/variance. This is very obvious if the above is actually written as a ratio—cf. Section 3.4. Likewise, the prior probabilities of the clusters cancel, because we assumed them to be equal.

From this objective function we get almost immediately[13] as an online gradient descent update rule $\forall i; 1 \leq i \leq c$: $\forall j; 1 \leq j \leq n$:

$$\begin{aligned} \vec{\mu}_i^{(t+1)} &= \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}} \cdot \nabla_{\vec{\mu}_i} \ln L^{(1)}_{\text{ratio}}(\mathbf{X}, \vec{z}; \mathbf{C}^{(t)}, \mathbf{W}) \\ &= \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}} \cdot \begin{cases} u_{ij}^{\oplus(t)} \cdot \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right), & \text{if } z_j = \zeta_i, \\ -u_{ij}^{\ominus(t)} \cdot \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right), & \text{if } z_j \neq \zeta_i, \end{cases} \end{aligned}$$

where ζ_i, as above, is the class associated with the i-th reference vector. The "membership degrees" $u_{ij}^{\oplus}$ and $u_{ij}^{\ominus}$ are

$$u_{ij}^{\oplus(t)} = \frac{\exp\left(-\frac{1}{2\sigma^2}\left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)^\top \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)\right)}{\sum_{y \in I(z_j)} \exp\left(-\frac{1}{2\sigma^2}\left(\vec{x}_j - \vec{\mu}_y^{(t)}\right)^\top \left(\vec{x}_j - \vec{\mu}_y^{(t)}\right)\right)} \quad \text{and}$$

[13] The derivation given by [Seo and Obermayer 2003] is unnecessarily complex, because in it they do not exploit $\log \frac{a}{b} = \log a - \log b$ for some unknown reason.

$$u_{ij}^{\ominus(t)} = \frac{\exp\left(-\frac{1}{2\sigma^2}\left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)^\top \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)\right)}{\sum_{y \in \{1,\ldots,c\}-I(z_j)} \exp\left(-\frac{1}{2\sigma^2}\left(\vec{x}_j - \vec{\mu}_y^{(t)}\right)^\top \left(\vec{x}_j - \vec{\mu}_y^{(t)}\right)\right)}.$$

Note the symbols $u_{ij}^{\oplus}$ and $u_{ij}^{\ominus}$ are justified by the fact that these terms are computed in basically the same way as the membership degrees in fuzzy clustering or the posterior probabilities of the clusters in expectation maximization, for which I used the same symbols. The split into the two cases $i = z_j$ (the class of the data point coincides with that of the distribution) and $i \neq z_j$ (the distribution has a different class than the data point) is obvious from the fact that each cluster center μ_i appears in only one of the two sums: either its index i is contained in $I(z_j)$ and then only the first sum contributes to the derivative, or it is not contained in $I(z_j)$ and then only the second sum contributes. The denominators of the fractions for u_{ij} and $\overline{u}_{ij}$ clearly result from taking the derivative of the natural logarithm.

The result is a **soft learning vector quantization** scheme [Seo and Obermayer 2003] (soft, because it does not employ a winner-takes-all principle for the update), which is very similar to *fuzzy learning vector quantization* as it was studied in Section 5.3.2. The difference consists in the missing exponent w for the u_{ij}—the standard difference between a fuzzy- or distance-based and a probability-based approach (cf. Section 5.2.3 and 5.2.4 for a similar case). With this approach all reference vectors having the same class as the data point under consideration are "attracted" to the data point, while all other reference vectors are "repelled" by the data point.

Hard or **crisp learning vector quantization** is easily derived from this scheme in the same way as for its clustering counterpart (cf. page 160 in Section 5.3.2), namely by letting the reference radii of the clusters go to zero. As studied on page 32f in Section 2.4, this leads to increasingly extreme values for the u_{ij} and thus, in the limit for $\sigma \to 0$ to a hard assignment

$$u_{ij}^{\oplus} = \delta_{i,k^{\oplus}(j)}, \quad \text{where} \quad k^{\oplus}(j) = \operatorname*{argmin}_{l \in I(z_j)} d(\vec{x}_j, \vec{\mu}_l), \quad \text{and}$$
$$u_{ij}^{\ominus} = \delta_{i,k^{\ominus}(j)}, \quad \text{where} \quad k^{\ominus}(j) = \operatorname*{argmin}_{l \in \{1,\ldots,c\}-I(z_j)} d(\vec{x}_j, \vec{\mu}_l).$$

Note, however, that this scheme is *not* identical to the scheme proposed by [Kohonen 1990, Kohonen 1995]. While Kohonen's scheme determines the two closest reference vectors and updates them only if they belong to different classes, one of which equals the class of the data point, this scheme *always* updates two reference vectors: the closest reference vector among

those having the same class (this vector is attracted) and the closest reference vector among those having a different class (this vector is repelled). Note that these need not be the two overall closest reference vectors. Although one of them must be the one overall closest reference vector, the other can be further away than several other reference vectors. Nevertheless it is clear that the two schemes are very similar.

One advantage of the above approach is that it provides means to analyze the reasons for the sometimes diverging behavior of this version of learning vector quantization for classification [Seo and Obermayer 2003]: We compute the expected value of the logarithm of the likelihood ratio, which yields an insight into its behavior in the limit, that is, for a number n of data points that goes to infinity. Assuming that the true probability distribution underlying the data generation can be described by the cluster parameters $\mathbf{C}^*$, this expected value is for soft learning vector quantization

$$\begin{aligned} & E(\ln L^{(1)}_{\text{ratio}}(\vec{x}, z; \mathbf{C}, \mathbf{W})) \\ &= \sum_{z=1}^{s} \int_{\vec{x} \in \text{dom}(\vec{X})} f_{\vec{X},Z}(\vec{x}, z; \mathbf{C}^*) \cdot \ln L^{(1)}_{\text{ratio}}(\vec{x}, z; \mathbf{C}, \mathbf{W}) \, \mathrm{d}\vec{x} \\ &= \sum_{z=1}^{s} \int_{\vec{x} \in \text{dom}(\vec{X})} f_{\vec{X},Z}(\vec{x}, z; \mathbf{C}^*) \cdot \ln \frac{f_{\vec{X},Z}(\vec{x}, z; \mathbf{C})}{f_{\vec{X},Z}(\vec{x}, \overline{z}; \mathbf{C})} \, \mathrm{d}\vec{x}. \end{aligned}$$

That is, at every point $(\vec{x}, z)$ of the data space $\text{dom}(\vec{X}) \times \{1, \ldots, s\}$ the likelihood ratio as it results from the cluster model specified by the cluster parameters $\mathbf{C}$ is weighted with the true probability of seeing the data point, as it results from the true cluster parameters $\mathbf{C}^*$. This leads to

$$\begin{aligned} & E(\ln L^{(1)}_{\text{ratio}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W})) \\ &= \sum_{z=1}^{s} \int_{\vec{x} \in \text{dom}(\vec{X})} f_{\vec{X},Z}(\vec{x}, z; \mathbf{C}^*) \cdot \ln \frac{f_{\vec{X},Z}(\vec{x}, z; \mathbf{C}) \cdot f_{\vec{X},Z}(\vec{x}, z; \mathbf{C}^*)}{f_{\vec{X},Z}(\vec{x}, \overline{z}; \mathbf{C}) \cdot f_{\vec{X},Z}(\vec{x}, z; \mathbf{C}^*)} \, \mathrm{d}\vec{x} \\ &= \sum_{z=1}^{s} \int_{\vec{x} \in \text{dom}(\vec{X})} f_{\vec{X},Z}(\vec{x}, z; \mathbf{C}^*) \cdot \ln \frac{f_{\vec{X},Z}(\vec{x}, z; \mathbf{C}^*)}{f_{\vec{X},Z}(\vec{x}, \overline{z}; \mathbf{C})} \, \mathrm{d}\vec{x} \\ &- \sum_{z=1}^{s} \int_{\vec{x} \in \text{dom}(\vec{X})} f_{\vec{X},Z}(\vec{x}, z; \mathbf{C}^*) \cdot \ln \frac{f_{\vec{X},Z}(\vec{x}, z; \mathbf{C}^*)}{f_{\vec{X},Z}(\vec{x}, z; \mathbf{C})}. \, \mathrm{d}\vec{x} \end{aligned}$$

Hence the expected value of the above likelihood ratio is the difference of two terms, both of which are a Kullback–Leibler information divergence [Kullback and Leibler 1951] between two probability density functions.

Kullback–Leibler information divergence is generally defined as

$$d_{\mathrm{KL}}(f_1, f_2) = \int_{\omega\in\Omega} f_1(\omega) \ln \frac{f_1(\omega)}{f_2(\omega)} \,\mathrm{d}\omega,$$

where f_1 and f_2 are two arbitrary probability density functions on the same sample space Ω. For discrete probability distributions the integral has to be replaced by a sum.[14] Kullback–Leibler information divergence can be seen as a measure of how much two probability distributions or density functions differ, because it can be shown that it is non-negative and zero only if the probability distributions coincide.[15] Therefore we may write

$$\begin{aligned} E(\ln L^{(1)}_{\mathrm{ratio}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W})) &= d_{\mathrm{KL}}(f_{\vec{X},Z}(\vec{x}, z; \mathbf{C}^*), f_{\vec{X},Z}(\vec{x}, \overline{z}; \mathbf{C})) \\ &- d_{\mathrm{KL}}(f_{\vec{X},Z}(\vec{x}, z; \mathbf{C}^*), f_{\vec{X},Z}(\vec{x}, z; \mathbf{C})). \end{aligned}$$

Hence the expected value of the likelihood ratio is maximized if the difference between the true density function for a class and the distribution described by the set $\mathbf{C}$ of cluster parameters is minimized (second term), while at the same time the difference between the true density function for a class and the distribution for all other classes is maximized. Obviously, the former is uncritical, but the latter can lead to divergence. The reason is that clearly two Gaussian distributions differ the more, the further apart their center vectors are. Since the reference vectors for the different classes are disjoint, the force driving them apart never vanishes. To counteract this force, it is necessary to introduce heuristics like the *window rule*.

Fortunately, the approach to justify learning vector quantization on a clean probabilistic basis also provides a solution to this problem that does without a window rule. All we have to do is to use the second likelihood ratio, as it was introduced in Section 3.4, instead of the one used above. Assuming again Gaussian distributions of equal size σ and equal weight $\varrho = \frac{1}{c}$, this second likelihood ratio reads [Seo and Obermayer 2003]

$$\begin{aligned} \ln L^{(2)}_{\mathrm{ratio}}(\mathbf{X}, \vec{z}; \mathbf{C}, \mathbf{W}) &= \sum_{j=1}^{n} \ln \sum_{y\in I(z_j)} \exp\left(-\frac{(\vec{x}_j - \vec{\mu}_y)^\top (\vec{x}_j - \vec{\mu}_y)}{2\sigma^2}\right) \\ &- \sum_{j=1}^{n} \ln \sum_{y\in\{1,\ldots,c\}} \exp\left(-\frac{(\vec{x}_j - \vec{\mu}_y)^\top (\vec{x}_j - \vec{\mu}_y)}{2\sigma^2}\right). \end{aligned}$$

[14] This explains why there are both an integral and a sum in the above expression for the expected value of the likelihood ratio: $\mathrm{dom}(\vec{X})$ is a continuous space, while Z has the finite set $\{1, \ldots, s\}$ as its domain. Hence both versions have to be combined.

[15] A proof of this statement can be found, for example, in [Borgelt and Kruse 2002].

Note that the only difference to the first likelihood ratio consists in the different bounds for the second sum in the second term. Everything else is identical. Here we get as a **soft learning vector quantization** scheme the online gradient descent update rule $\forall i; 1 \leq i \leq c$: $\forall j; 1 \leq j \leq n$:

$$\begin{aligned}
\vec{\mu}_i^{(t+1)} &= \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}} \cdot \nabla_{\vec{\mu}_i} \ln L_{\text{ratio}}^{(1)}(\mathbf{X}, \vec{z}; \mathbf{C}^{(t)}, \mathbf{W}) \\
&= \vec{\mu}_i^{(t)} + \eta_{\vec{\mu}} \cdot \begin{cases} \left(u_{ij}^{\oplus(t)} - u_{ij}^{(t)}\right) \cdot \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right), & \text{if } z_j = \zeta_i, \\ -u_{ij}^{(t)} \cdot \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right), & \text{if } z_j \neq \zeta_i. \end{cases}
\end{aligned}$$

(Note that this time the reference vectors having the correct class appear in both terms of the likelihood ratio, which explains the sum $u_{ij}^{\oplus} - u_{ij}$ in the first case. Also recall that ζ_i is the class associated with the i-th reference vector.) The "membership degrees" $u_{ij}^{\oplus}$ and u_{ij} are

$$u_{ij}^{\oplus(t)} = \frac{\exp\left(-\frac{1}{2\sigma^2}\left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)^\top \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)\right)}{\sum_{y \in I(z_j)} \exp\left(-\frac{1}{2\sigma^2}\left(\vec{x}_j - \vec{\mu}_y^{(t)}\right)^\top \left(\vec{x}_j - \vec{\mu}_y^{(t)}\right)\right)} \quad \text{and}$$

$$u_{ij}^{(t)} = \frac{\exp\left(-\frac{1}{2\sigma^2}\left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)^\top \left(\vec{x}_j - \vec{\mu}_i^{(t)}\right)\right)}{\sum_{y \in \{1,\ldots,c\}} \exp\left(-\frac{1}{2\sigma^2}\left(\vec{x}_j - \vec{\mu}_y^{(t)}\right)^\top \left(\vec{x}_j - \vec{\mu}_y^{(t)}\right)\right)}.$$

Hard or **crisp learning vector quantization** can again be derived from this scheme in the limit $\sigma \to 0$, that is, for vanishing cluster radii. We get

$$u_{ij}^{\oplus} = \delta_{i,k^{\oplus}(j)}, \quad \text{where} \quad k^{\oplus}(j) = \operatorname*{argmin}_{l \in I(z_j)} d(\vec{x}_j, \vec{\mu}_l), \quad \text{and}$$
$$u_{ij} = \delta_{i,k(j)}, \quad \text{where} \quad k(j) = \operatorname*{argmin}_{l \in \{1,\ldots,c\}} d(\vec{x}_j, \vec{\mu}_l).$$

This update rule is very similar to, but again slightly different from the rule of [Kohonen 1990, Kohonen 1995]. Intuitively, it can be interpreted as follows: if the closest reference vector has the same class as the data point, no update is performed. On the other hand, if the class of the closest reference vector differs from the class of the data point, this closest vector is repelled, while the closest vector having the same class is attracted. In other words: an update is performed only if the data point is misclassified by a nearest prototype classifier based on the current positions of the reference vectors. Otherwise the positions of the reference vectors are kept.

Although this is clearly an improvement, as no window rule is needed (since the second likelihood ratio is bounded, see Section 3.4), one has to admit that this approach optimizes classification boundaries rather than captures the distribution of the data. As a consequence it behaves similar to *support vector machines* (cf. Section 3.5), where it is also unlikely that the resulting support vectors are representative of the data distribution. Although experiments by [Seo and Obermayer 2003] show that it performs better than Kohonen's version (which also tries to optimize classification boundaries), I rather prefer the classical scheme in which only one reference vector is adapted. Although it may lead to worse classification accuracy, it better serves the purpose to capture the data distribution.

5.4 Guided Random Search

Apart from the specific update methods considered in the preceding sections, a large class of heuristics, but very general optimization methods needs at least a brief consideration here. I like to call the methods in this class *guided random search methods*, because all of them employ chance (in the form of random modifications of solution candidates), but not in a blind manner, but guided by an objective function. These methods are very generally applicable, because they do not presuppose a model for the optimization process, but are defined only by a very general scheme for the optimization. As a consequence, however, they are usually less efficient and less effective than methods that are particularly geared to a specific problem. Nevertheless considering them can be worthwhile even in such cases.

5.4.1 Simulated Annealing

One of the best known methods in this direction is *simulated annealing* [Metropolis *et al.* 1953, Kirkpatrick *et al.* 1983]. The core idea underlying it is to start with a randomly generated candidate solution, which is evaluated. Then this candidate solution is modified randomly and the resulting new candidate solution is evaluated. If the new candidate solution is better than the original one, it is accepted and replaces the original one. If it worse, it is accepted only with a certain probability that depends on how much worse the new candidate solution is. In addition, this probability is lowered in the course of time, so that eventually only those new candidate solutions are accepted that are better than the current. Often the best solution found so far is recorded in parallel, in order to avoid losing good solutions again.

The reason for accepting a new candidate solution even though it is worse than the current is that without doing so the approach would be very similar to a gradient ascent (or descent, cf. Section 5.1). The only difference is that the direction of the gradient of the solution quality is not computed, but that the upward (or downward) direction is searched for by trial and error. However, it is well known that a gradient-based approach can easily get stuck in a local optimum. By accepting worse candidate solutions at the beginning of the process it is tried to overcome this undesired behavior. Intuitively, accepting worse candidate solutions makes it possible to cross the "barriers" that separate local optima from the global one, i.e., regions of the search space where the quality of the candidate solutions is worse. Later, however, when the probability for accepting worse candidate solutions is lowered, the objective function is optimized locally.

The name "simulated annealing" for this approach stems from the fact that it is similar to the physical minimization of the energy function (to be more precise: the atom lattice energy) when a heated piece of metal is cooled down very slowly. This process is usually called "annealing" and is used to soften a metal, relieve internal stresses and instabilities, and thus make it easier to work or machine. Physically, the thermal activity of the atoms prevents them from settling in a configuration that may be only a local minimum of the energy function. They "jump out" of this configuration. Of course, the "deeper" the (local) energy minimum, the harder it is for the atoms to "jump out" of the configuration. Hence, by this process they are likely to settle in a configuration of very low energy, the optimum of which is, in the case of a metal, a monocrystalline structure. It is clear, though, that it cannot be guaranteed that the global minimum of the energy function is reached. Especially if the piece of metal is not heated long enough, the atoms are likely to settle in a configuration that is only a local minimum (a polycrystalline structure in the case of a metal). Hence it is important to lower the temperature very slowly, so that there is a high probability that local minima, once reached, are left again.

This energy minimization process can easily be visualized by imagining a ball rolling on a curved landscape [Nauck *et al.* 1997]. The function to be minimized is the potential energy of the ball. At the beginning the ball is endowed with a certain kinetic energy which enables it to "climb" slopes. But due to friction this kinetic energy is diminished in the course of time and finally the ball will come to rest in a valley (a minimum of the objective function). Since it takes a higher kinetic energy to roll out of a deep valley than out of a shallow one, the final resting point is likely to be in a rather deep valley and maybe in the deepest one around (the global minimum).

Obviously, the thermal energy of the atoms in the annealing process or the kinetic energy of the ball in the illustration is modeled by the decreasing probability for accepting a worse candidate solution. Often an explicit temperature parameter is introduced, from which the probability (parameterized by how much worse the new candidate solution is) is computed. Since the probability distribution of the velocities of atoms is often an exponential distribution (cf., for example, the Maxwell distribution, which describes the velocity distribution for an ideal gas [Greiner *et al.* 1987]), a function like $P(\text{accept}) = ce^{-\frac{dQ}{T}}$ is frequently used to compute the probability for accepting a worse solution. Here dQ is the quality difference of the current and the new candidate solution, T is the temperature parameter and c is a normalization constant.

5.4.2 Genetic or Evolutionary Algorithms

The idea of *genetic* or *evolutionary algorithms* in their various forms [Goldberg 1989, Bäck *et al.* 1991, Koza 1992, Michalewicz 1996, Nilsson 1998, Mitchell 1998, Whitley 2001], is to employ an analog of biological evolution [Darwin 1859, Dawkins 1976, Dawkins 1987] to optimize a given function. In this approach the candidate solutions are coded into *chromosomes* with individual *genes* representing the components of a candidate solution. Most often such chromosomes are simple bit strings or vectors of real numbers, which hold the parameters of the candidate solution.

A genetic or evolutionary algorithm starts by generating a random initial *population* of individuals, each with its own chromosome. These individuals—or, to be more precise, the candidate solutions represented by their chromosomes[16]—are evaluated by a *fitness function*, which is the function to be optimized (or derived from it).

From the initial population a new population is generated by two means: The first is a simple *selection* process. A certain number of individuals is selected at random, with the probability that a given individual gets selected depending on its fitness. A simple method to achieve such a selection behavior is *tournament selection*: A certain number of individuals is picked at random from the population and the one with the highest fitness among them (the "winner of the tournament") is selected. It is clear that with this selection method individuals with a high fitness have a better chance to be passed into the new population than those with a low fitness and thus

[16] As in biology one may distinguish between the *genotype* of a living being, which is its genetic constitution, and its *phenotype*, which denotes its physical appearance or, in the context of genetic algorithms, the represented candidate solution.

only the fittest individuals of a population "survive", illustrating the (somewhat simplistic) characterization of biological evolution as the *survival of the fittest*. Of course, the individuals are also randomly modified from time to time (as in simulated annealing), thus imitating *mutation*, which in living beings occurs due to errors in the copying process chromosomes undergo.

The second process that is involved in generating the new population imitates *sexual reproduction*. Two "parent" individuals are chosen from the population, again with a probability depending on their fitness (for example, using tournament selection). Then their chromosomes are *crossed over* in order to obtain two new individuals that differ from both "parents".[17] A very simple method to do so is to fix a breakage point on the chromosomes and then to exchange one of the parts (so-called *one-point crossover*).

The idea underlying the crossing-over of chromosomes is that each of the "parent" chromosomes may already describe a good partial solution, which accounts for their high fitness (recall that the "parents" are selected with a probability depending on their fitness, so individuals with a high fitness are more likely to become "parents"). By crossing-over their chromosomes there is a good chance that these partial solutions are combined and that consequently an "offspring" chromosome is better than both of the "parents". This plausible argument is made formally more precise by the *schema theorem* [Michalewicz 1996]. It explains why evolution is much faster with sexual reproduction than without it (i.e., with mutation being the only mechanism by which genetically new individuals can emerge).

Of course, the new population is then taken as a starting point for generating the next and so on, until a certain number of generations has been created or the fitness of the best member of the population has not increased in the last few generations. The result of a genetic algorithm is the fittest individual of the final generation or the fittest individual that emerged during the generations (if it is kept track of).

There are several variants of genetic or evolutionary algorithms, depending on whether only "offspring" is allowed into the next population or whether "parents" are passed too, whether the population is processed as a whole or split into subgroups with "mating" occurring only within subgroups and only rare "migrations" of individuals from one subpopulation to another etc. [Michalewicz 1996].

[17] The term *crossing-over* was chosen in analogy to the biological process with the same name in which genetic material is exchanged between (homologous) chromosomes by breakage and reunion. This process happens during *meiosis* (reduction division), i.e, the division of (homologous) chromosome pairs so that each *gamete* (a sex cell, e.g., an egg) receives one chromosome.

5.4.3 Application to Classification and Clustering

Applying a form of guided random search to clustering or classification tasks is straightforward: the space of cluster parameters in $\mathbf{C}$, combined with the space of weighting parameters in $\mathbf{W}$ if the task is classification, is the search space. That these parameters can easily be written as a vector of real numbers makes it possible to employ or easily adapt standard implementations of genetic algorithms and simulated annealing. The fitness function is, of course, one of the functions defined in Chapter 3. Adding a vector of normally distributed random numbers may be used to implement mutation or to produce a variant of the current solution in simulated annealing. It is plausible that the crossover operation should be chosen in such a way that there is a tendency that the parameters of a cluster stay together, so that it mainly produces recombinations of clusters rather than of arbitrary subsets of cluster parameters. If the task is classification, clusters should also keep their associated weighting parameters.

Details about simulated annealing approaches to clustering can be found, for example, in [Rose 1998]. Approaches that are based on genetic algorithms are more numerous, possibly because it is hoped that the lower efficiency of genetic algorithms in approaching an optimum will be outweighed by the parallel search with several solution candidates, which provides better chances to avoid getting stuck in a local optimum. Examples of such approaches, which cover a large variety of aspects (including determining the number of clusters), can be found in [Babu and Murty 1994, Bezdek *et al.* 1994, Hall *et al.* 1994, Petry *et al.* 1994, Van Le 1995, Nascimiento and Moura-Pires 1997, Klawonn and Keller 1998, Hall *et al.* 1999, Gerdes *et al.* 2004]. Here I skip a more detailed consideration of these approaches, mainly because of the results of [Hall *et al.* 1999] (as cited in [Gerdes *et al.* 2004]), who found in hard and fuzzy clustering experiments that on the tested data sets a genetic algorithm based approach never found a better optimum than alternating optimization, which was carried out several times with different initializations. Only equally good optima could be found if the population and the number of computed generations was large enough. However, to find these optima the execution time of the genetic algorithm based approach was, on most data sets, about two orders of magnitude greater than that of the standard alternating optimization scheme.

As a consequence, guided random search techniques cannot be recommended for standard clustering tasks. The same holds for classification, as the same arguments apply: simulated annealing and genetic algorithms are simply much slower in approaching an optimum and the problem of get-

ting stuck in local optima can be more efficiently solved by simply running gradient descent, alternating optimization, or learning vector quantization several times with different initializations.

Using guided random search can be useful only if, due to a particular application-specific objective function, the standard approaches are not applicable. Such an application area seems to be semi-supervised classification [Klose 2004], in which two objective functions—one for classification and one for clustering—have to be combined. Such a combination can render the standard numerical techniques infeasible, while approaches based on genetic algorithms are easily applicable. Another application area is using fuzzy clustering for learning fuzzy rules [Klawonn and Keller 1998, Gerdes *et al.* 2004], where (depending on the type of fuzzy system) a distance measure is used that is not continuously differentiable, so that the standard numerical techniques fail.

Chapter 6

Update Modifications

The update methods as they were studied in the preceding chapter already provide a complete prescription for updating the parameters of a cluster model or a classifier. Hence, at first sight, nothing else seems to be needed for the learning process. However, there are several situations in which it is useful to modify these update rules in order to achieve the goals of either a higher *robustness* of the method or to *accelerate* the training process. In the two sections of this chapter I study such modifications, dealing with the former goal in Section 6.1 and with the latter in Section 6.2.

Methods to enhance robustness mainly deal with the effects of outliers or specifically skewed data distributions that can deteriorate the results. In this connection I study noise clustering [Ohashi 1984, Davé 1991, Davé and Krishnapuram 1997, Davé and Sen 1997] as well as regularization methods for the shape, size, and weight of the clusters [Borgelt and Kruse 2004, Borgelt and Kruse 2005]. Methods to enhance the learning speed, mainly by reducing the number of iterations that are needed until convergence, are based on a transfer of techniques that were developed for training neural networks with gradient descent [Borgelt and Kruse 2003].

Simple forms of modifications that are closely related to those examined here were already considered in the preceding chapter. Examples are the "repair" of results obtained by gradient descent, the normalization of covariance matrices to determinant 1, or the choice of the learning rate in gradient descent. Here, however, they are studied in a more principled and more general way. In addition, most of these methods are not limited to a specific update method, but can be combined with any of the methods presented in the preceding chapter. Therefore it is useful to discuss them separately.

6.1 Robustness

Data points that lie far away from the majority of the data—so-called **outliers**—are a problem for most clustering approaches. This problem is particularly severe if they are based on minimizing the sum of squared distances from the cluster centers, and even more so if hard clustering is employed. The reason is that they strongly affect the clusters they are assigned to, since they have a large squared distance to them (regardless of which cluster they are assigned to). Thus reducing this distance by moving a cluster center closer to them can outweigh the increase in the squared distance to several other data points. In order to deal with this problem, *noise clustering* has been introduced, which is reviewed in Section 6.1.1.

Methods for both classification and clustering can be negatively affected by skewed data distributions if size and shape parameters are used for the cluster prototypes. Among the effects that can occur in such situations are that clusters are drawn into very long and thin (hyper-)ellipsoids or that they (almost) collapse to radius 0, focusing on very few or even a single data point. Depending on the clustering or classification method used, these effects may even occur for fairly "normal" data distributions, simply because the method reacts (too) sensitively to certain properties of the data set. In order to handle these problems, I introduce *regularization methods* for the shape, size, and weight parameters of the clusters, which cause a tendency towards spherical shape, equal size, and equal weight of the clusters. The strength of this tendency can be controlled with parameters, yielding very flexible methods. These methods are presented in Sections 6.1.2 to 6.1.4.

6.1.1 Noise Clustering

The idea of noise clustering [Ohashi 1984, Davé 1991, Davé and Krishnapuram 1997, Davé and Sen 1997], which is a modification of fuzzy clustering, is very simple: In addition to the cluster prototypes, the parameters of which are adapted, one introduces a so-called **noise cluster**, which has the same distance to every data point. This cluster is *not* described by a prototype and hence no parameters are adapted. Throughout the update process it always has the same distance to every data point, regardless of the locations of the other clusters. Its distance to the data points (or, alternatively, the (unnormalized) membership degree every data point has to it) is specified by the user. The smaller the distance (or the larger the membership degree), the more pronouncedly outliers are assigned to the noise cluster. Standard fuzzy clustering results in the limit for an infinite distance.

To illustrate the effect of introducing a noise cluster, Figures 6.1 to 6.3 show the results of fuzzy c-means clustering for a very simple data set. In Figure 6.1 there is no outlier, so the clustering method is successful without effort, yielding two clusters in symmetric positions. In Figure 6.2 another data point has been added to the data set. It is clearly an outlier, since it is far away from both groups of data points (at the top of the diagram). Intuitively, one would say that it should not be considered when forming the clusters. However, since it has the same distance to both clusters, it is assigned with a degree of membership of 0.5 to both of them and thus drags the cluster centers away from their natural positions. This effect would be even worse for hard c-means clustering, since in this case the outlier would be assigned exclusively to one cluster, affecting it even more. This deteriorating effect can effectively be removed with noise clustering as can be seen in Figure 6.3, where it was specified that every data point should have an (unnormalized) membership degree of 0.2 to the noise cluster. Due to this the outlier is assigned almost exclusively to the noise cluster, and as a consequence the other two clusters are almost unaffected by the outlier.

Note, however, that the membership to the noise cluster has to be chosen with care. In the example, the two clusters are still slightly affected by the outlier if the (unnormalized) membership degree to the noise cluster is set to 0.1. On the other hand, if the membership degree to the noise cluster is set too large, data points that can be assigned well to a cluster, although they are only on its rim, already get assigned to the noise cluster and thus are (partially) lost for estimating the cluster parameters. A good strategy is to check the sum of (normalized) membership degrees to the noise cluster in a clustering result. If this sum considerably exceeds the percentage of the data set that may be outliers, the (unnormalized) membership degree to the noise cluster should be reduced (the distance to the noise cluster should be increased) and the clustering should be repeated. Several runs may be necessary until an appropriate value for the noise cluster parameter (i.e. distance of membership degree) is found.

Note also that noise clustering removes the undesired effect of a normalization to sum 1 in connection with a Cauchy radial function as it was studied in Section 2.4 (cf. page 29ff), namely that the membership degrees to the different clusters tend to become equal far away from the cluster centers. Since the distance (and thus the (unnormalized) degree of membership) to the noise cluster is fixed, the (normalized) membership to the noise cluster is not subject to this effect. Rather the (normalized) membership degree to the noise cluster becomes larger and larger and approaches 1 in the limit, which is exactly why it captures outliers effectively.

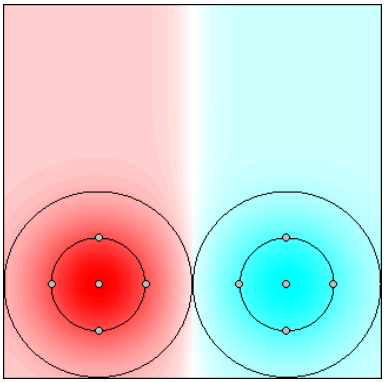

Figure 6.1: Fuzzy c-means clustering of a very simple data set without outliers. The structure of the data points is captured well.

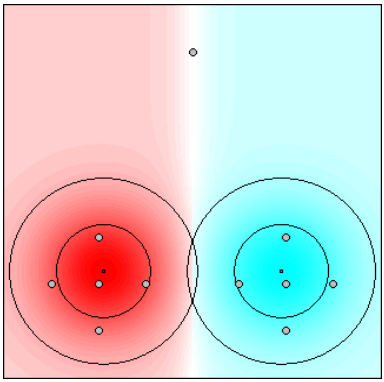

Figure 6.2: Fuzzy c-means clustering of the same data set as in Figure 6.1, but with an additional data point that is fairly far away from both clusters (the single data point at the top of the picture). This outlier has a harmful effect as it drags the cluster centers away from their natural positions.

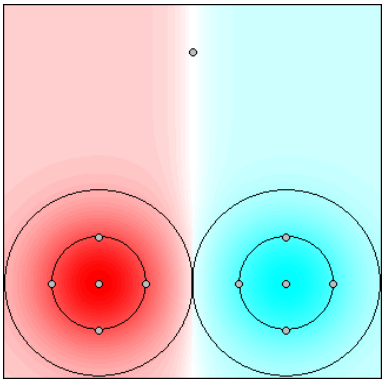

Figure 6.3: Fuzzy c-means clustering of the same data set as in Figure 6.1, but with noise clustering, using an (unnormalized) membership degree of 0.2 to the noise cluster. The deteriorating effect of the outlier is effectively removed. With an (unnormalized) membership degree of 0.1 a slight change of the cluster centers can still be observed.

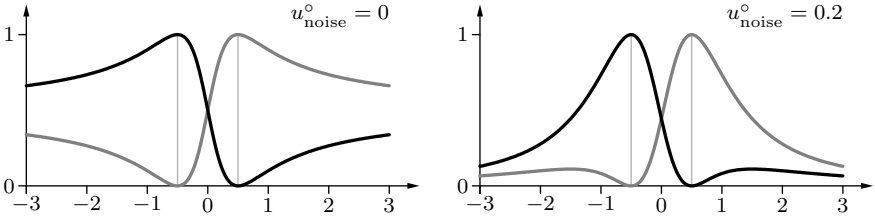

Figure 6.4: Membership degrees normalized to sum 1 for the generalized Cauchy function ($a = 2$, $b = 0$) for two cluster centers at -0.5 and 0.5. Left: without a noise cluster, right: with a noise cluster to which every point has an unnormalized degree of membership of 0.2.

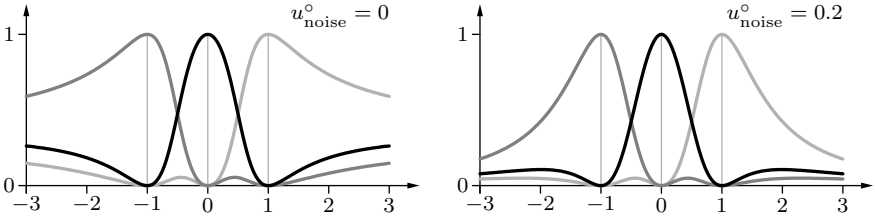

Figure 6.5: Membership degrees normalized to sum 1 for the generalized Cauchy function ($a = 2$, $b = 0$) for three cluster centers at -1, 0, and 1. Left: without a noise cluster, right: with a noise cluster to which every point has an unnormalized degree of membership of 0.2.

This is illustrated in Figures 6.4 and 6.5 for two and three cluster centers, respectively. The left diagram shows the membership degrees for a Cauchy function with $a = 2$ and $b = 0$ (that is, for standard fuzzy clustering) after a normalization to sum 1 if no noise cluster is used (cf. Figures 2.12 and 2.18 on pages 30 and 34, respectively). The right diagrams show the corresponding (normalized) membership degrees if a noise cluster is introduced, to which every point has an (unnormalized) membership degree of 0.2. It is clearly visible that the membership degrees to the two clusters approach zero in the limit for a large distance to the cluster centers, because the (normalized) membership degree to the noise cluster approaches 1. Although directly beyond the other cluster center the membership degree to a cluster increases, which is still slightly unintuitive, this effect is limited to rather small membership degrees and a fairly small region.

In principle, the idea of noise clustering may also be applied with probability based approaches like the expectation maximization algorithm. In this case the noise cluster is described formally by a uniform distribution on the data space. Of course, the corresponding density function can be proper (that is, can only have integral 1) if it is restricted to a finite subspace. However, this technical problem can easily be handled by limiting the noise cluster to an appropriate (hyper-)box enclosing all points in the given data set. To parameterize the approach in this case it is most convenient to specify the probability of observing a data point that is generated by the noise cluster, that is, the prior probability of the noise cluster times the probability density inside the (hyper-)box chosen for the uniform distribution. (Note that these two parameters cannot be separated well anyway, since a higher prior probability can be compensated by a larger region for the uniform distribution, so it is best to specify their product directly.)

A more sophisticated approach to deal with outliers, which is related to noise clustering, has been suggested in [Keller 2000, Keller 2002]. It introduces a weight for each data point that is meant to describe the degree to which the data point is an outlier. These weights are adapted during the clustering process. Formally, the approach is based on the objective function

$$J(\mathbf{X}, \mathbf{U}, \mathbf{C}) = \sum_{i=1}^{c} \sum_{j=1}^{n} \frac{u_{ij}^{w}}{\omega_j^{\nu}} d_{ij}^2,$$

where the ω_j are the data point weights and ν is a parameter by which the influence of the weights can be controlled. In addition, a value ω has to be chosen by a user, which is used in the constraint

$$\sum_{j=1}^{n} \omega_j = \omega.$$

Based on these definitions, the update rules for the cluster parameters and the data point weights can easily be derived in the same way as it was done in Chapter 5 for the corresponding cases without data point weights. For an alternating optimization approach the membership degrees u_{ij} are computed in exactly the same way as for the standard algorithm, while the update rule for the data point weights is $\forall j; 1 \leq j \leq n$:

$$\omega_j^{(t+1)} = \frac{\left(\sum_{i=1}^{c} \left(u_{ij}^{(t)} \right)^w \left(d_{ij}^{(t)} \right)^2 \right)^{\frac{1}{\nu+1}}}{\sum_{k=1}^{n} \left(\sum_{i=1}^{c} \left(u_{ij}^{(t)} \right)^w \left(d_{ij}^{(t)} \right)^2 \right)^{\frac{1}{\nu+1}}}.$$

From this update rule it is easy to see that in the limit for $\nu \to \infty$ all weighting factors for the data points approach $\frac{\omega}{n}$, so that we obtain the standard approach for the choice $\omega = n$. On the other hand, for $\nu \to 0$ the influence of the weighting factors becomes maximal.

Note that for the update of the cluster parameters (centers and covariance matrices) the membership degrees and the weighting factors can simply be combined into quantities $\tilde{u}_{ij} = \frac{u_{ij}}{\omega_j}$, $1 \leq i \leq c$, $1 \leq j \leq n$, because they are both considered fixed in this update step. These quantities simply replace the u_{ij} in the standard update rules.

As an alternative to noise clustering, possibilistic fuzzy clustering is also often advocated as a means to treat data sets containing a non-negligible amount of outliers. An argument supporting this is that in possibilistic clustering the degree of membership depends only on the distance to a cluster, and since there is no normalization, a data point far away from all clusters (i.e. an outlier) will have a small degree of membership to all clusters. As a consequence, the influence of such an outlier on the clustering result is limited. However, although this argument is plausible, the detailed discussion in Section 3.1 (cf. page 53ff) should have made it clear that due to its severe drawbacks possibilistic clustering is not a recommendable method. Therefore noise clustering should be preferred.

Other approaches to handle outliers in clustering, in particular fuzzy clustering, which I do not discuss in detail here, can be found in [Davé and Krishnapuram 1997, Davé and Sen 1997, Kharin 1997].

6.1.2 Shape Regularization

The large number of parameters (mainly the elements of the covariance matrices) of the more flexible clustering algorithms as well as of classification algorithms allowing for cluster size and shape adaptation can render these algorithms less robust or even fairly unstable, compared to their simpler counterparts that only adapt the cluster centers. Common undesired results include very long and thin ellipsoids as well as clusters collapsing to very few or even a single data point. To counteract such undesired tendencies, I introduce shape and size constraints into the update scheme [Borgelt and Kruse 2004, Borgelt and Kruse 2005]. The basic idea is to modify, in every update step, the cluster parameters in such a way that certain constraints are satisfied or at least that a noticeable tendency (of varying strength, as specified by a user) towards satisfying these constraints is introduced. In particular I consider regularizing the (ellipsoidal) shape as well as constraining the (relative) size and the (relative) weight of a cluster.

In accordance with the common usage of this term, I call all of these approaches **regularization methods**. The origin of this term is, however, actually in linear optimization, in which ill-conditioned matrices describing a linear equation system are modified in such a way that they are "less singular" and thus "more regular" [Tikhonov and Arsenin 1977, Engl *et al.* 1996]. (Recall that an ill-conditioned matrix may behave numerically like a singular one in computations on a finite precision machine—cf. Section A.5 in the appendix for more explanations.) However, over the years the term "regularization" has acquired a more general meaning and is used now whenever a tendency towards certain desired properties (or, alternatively, a penalty against certain undesired properties) of the solution of an optimization problem is introduced. Hence it is appropriate not only to speak of *shape regularization* (which, as will become clear below, is actually based on a regularization approach in the original sense), but also of shape and weight regularization (cf. Sections 6.1.3 and 6.1.4, respectively).

This section deals with **shape regularization**. As discussed in Section 2.3, the shape of a cluster is represented by its covariance matrix $\mathbf{\Sigma}_i$. Intuitively, $\mathbf{\Sigma}_i$ describes a general (hyper-)ellipsoidal shape, which can be obtained, for example, by computing the Cholesky decomposition or the eigenvalue decomposition of $\mathbf{\Sigma}_i$ and mapping the unit (hyper-)sphere with it (cf. Sections A.3 and A.4, respectively, in the appendix).

Shape regularization means to modify the covariance matrix, so that a certain (user-specified) relation of the lengths of the major axes of the represented (hyper-)ellipsoid is obtained or that at least a tendency towards this relation is introduced. Since the lengths of the major axes are the roots of the eigenvalues of the covariance matrix (cf. Section A.4 in the appendix), regularizing it means *shifting the eigenvalues* of $\mathbf{\Sigma}_i$. Note that such a shift leaves the *eigenvectors* unchanged, i.e., the orientation of the represented (hyper-)ellipsoid is preserved (cf. Section A.6 in the appendix). Note also that such a shift of the eigenvalues is the basis of the Tikhonov regularization of linear optimization problems [Tikhonov and Arsenin 1977, Engl *et al.* 1996], which inspired my approach. I suggest two shape regularization methods [Borgelt and Kruse 2004, Borgelt and Kruse 2005]:

Method 1: The covariance matrices $\mathbf{\Sigma}_i$, $i = 1, \ldots, c$, are adapted (in every update step) according to

$$\mathbf{\Sigma}_i^{(\text{adap})} = \sigma_i^2 \cdot \frac{\mathbf{S}_i + h^2\mathbf{1}}{\sqrt[m]{|\mathbf{S}_i + h^2\mathbf{1}|}} = \sigma_i^2 \cdot \frac{\mathbf{\Sigma}_i + \sigma_i^2 h^2\mathbf{1}}{\sqrt[m]{|\mathbf{\Sigma}_i + \sigma_i^2 h^2\mathbf{1}|}},$$

where m is the dimension of the data space, $\mathbf{1}$ is a unit matrix, $\sigma_i^2 = \sqrt[m]{|\mathbf{\Sigma}_i|}$

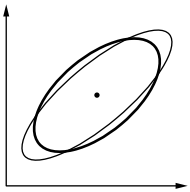

Figure 6.6: The effect of shape regularization: the orientation is left unchanged, but a tendency towards spherical shape is introduced (thin ellipse: before regularization, thick ellipse: after regularization).

is the equivalent isotropic variance (equivalent in the sense that it leads to the same (hyper-)volume, i.e., $|\mathbf{\Sigma}_i| = |\sigma_i^2 \mathbf{1}|$), $\mathbf{S}_i = \sigma_i^{-2}\mathbf{\Sigma}_i$ is the covariance matrix scaled to determinant 1, and h is the regularization parameter.

This modification of the covariance matrix shifts all eigenvalues by the value of $\sigma_i^2 h^2$ and then renormalizes the resulting matrix so that the determinant of the old covariance matrix is preserved (i.e., the (hyper-)volume is kept constant). This regularization tends to equalize the lengths of the major axes of the represented (hyper-)ellipsoid and thus introduces a tendency towards (hyper-)spherical clusters. This tendency is the stronger, the greater the value of h. In the limit, for $h \to \infty$, the clusters are forced to be exactly spherical; for $h = 0$ the shape is left unchanged. An illustration of the effect is shown in Figure 6.6: the thin ellipse shows the situation before the shape regularization, the thick ellipse the situation afterwards.

Method 2: The above method always changes the length ratios of the major axes and thus introduces a general tendency towards (hyper-)spherical clusters. In this (second) method, however, a limit r, $r > 1$, for the length ratio of the longest to the shortest major axis of the represented (hyper-) ellipsoid is used and only if this limit is exceeded, the eigenvalues are shifted in such a way that the limit is satisfied. Formally: let λ_k, $k = 1, \ldots m$, be the eigenvalues of the matrix $\mathbf{\Sigma}_i$. Set (in every update step)

$$h^2 = \begin{cases} 0, & \text{if } \dfrac{\max_{k=1}^{m} \lambda_k}{\min_{k=1}^{m} \lambda_k} \leq r^2, \\ \dfrac{\max_{k=1}^{m} \lambda_k - r^2 \min_{k=1}^{m} \lambda_k}{\sigma_i^2 (r^2 - 1)}, & \text{otherwise,} \end{cases}$$

and execute Method 1 with this value of h^2. This method also has the advantage that it is much easier to choose r than to choose h.

Note that both methods are heuristics in the sense that they cannot be justified, for example, by setting up a (modified) objective function from which they can be derived. Nevertheless they turn out to be very useful in practice (cf. Section 8.1 for experimental results).

6.1.3 Size Regularization

As explained in Section 2.3, the size of a cluster can be described in different ways, for example, by the determinant of its covariance matrix $\mathbf{\Sigma}_i$, which is a measure of the clusters squared (hyper-)volume, by an equivalent isotropic variance σ_i^2 or by an equivalent isotropic radius (standard deviation) σ_i (equivalent in the sense that they lead to the same (hyper-)volume). The latter two measures are defined as $\sigma_i^2 = \sqrt[m]{|\mathbf{\Sigma}_i|}$ and $\sigma_i = \sqrt{\sigma_i^2} = \sqrt[2m]{|\mathbf{\Sigma}_i|}$ and thus the (hyper-)volume may also be written as $\sigma_i^m = \sqrt{|\mathbf{\Sigma}_i|}$.

Size regularization means to ensure a certain relation between the cluster sizes or at least to introduce a tendency into this direction. I suggest three different versions of size regularization, in each of which the measure that is used to describe the cluster size is specified by an exponent κ of the equivalent isotropic radius σ_i, with the special cases (cf. Section 2.3):

$$\begin{aligned} \kappa = 1: &\quad \text{equivalent isotropic radius,} \\ \kappa = 2: &\quad \text{equivalent isotropic variance,} \\ \kappa = m: &\quad \text{(hyper-)volume.} \end{aligned}$$

For regularizing of the (relative) cluster sizes I consider three methods [Borgelt and Kruse 2004, Borgelt and Kruse 2005], with the first and the third being analogous to the shape regularization methods presented in the preceding section. The second method is a simplification with the goal to make the computations more efficient.

Method 1: The equivalent isotropic radii σ_i are adapted (in every update step) according to

$$\begin{aligned} \sigma_i^{(\text{adap})} &= \sqrt[\kappa]{\nu \cdot \frac{\sum_{k=1}^{c} \sigma_k^\kappa}{\sum_{k=1}^{c} (\sigma_k^\kappa + o)} \cdot (\sigma_i^\kappa + o)} \\ &= \sqrt[\kappa]{\nu \cdot \frac{\sum_{k=1}^{c} \sigma_k^\kappa}{c \cdot o + \sum_{k=1}^{c} \sigma_k^\kappa} \cdot (\sigma_i^\kappa + o)}. \end{aligned}$$

That is, each cluster size is increased by the value of the regularization parameter o and then the sizes are renormalized so that the sum of the cluster sizes is preserved. However, the parameter ν may be used to scale the sum of the sizes up or down (by default $\nu = 1$). For $o \to \infty$ the cluster sizes are equalized completely, for $o = 0$ only the parameter ν has an effect. This method is inspired by Laplace correction or Bayesian estimation with an uninformative prior (cf. also Section 6.1.4).

Method 2: This (second) method, which is meant as a simplified and thus more efficient version of method 1, does not renormalize the sizes, so that the size sum increases by $c \cdot o$. However, this missing renormalization may be mitigated to some degree by specifying a value of the scaling parameter ν that is smaller than 1. The equivalent isotropic radii σ_i are adapted (in every update step) according to

$$\sigma_i^{(\text{adap})} = \sqrt[\kappa]{\nu \cdot (\sigma_i^\kappa + o)}.$$

Method 3: The above methods always change the relation of the cluster sizes and thus introduce a general tendency towards clusters of equal size. In this (third) method, however, a limit r, $r > 1$, for the size ratio of the largest to the smallest cluster is used and only if this limit is exceeded, the sizes are changed in such a way that the limit is satisfied. To achieve this, o is set (in every update step) according to

$$o = \begin{cases} 0, & \text{if } \frac{\max_{k=1}^{c} \sigma_k^\kappa}{\min_{k=1}^{c} \sigma_k^\kappa} \leq r, \\ \frac{\max_{k=1}^{c} \sigma_k^\kappa - r \min_{k=1}^{c} \sigma_k^\kappa}{r-1}, & \text{otherwise,} \end{cases}$$

and then Method 1 is executed with this value of o. Similar to the second method for shape regularization, this method also has the advantage that it is much easier to choose an appropriate value for r than it is to choose an appropriate value for o.

Note again (as in Section 6.1.2) that all methods are heuristics in the sense that they cannot be justified, for example, by setting up a (modified) objective function from which they can be derived. Nevertheless they yield useful results in practice (cf. Section 8.1 for experimental results).

6.1.4 Weight Regularization

A cluster weight ϱ_i only appears in the mixture model approach and the fuzzy maximum likelihood (FMLE) algorithm, where it describes the prior probability of a cluster. For the cluster weight we may use basically the same regularization methods as for the cluster size. An exception is the scaling parameter ν, since the ϱ_i are probabilities, i.e., we must ensure $\sum_{i=1}^{c} \varrho_i = 1$. I suggest two methods, which are directly analogous to the two methods of shape regularization and to the first and third method of size regularization [Borgelt and Kruse 2004, Borgelt and Kruse 2005]:

Method 1: The cluster weights ϱ_i, $1 \leq i \leq c$, are adapted (in every update step) according to

$$\varrho_i^{(\text{adap})} = \frac{\sum_{k=1}^{c} \varrho_k}{\sum_{k=1}^{c} (\varrho_k + o)} \cdot (\varrho_i + o) = \frac{\sum_{k=1}^{c} \varrho_k}{c \cdot o + \sum_{k=1}^{c} \varrho_k} \cdot (\varrho_i + o),$$

where o is the regularization parameter that has to be specified by a user. Note that this method is equivalent to a Laplace corrected estimation of the prior probabilities or a Bayesian estimation with an uninformative (uniform) prior. Such an estimation is meant to ensure that all estimated probabilities are positive (non-vanishing). It is also used to introduce a tendency towards a uniform distribution, which is the most appropriate prior assumption if no background knowledge is available about the probabilities.

Method 2: The value of the regularization parameter o is computed as

$$o = \begin{cases} 0, & \text{if } \frac{\max_{k=1}^{c} \varrho_k}{\min_{k=1}^{c} \varrho_k} \leq r, \\ \frac{\max_{k=1}^{c} \varrho_k - r \min_{k=1}^{c} \varrho_k}{r - 1}, & \text{otherwise,} \end{cases}$$

with a user-specified maximum weight ratio r, $r > 1$, and then Method 1 is executed with this value of the regularization parameter o.

Note again (as in Sections 6.1.2 and 6.1.3) that all methods are heuristics in the sense that they cannot be justified, for example, by setting up a (modified) objective function from which they can be derived.

6.2 Acceleration

All update methods that were discussed in Chapter 5 are iterative in nature, because it is not possible to optimize any of the objective functions reviewed in Chapter 3 directly (expect for rare and practically irrelevant special cases). The time complexity of each iteration is usually, regardless of the update method used, $O(cnm)$ for the Euclidean distance and $O(cnm^2)$ for the Mahalanobis distance, where c is the number of clusters, n is the number of data points, and m is the number of dimensions of the data space. The reason is that the complexity of one iteration is governed by the computation of the distance/membership degree of the data points to the different cluster centers. Computing this distance for one data point and one cluster takes $O(m)$ operations for the Euclidean distance, because then it is basically the computation of the inner product of the difference

vector to the cluster center with itself. For a Mahalanobis distance the time complexity is $O(m^2)$, because the difference vector has to be multiplied with a covariance matrix.

Recomputing the cluster parameters can usually be neglected, since in most methods this recomputation is carried out only once per iteration. Its costs are $O(cm)$ for the Euclidean distance and $O(cm^3)$ for the Mahalanobis distance.[1] Therefore recomputing the cluster parameters does not change the asymptotic time complexity even for the Mahalanobis distance, because usually $m \ll n$. By an analogous argument we can neglect the time complexity for recomputing the weighting parameters w_{ik}, $1 \leq i \leq c$, $1 \leq k \leq s$, of a classifier, which has time complexity $O(cs)$, because usually $s \ll n$.

As a consequence, the total time complexity of building a cluster model or of constructing a classifier is $O(cnmT)$ or $O(cnm^2T)$, depending on whether the Euclidean or the Mahalanobis distance is used, where T is the number of iterations needed until convergence. Unfortunately, basically nothing is known about the number of iterations needed to reach the (local) optimum of the chosen objective function with some given error limit ε, regardless of the chosen update method. However, what can easily be seen is this: if we want to improve the execution time of the algorithms, trying to reduce the number T of iterations needed until convergence is the most promising line of attack. Alternatives are, of course, to improve the computations that are carried out in one iteration, but most of these improvements only change the constant that is hidden in the O-notation.

A fairly large variety of modifications that aim at reducing the number of update steps are known for gradient methods. Most of them were developed for training neural networks, in particular multilayer perceptrons (see, for example, [Zell 1994, Nauck *et al.* 2003]). The reason is that the so-called *backpropagation method* for training such neural networks is basically a gradient descent on the error function, very much in the same way as it was studied in Section 5.1.3 for a prototype-based classifier.

However, in principle these modifications are applicable together with any iterative update scheme. All one has to do is to replace, in their original form, the gradient with the difference between the new and the old parameters. This is the approach I suggested in [Borgelt and Kruse 2003], where I applied it to fuzzy clustering. An earlier, but less general investigation in the same direction (it is, for example, restricted to fuzzy c-means clustering) has been done by [Hershfinkel and Dinstein 1996].

[1] Note that the exponent 3 for the number m of dimensions in this complexity expression stems from the fact that the computed covariance matrix usually has to be inverted, an operation that has time complexity $O(m^3)$.

In the following sections I review several techniques for modifying the update steps in such a way that the number of iterations needed until convergence is potentially[2] reduced. I review them here w.r.t. gradient descent, because this is the area in which they were developed. However, the (negative) gradient should rather be seen as a kind of placeholder that may be replaced by a parameter change as it can be computed with other update methods (like, for example, alternating optimization). Experimental results with these methods for fuzzy clustering are presented in Section 8.2.

6.2.1 Step Expansion

In order to have a reference point, recall the update rule for basic gradient descent on an error function (that is, standard so-called *error backpropagation*). This update rule reads for an arbitrary parameter θ_i, $1 \leq i \leq k$:

$$\theta_i^{(t+1)} = \theta_i^{(t)} + \Delta\theta_i^{(t)} \qquad \text{where} \qquad \Delta\theta_i^{(t)} = -\eta \nabla_{\theta_i} e\big(\vec{\theta}^{(t)}\big).$$

Here t is the time step, e is the objective function (here: error function), which is to be minimized, and $\vec{\theta} = (\theta_1, \ldots, \theta_k)$ is the full set of parameters of this objective function. This update rule prescribes to compute the new value of the parameter θ_i (in step $t+1$) from its old value (in step t) by adding a change, which is computed from the gradient $\nabla_{\theta_i} e^{(t)}(\vec{\theta})$ of the error function w.r.t. the parameter θ_i. η is a learning rate that influences the size of the steps that are carried out. The minus sign results from the fact that the gradient points into the direction of the steepest *ascent*, but we want to carry out a gradient *descent*. Depending on the definition of the error, the gradient may also be preceded by a factor of $\frac{1}{2}$ in this formula, in order to cancel a factor of 2 that results from differentiating a squared error (cf. Section 5.1 for several concrete examples of such update rules).

Although this is the standard approach for gradient descent, it is (also) listed as a modification here, because the learning rate only appeared in the gradient descent approaches as they were studied in Section 5.1. However, if we use a different update method, we may still use a scheme like this. For example, we may compute the parameter change with alternating optimization and then multiply this change with a factor η that resembles the learning rate in gradient descent. Of course, in this case this factor should be greater than one, since $\eta = 1$ yields the standard approach and $\eta < 1$ is likely to slow it down. As a consequence I call this update modification *step expansion*: it expands the step in the parameter space.

[2] There is, of course, no guarantee that the methods actually achieve this goal.

6.2.2 Momentum Term

The momentum term method [Rumelhart *et al.* 1986] consists in adding a fraction of the parameter change of the previous step to a normal gradient descent step. The rule for changing the parameters thus becomes

$$\Delta\theta_i^{(t)} = -\eta\nabla_{\theta_i} e\big(\vec{\theta}^{(t)}\big) + \beta\, \Delta\theta_i^{(t-1)},$$

where β is a parameter, which must be smaller than 1 in order to make the method stable. In neural network training β is usually chosen between 0.5 and 0.95, but sometimes values even up to 0.99 can be useful.

The additional term $\beta\ \Delta\theta_i^{(t-1)}$ is called *momentum term*, because its effect corresponds to the momentum that is gained by a ball rolling down a slope. The longer the ball rolls in the same direction, the faster it gets. As a consequence it has a tendency to keep on moving in the same direction (this is modeled by the momentum term), but it also follows, though slightly retarded, the shape of the surface (this is modeled by the gradient term).

By adding a momentum term the learning process can be accelerated, especially in areas of the parameter space, in which the objective function is (almost) flat, but descends in a uniform direction. In this case larger and larger steps will be made in the direction opposite to the gradient. Adding a momentum term also mitigates the problem of how to choose the value of the learning rate, because the momentum terms enlarges or shrinks the step width depending on the shape of the error function. However, if the learning rate is far too small, the momentum term can not fully compensate for this, since the step width $|\Delta\theta_i|$ is bounded by

$$|\Delta\theta_i| \leq \left| \frac{\eta\nabla_{\theta_i} e(\vec{\theta})}{2(1-\beta)} \right|,$$

provided the gradient $\nabla_{\theta_i} e(\vec{\theta})$ is constant (in the current region of the parameter space). In addition, if the learning rate η is too large, it is still possible—actually even more likely than without a momentum term—that oscillations and chaotic behavior result (cf. Section 5.1.5).

6.2.3 Super Self-Adaptive Backpropagation

The idea of so-called (super) self-adaptive backpropagation (SuperSAB) [Jakobs 1988, Tollenaere 1990] is to introduce an individual learning rate η_w for each parameter of the objective function. These learning rates are then

adapted (before they are used in the current update step) according to the values of the current and the previous gradient. The exact adaptation rule for the update of the learning rates η_i, $1 \leq i \leq k$, is

$$\eta_i^{(t)} = \begin{cases} \beta_- \cdot \eta_i^{(t-1)}, & \text{if } \nabla_{\theta_i} e\big(\vec{\theta}^{(t)}\big) \cdot \nabla_{\theta_i} e\big(\vec{\theta}^{(t-1)}\big) < 0, \\ \beta_+ \cdot \eta_i^{(t-1)}, & \text{if } \nabla_{\theta_i} e\big(\vec{\theta}^{(t)}\big) \cdot \nabla_{\theta_i} e\big(\vec{\theta}^{(t-1)}\big) > 0, \\ & \wedge\ \nabla_{\theta_i} e\big(\vec{\theta}^{(t-1)}\big) \cdot \nabla_{\theta_i} e\big(\vec{\theta}^{(t-2)}\big) \geq 0, \\ \eta_i^{(t-1)}, & \text{otherwise.} \end{cases}$$

β_- is a shrink factor ($\beta_- < 1$), which is used to reduce the learning rate if the current and the previous gradient have opposite signs. In this case one has leaped over the minimum, so smaller steps are necessary to approach it. Typical values for β_- are between 0.5 and 0.7.

β_+ is a growth factor ($\beta_+ > 1$), which is used to increase the learning rate if the current and the previous gradient have the same sign. In this case two steps are carried out in the same direction, so it is plausible to assume that we have to run down a longer slope of the error function. Consequently, the learning rate should be increased in order to proceed faster. Typically, β_+ is chosen between 1.05 and 1.2, so that the learning rate grows slowly.

The second condition for the application of the growth factor β_+ prevents that the learning rate is increased immediately after it has been decreased in the previous step. A common way of implementing this is to simply set the previous gradient to zero in order to indicate that the learning rate was decreased. Although this somewhat crude measure also suppresses two consecutive reductions of the learning rate, it has the advantage that it eliminates the need to store the gradient $\nabla_{\theta_i} e\big(\vec{\theta}^{(t-2)}\big)$ (or at least its sign).

In order to prevent the weight changes from becoming too small or too large, it is common to limit the learning rate to a reasonable range. It is also recommended to use *batch training* (update after all data points have been processed), as *online training* (update after each data point) tends to be unstable (the learning rates may be updated in an inconsistent way).

6.2.4 Resilient Backpropagation

The resilient backpropagation approach (Rprop) [Riedmiller and Braun 1993] can be seen as a combination of the ideas of Manhattan training (which is like standard backpropagation, but only the sign of the gradient is used, so that the learning rate determines the step width directly) and self-adaptive backpropagation. For each parameter of the objective function a

step width $\Delta\theta_i$ is introduced, which is adapted according to the values of the current and the previous gradient. The adaptation rule reads $\forall i; 1 \leq i \leq k$:

$$\eta_i^{(t)} = \begin{cases} \beta_- \cdot \Delta\theta_i^{(t-1)}, & \text{if } \nabla_{\theta_i} e\big(\vec{\theta}^{(t)}\big) \cdot \nabla_{\theta_i} e\big(\vec{\theta}^{(t-1)}\big) < 0, \\ \beta_+ \cdot \Delta\theta_i^{(t-1)}, & \text{if } \nabla_{\theta_i} e\big(\vec{\theta}^{(t)}\big) \cdot \nabla_{\theta_i} e\big(\vec{\theta}^{(t-1)}\big) > 0, \\ & \wedge\ \nabla_{\theta_i} e\big(\vec{\theta}^{(t-1)}\big) \cdot \nabla_{\theta_i} e\big(\vec{\theta}^{(t-2)}\big) \geq 0, \\ \Delta\theta_i^{(t-1)}, & \text{otherwise.} \end{cases}$$

In analogy to self-adaptive backpropagation, β_- is a shrink factor ($\beta_- < 1$) and β_+ a growth factor ($\beta_+ > 1$), which are used to decrease or increase the step width. The application of these factors is justified in exactly the same way as for self-adaptive backpropagation (see preceding section). The typical ranges of values also coincide ($\beta_- \in [0.5, 0.7]$ and $\beta_+ \in [1.05, 1.2]$).

Like in self-adaptive backpropagation the step width is restricted to a reasonable range in order to avoid far jumps in the parameter space as well as slow learning. It is also advisable to use *batch training*, because *online training* can be very unstable (compare the preceding section, as the same recommendations hold for super self-adaptive backpropagation).

In several applications resilient backpropagation has proven to be superior to a lot of other approaches (including momentum term, self-adaptive backpropagation, and quick backpropagation), especially w.r.t. the training time [Zell 1994]. It is definitely one of the most highly recommendable methods for training multilayer perceptrons with a gradient descent approach.

6.2.5 Quick Backpropagation

The idea underlying quick backpropagation (Quickprop) [Fahlman 1988] is to locally approximate the error function by a parabola (see Figure 6.7). The parameter is then changed in such a way that we end up at the apex of this parabola, that is, the parameter is simply set to the value at which the apex of the parabola lies. If the objective function is "good-natured", i.e., can be approximated well by a parabola, this enables us to get fairly close to the true minimum in one or very few steps.

The update rule for the parameters can easily be derived from the derivative of the approximation parabola (see Figure 6.8). Clearly it is (consider the shaded triangles, both of which describe the ascent of the derivative)

$$\frac{\nabla_{\theta_i} e\big(\vec{\theta}^{(t-1)}\big) - \nabla_{\theta_i} e\big(\vec{\theta}^{(t)}\big)}{\theta_i^{(t-1)} - \theta_i^{(t)}} = \frac{\nabla_{\theta_i} e\big(\vec{\theta}^{(t)}\big)}{\theta_i^{(t)} - \theta_i^{(t+1)}}.$$

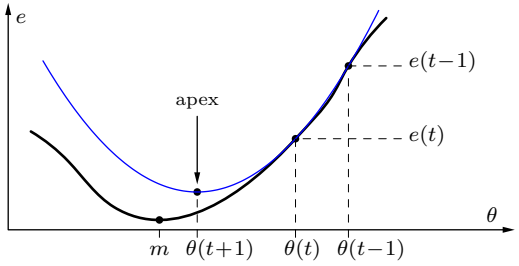

Figure 6.7: Quick backpropagation approximates the error function locally with a parabola. m is the true minimum.

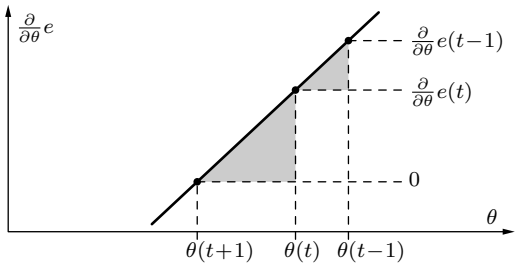

Figure 6.8: The formula for the weight change can easily be derived from the derivative of the approximation parabola.

Solving for $\Delta\theta_i^{(t)} = \theta_i^{(t+1)} - \theta_i^{(t)}$ and exploiting that $\Delta\theta_i^{(t-1)} = \theta_i^{(t)} - \theta_i^{(t-1)}$, we get

$$\Delta\theta_i^{(t)} = \frac{\nabla_{\theta_i} e\big(\vec{\theta}^{(t)}\big)}{\nabla_{\theta_i} e\big(\vec{\theta}^{(t-1)}\big) - \nabla_{\theta_i} e\big(\vec{\theta}^{(t)}\big)} \cdot \Delta\theta_i^{(t-1)}.$$

However, it has to be taken into account that the above formula does not distinguish between a parabola that opens upwards and one that opens downwards, so that a maximum of the error function may be approached. Although this can be avoided by checking whether

$$\frac{\nabla_{\theta_i} e\big(\vec{\theta}^{(t-1)}\big) - \nabla_{\theta_i} e\big(\vec{\theta}^{(t)}\big)}{\Delta\theta_i^{(t-1)}} < 0$$

holds (parabola opens upwards), this check is often missing in implementations. Furthermore a growth factor is introduced, which limits the weight change relative to the previous step. That is, it is made sure that

$$|\Delta\theta_i^{(t)}| \leq \beta \cdot |\Delta\theta_i^{(t-1)}|,$$

where β is a parameter, which is commonly chosen between 1.75 and 2.25. That is, the step width should at most double from one step to the next.

In addition, neural network implementations of this method often add a normal gradient descent step if the two gradients $\nabla_{\theta_i} e\big(\vec{\theta}^{(t)}\big)$ and $\nabla_{\theta_i} e\big(\vec{\theta}^{(t-1)}\big)$ have the same sign, i.e., if the minimum does not lie between the current and the previous weight value. Finally, it is advisable to limit the weight change in order to avoid far jumps in the parameter space.

If the assumptions underlying the quick backpropagation method hold, namely if the error function can be approximated locally by a parabola that opens upwards and the parameters can be changed fairly independent of each other, and if *batch training* is used, it is one of the fastest gradient-based learning methods for multilayer perceptrons. Otherwise it tends to be unstable and is fairly susceptible to oscillations.

A more sophisticated method, which is closely related to quickpropagation, approximates the objective function locally by a multidimensional quadratic function in the full parameter space [Press *et al.* 1992], rather than only in single parameters as quickpropagation does. An advantage of such an approach is, of course, that a better approximation of the objective function can be achieved and thus the number of iterations needed until convergence may be even smaller. However, a severe disadvantage of such an approach is that a large number of derivatives (for the gradient and the second derivatives for the Hessian matrix, which are in all in the order of k^2 derivatives, where k is the number of parameters) have to be computed from the objective function.

An even more sophisticated update method, which builds on the above, is known as the **Levenberg-Marquardt method** [Levenberg 1944, Marquardt 1963]. Intuitively, it varies smoothly between an update based on the approximation by a multidimensional quadratic function and a pure gradient-based update. Details about this methods, which are beyond the scope of this thesis, can be found, for example, in [Press *et al.* 1992].

Chapter 7

Evaluation Methods

After a classifier or a cluster model have been constructed, one would like to know how "good" it is. Quality criteria are fairly easy to find for classifiers: we desire that they *generalize* well and thus yield, on new data, an error rate that is as small as possible. However, due to possible *overfitting* to the training data (that is, adaptations to features of the data that are not regular, but accidental) the error rate on the training data is usually not too indicative of this: the classifier yielding the lowest error rate on the training data is not necessarily the one yielding the lowest error rate on new data. Hence approaches to assess the classification quality are usually based on the idea of a *validation data set*, with so-called *cross validation* being one of the most popular techniques. Such methods are studied in Section 7.1.

Assessing the quality of a clustering result, discussed in Section 7.2, is much more difficult, because there is no target variable that has to be hit as closely as possible. As a starting point we only have the somewhat vague goal that data points in the same cluster should be as similar as possible while data points assigned to different clusters should be as dissimilar as possible. As a consequence a large variety of approaches is based on developing concepts that make this goal more precise. This gives rise to *evaluation measures*, by which the quality of a cluster model is quantified, thus making it possible to compare different cluster models (cf. Sections 7.2.1 and 7.2.2). A second approach is based on the insight that a clustering algorithm always yields a result, regardless of whether the data exhibit a (clear) cluster structure or not. In order to determine whether the result captures regularities in the data or is an artefact of the algorithm, one may *resample* the data set several times and compare the results (cf. Section 7.2.3).

7.1 Assessing the Classification Quality

Classifier assessment serves two purposes: in the first place we would like to know how reliable a classifier is, that is, we would like to have an estimate of how accurately it will perform on new data points. Depending on the result we will then base our decisions mainly on the classification result (if the classifier is highly reliable) or will take it only as a hint that has to be validated by other means in order to make a decision.

Secondly, an estimate of a classifier's performance on new data can be used for model selection. Usually we have the possibility to construct several different classifiers, for example, by choosing different numbers of prototypes or allowing for different prototype properties (cf. Section 2.3). From these classifiers we would like to the select the one that performs best, though, of course, not on the training data. Hence we face again the problem of estimating the performance of a classifier on new data.

It should be noted that, although the focus of this thesis is on prototype-based classifiers, the assessment methods discussed in this section are general and can be applied with any type of classifier.

7.1.1 Causes of Classification Errors

In order to gain a better insight into the problems of estimating the performance of a classifier on new data, it is advantageous to split the classification error into three parts that refer to three different error causes. The three parts are the *Bayes error*, the *bias error*, and the *variance error* or *scatter*.

The **Bayes error** is due to the fact that given any data point usually several classes have a non-vanishing probability. However, if a decision has to be made for a given data point, one has to choose one of these classes. Since several classes are possible, it cannot be guaranteed that the choice is correct, regardless of how the choice is made. The error may, of course, be minimized by choosing the most probable class given the data point. This is the approach of **Bayesian classification**, which explains the name *Bayes error*: it is the error that such an (optimal) Bayes classifier makes. However, even the most probable class may be wrong (if it is not the only one possible). Therefore this error type is independent of the classifier and cannot be eliminated as long as the data space is left unchanged. The only way in which it may be possible to get rid of the Bayes error is by enhancing the data space with additional attributes, which lead to a distribution in which for each data point only one class is possible. However, such additional attributes are not always available or sufficiently easy to obtain.

The **bias error** is connected to the family of classifiers, a member of which is selected by the classifier construction process. For example, we may consider the family of classifiers having a given limited number of prototypes with certain properties, so that constructing the classifier consists only in adapting parameters.[1] The bias error results from the fact that the employed family of classifiers may have only a limited capacity to model the optimal classification function, that is, the one that predicts the most probable class for each data point. In such a case even the best classifier from the family (i.e., the one that is closest to an (optimal) Bayes classifier) may perform worse than Bayesian classification and thus has a higher error rate than the Bayes error. The bias error is simply the difference of its error rate to the Bayes error. Unlike the Bayes error, however, the bias error can be reduced or even made to vanish by changing the family of classifiers under consideration. In general more complex classifiers (for example, with more prototypes or with more parameters to describe the shape and size of each prototype) have a lower bias error, because they have more degrees of freedom that can be adapted, making it possible to model the optimal classification function more precisely.

The **variance error** or **scatter** is due to the fact that, in practice, we cannot determine a classifier by selecting the member of the given family of classifiers that best approximates the optimal classification function (in this case we would only have the Bayes error and the bias error). The reason simply is that usually we do not have direct access to the optimal classification function (and if we had, there would be no need to construct a classifier). Rather we build a classifier by adapting it in such a way that its error on a given data set is minimized. However, the given data set is only sampled from the underlying distribution and thus may not portray it well. In addition, it may exhibit random peculiarities due to the chance mechanisms by which it was obtained. Since the training procedure is given only the data set, it cannot distinguish between regularities of the underlying distribution and accidental peculiarities of the given data set. Hence the classifier may be adapted in such a way that it captures not only the regularities but also the accidental properties that are exhibited. This effect is known as **overfitting** the data set. It results in a high variance of the classifiers learned from different samples from the underlying distribution, which explains the name *variance error*. Alternatively, we may say that the learned classifiers follow the variance in sampled data sets.

[1] This is analogous to statistical parameter estimation, which, by choosing parameter values, selects a member of a given family of distribution functions. If the classifier is based on a probabilistic model, its training is equivalent to parameter estimation.

From the above explanations it should be clear that the *bias error* and the *variance error* are complementary. A family of classifiers has a low bias error if it can model the optimal classification function well. However, by the same means it will usually be able to model accidental properties of the training data set well and thus will be highly susceptible to overfitting, leading to a high variance error. As a consequence one usually has to aim for a tradeoff between the bias error and the variance error, so that the overall error (sum of *Bayes error*, *bias error*, and *variance error*) is minimized.

7.1.2 Cross Validation

Unfortunately, a classifier's error rate on the training data set is not too indicative of how well it will perform on new data. Although these errors should comprise the *Bayes error* (because this error is independent of the classifier) and the *bias error* (because this error depends only on how well the optimal classification function can be modeled), they do not yield any information about the variance error. Rather it is clear that the error rate on the training data is likely to be (much) lower than the true overall error rate on new data, because the classifier was tuned to fit the training data well. Depending on how well the training data set represents the underlying distribution and how flexibly the chosen type of classifier can be adapted, the overfitting may even be so strong that the error rate is less than the sum of *Bayes error* and *bias error* (for example, with one prototype per data point one can usually achieve a perfect classification *of the training data*).

A common approach to cope with this problem is to use a second data set to estimate the error on new data. That is, the given data set to learn from is split into two parts: the *training data set* and the *validation data set*. The classifier is trained only with the data points in the former and then executed on the data points in the latter. Since the validation data set has not been used to train the classifier, the classifier cannot be adapted to its accidental properties. Therefore the error rate on the validation data set will be indicative of the overall error rate on new data.

If the goal is model selection, several classifiers, in particular, classifiers of different complexity, are trained on the training data and tested on the validation data. Finally the classifier that performs best on the validation data is selected as the result. In this way an overfitting to accidental properties of the training data set can be ruled out, since the validation data set will exhibit different random peculiarities. However, since the classifier is selected based on its performance on the validation data set, there is still some, though considerably lower danger that it will be overfitted to the val-

idation data set. Hence its performance on the validation data set should not be seen as an estimate of its performance on new data. In this case a third data set, usually called *test data set*, is needed to actually estimate the performance of the selected classifier on new data.

Using a validation data set has the disadvantage that some of the available data has to be set apart and cannot be used for training the classifier. As a consequence the estimates of its parameters may be worse than necessary. A popular method to overcome this drawback is so-called **cross validation** [Mosier 1951, Toussaint 1974, Mitchell 1997]. In this approach the available data is split into k parts of roughly equal size, called **folds**, with k to be specified by a user: **k-fold cross validation**. $k-1$ folds are combined into a training set and the remaining fold forms the validation set. Then a classifier is trained on the training data and tested on the validation data. This training and testing is repeated for each of the k possible combinations of $k-1$ folds, that is, in all k classifiers are constructed. Finally the average of the error rates (one for each classifier) on the k validation data sets (which together comprise the whole data set), is taken as an indicator of the performance on new data. The final classifier, however, is not selected from the k already trained classifiers, but constructed anew from the full data set, thus exploiting all available information.

In principle, the split into folds can (and should) be done randomly. However, a completely random split can lead to class distributions in the folds that differ considerably from the overall class distribution, especially if the individual folds are fairly small. Such a divergence of the class distributions in the folds from the overall distribution can be detrimental for the quality of the error estimate provided by cross validation, because a classifier faces a situation in the validation data set that does not represent the situation it is likely to meet in new data. Hence it may perform worse than it actually is. In order to avoid such misjudgment, a technique called **stratification** or **stratified sampling** is used: the data points in the folds are sampled in such a way that the overall class distribution is preserved as closely as possible. A simple way of implementing stratified sampling is to shuffle the data set randomly, then to sort it w.r.t. the class labels, and finally to build a fold by selecting every k-th sample case.

Note that cross validation never tests the final classifier on hold-out data. Hence it does not estimate the performance of this classifier on new data directly. It rather assesses the capacity of a family of classifiers together with a training procedure to produce a classifier that performs well on new data. Therefore cross validation is not only well suited for model selection, but also for comparing and evaluating learning procedures.

7.1.3 Evaluation Measures

In order to assess the quality of a classifier on the validation data, we need an evaluation measure. Some of these measures accept degrees of membership or probabilities for the different classes, but most of them are (originally) defined for a crisp classification. As explained in Section 2.5, if the classifier under consideration is a *nearest prototype* or *maximum membership classifier*, such a crisp prediction o (output) is obtained as $\forall j; 1 \leq j \leq n$:

$$o_j = \zeta_k \qquad \text{where} \qquad k = \operatorname{argmax}_{i=1}^{c} u_i(\vec{x}_j)$$

and the ζ_i, $1 \leq i \leq c$, are the classes associated with the c clusters. If the classifier is based on (linear) classification functions (whether interpreted probabilistically or not), it is computed as $\forall j; 1 \leq j \leq n$:

$$o_j = \operatorname{argmax}_{k=1}^{s} g_k(\vec{x}_j),$$

where the g_k, $1 \leq k \leq s$, are the (linear) classification functions associated with the different classes (cf. Section 2.5). On the other hand, if the evaluation measure allows for degrees of membership or probabilities, the values of $g_k(\vec{x}_j)$, $1 \leq k \leq s$, directly enter the evaluation measure.

Note that even though we want to minimize the number of wrong classifications in order to minimize the number of wrong decisions, it can be useful not to rely exclusively on crisp predictions, but to take degrees of membership or probabilities into account. The reason is that the degrees of membership provide information about the reliability of the classification, which may be exploited in the decision making process. Thus a classifier that yields more misclassifications, but indicates several of them by a slim majority for the predicted class, can be more useful than a crisp classifier with a lower misclassification rate. A selection aiming at such classifiers can be based on a measure that uses class membership degrees directly.

Objective Function

It is immediately clear that all objective functions for classification introduced in Chapter 3, like *0-1 loss*, *quadratic loss* (sum of squared errors), and *absolute loss* (sum of absolute errors) (cf. Section 3.2) as well as *likelihood ratio* (cf. Section 3.4), can be used to evaluate the classifier on the validation data. Such a choice is, in a way, the most natural, because then the performance is assessed by the same means the classifier was constructed with in the first place. In addition, all of these measures (with the exception of 0-1 loss) allow for class membership degrees or class probabilities. Details can be found in Sections 3.2 and 3.4.

Precision and Recall

The evaluation measures named in the preceding paragraph have the disadvantage that they fail to capture the quality with which the different classes are recognized. Especially if the class distribution is skewed, that is, if there are large differences in the frequencies with which the classes occur, they may give a wrong impression. For example, if in a two class problem one class occurs in 95% of all cases, while the other covers only the remaining 5%, a classifier that always predicts the first class reaches an impressive accuracy of 95%—without distinguishing between the classes at all.

Such unpleasant situations are fairly common in practice. For example, illnesses are (fortunately) rare and replies to mailings are (unfortunately?) scarce. Hence a classifier that tries to distinguish between ill and healthy patients or between addressees that reply and those that do not can easily achieve a (very) low error rate by predicting that everyone is healthy or a non-replier. However, it is obvious that such a classifier is useless to a physician or a product manager. In such cases higher error rates are accepted in exchange for a better coverage of the minority class.

In Section 3.2 I mentioned that such situations can be handled by introducing misclassification costs, which is a very effective method. Here, however, I review an approach that tries to measure how well the classes are captured without the need to specify such costs. In this approach two measures, called *precision* and *recall*, are computed for each class, which describe how well the class is captured by the classifier. Since they are class-specific, they allow for a more refined performance analysis.

In order to compute these measures a 2×2 contingency table (cf. Table 7.1) is set up for each class based on the true classes z_j associated with the data points $\vec{x}_j$, $1 \leq j \leq n$, and the classes o_j that are predicted by the classifier. The elements of these contingency tables are $\forall k; 1 \leq k \leq s$:

$$
\begin{aligned}
n_{11}^{(k)}(\mathbf{X}, \vec{z}, \mathbf{C}, \mathbf{W}) &= \sum_{j=1}^{n} \delta_{z_j,k} \cdot \delta_{o_j,k} && \text{(true positives)},\\
n_{01}^{(k)}(\mathbf{X}, \vec{z}, \mathbf{C}, \mathbf{W}) &= \sum_{j=1}^{n} (1 - \delta_{z_j,k}) \cdot \delta_{o_j,k} && \text{(false positives)},\\
n_{10}^{(k)}(\mathbf{X}, \vec{z}, \mathbf{C}, \mathbf{W}) &= \sum_{j=1}^{n} \delta_{z_j,k} \cdot (1 - \delta_{o_j,k}) && \text{(false negatives)},\\
n_{00}^{(k)}(\mathbf{X}, \vec{z}, \mathbf{C}, \mathbf{W}) &= \sum_{j=1}^{n} (1 - \delta_{z_j,k}) \cdot (1 - \delta_{o_j,k}) && \text{(true negatives)}.
\end{aligned}
$$

	$o = k$	$o \neq k$	Σ
$z = k$	$n_{11}^{(k)}$	$n_{10}^{(k)}$	$n_{1.}^{(k)}$
$z \neq k$	$n_{01}^{(k)}$	$n_{00}^{(k)}$	$n_{0.}^{(k)}$
Σ	$n_{.1}^{(k)}$	$n_{.0}^{(k)}$	n

Table 7.1: Contingency table for the computation of precision and recall.

Here $\delta_{z,k}$ is, as usual, the Kronecker symbol (cf. page 60 in Section 3.2). To make the formulae easier to read, the arguments $\mathbf{X}$, $\vec{z}$, $\mathbf{C}$, and $\mathbf{W}$ are dropped in the following. In addition, it is convenient to define the following abbreviations for the row and column sums (cf. Table 7.1)

$$n_{1.}^{(k)} = n_{11}^{(k)} + n_{10}^{(k)}, \qquad n_{0.}^{(k)} = n_{01}^{(k)} + n_{00}^{(k)},$$

$$n_{.1}^{(k)} = n_{11}^{(k)} + n_{01}^{(k)}, \qquad n_{.0}^{(k)} = n_{10}^{(k)} + n_{00}^{(k)}.$$

With these numbers, the **precision** π_k of the given classifier for class k, $1 \leq k \leq s$ and its **recall** ρ_k for this class, are defined as [Rijsbergen 1979]

$$\pi_k = \frac{n_{11}^{(k)}}{n_{.1}^{(k)}} = \frac{n_{11}^{(k)}}{n_{11}^{(k)} + n_{01}^{(k)}} \qquad \text{and} \qquad \rho_k = \frac{n_{11}^{(k)}}{n_{1.}^{(k)}} = \frac{n_{11}^{(k)}}{n_{11}^{(k)} + n_{10}^{(k)}}.$$

That is, precision is the ratio of true positives to all data points classified as class k, while recall is the ratio of true positives to all data points actually belonging to class k. In other words: precision is the fraction of data points for which the classification as class k is correct, and recall is the fraction of data points of class k that is identified by the classifier.

It is easy to see that usually higher levels of precision may be obtained at the price of lower values of recall and vice versa. For instance, if more data points are classified as class k, usually more of those data points actually belonging to class k will be classified as class k, thus raising recall. However, it is likely that at the same time more data points *not* belonging to class k will be misclassified as class k, thus reducing precision. In the extreme case, all data points are classified as class k, so that recall is 1. However, in this case precision is only the fraction of all data points that belong to class k. On the other hand, some data points can usually be classified as class k with high certainty, yielding a high precision. However, by restricting the classifier to assign class k to these few cases, one usually captures only a small fraction of the data points actually belonging to class k.

Precision and recall were originally developed for two-class problems, where one class is at the focus of attention [Rijsbergen 1979]. In this case precision and recall for this focus class provide a comprehensive assessment of the classifier. However, as it was implicitly done above, one may also see precision and recall as class-specific measures and use them for multi-class problems. In this case as classifier is evaluated by $2s$ numbers (where s is the number of classes). Although this provides a very detailed picture, it is, of course, a bit inconvenient. In order to obtain estimates for precision and recall in the collection as a whole, they are averaged to reduce them to two values. At least three different averages have been proposed.

The first method is **macro-averaging** [Sebastiani 2002]. In this method precision and recall are first computed for all classes individually and then the results are (arithmetically) averaged over all classes. That is, we have for (macro-averaged) overall precision

$$\pi_{\text{macro}} = \frac{1}{s}\sum_{k=1}^{s}\pi_k = \frac{1}{s}\sum_{k=1}^{s}\frac{n_{11}^{(k)}}{n_{.1}^{(k)}} = \frac{1}{s}\sum_{k=1}^{s}\frac{n_{11}^{(k)}}{n_{11}^{(k)}+n_{01}^{(k)}}$$

and for (macro-averaged) overall recall

$$\rho_{\text{macro}} = \frac{1}{s}\sum_{k=1}^{s}\rho_k = \frac{1}{s}\sum_{k=1}^{s}\frac{n_{11}^{(k)}}{n_{1.}^{(k)}} = \frac{1}{s}\sum_{k=1}^{s}\frac{n_{11}^{(k)}}{n_{11}^{(k)}+n_{10}^{(k)}}.$$

A second, closely related method is to weight the precision and recall values in the average with the relative frequency of the class to which they refer. That is, the overall precision is computed as

$$\pi_{\text{wgt}} = \sum_{k=1}^{s}\frac{n_{1.}^{(k)}}{n}\pi_k = \frac{1}{n}\sum_{k=1}^{s}\frac{n_{1.}^{(k)}}{n_{.1}^{(k)}}n_{11}^{(k)}$$

and the overall recall as

$$\rho_{\text{wgt}} = \sum_{k=1}^{s}\frac{n_{1.}^{(k)}}{n}\rho_k = \sum_{k=1}^{s}\frac{n_{1.}^{(k)}}{n}\frac{n_{11}^{(k)}}{n_{1.}^{(k)}} = \frac{1}{n}\sum_{k=1}^{s}n_{11}^{(k)}.$$

In this case recall obviously coincides with the (relative) **classification accuracy**: it is the ratio of the total number of true positives, that is, the number of correctly classified instances, to the total number of instances. Hence it states the fraction of correctly classified data points. Precision also takes into account how well the classifier captures the class frequencies.

It should be noted that these two versions of averaging precision and recall may give fairly different results if the classes differ considerably in their frequency. Since the first version treats all classes the same, regardless of the number of data points belonging to them, it emphasizes the ability of the classifier to yield good results on small classes (i.e., classes with few data points). This is actually very similar to a cost-based approach (cf. the introduction of misclassification costs into the objective function in Section 3.2) in which the misclassification costs for each class are inversely proportional to its frequency. Which of the two methods is more appropriate depends on the application and the class frequency distribution.

A third method to obtain overall precision and recall values is known under the name of **micro-averaging** [Sebastiani 2002]. In this method the counts for true positives, false positives and false negatives (from which precision and recall are computed) are first summed over all classes. Then overall values of precision and recall are computed using the global values. That is, we have for (micro-averaged) precision

$$\pi_{\text{micro}} = \frac{\sum_{k=1}^{s} n_{11}^{(k)}}{\sum_{k=1}^{s} n_{.1}^{(k)}} = \frac{\sum_{k=1}^{s} n_{11}^{(k)}}{\sum_{k=1}^{s} \left(n_{11}^{(k)} + n_{01}^{(k)}\right)} = \frac{1}{n} \sum_{k=1}^{s} n_{11}^{(k)}$$

and for (micro-averaged) recall

$$\rho_{\text{micro}} = \frac{\sum_{k=1}^{s} n_{11}^{(k)}}{\sum_{k=1}^{s} n_{1.}^{(k)}} = \frac{\sum_{k=1}^{s} n_{11}^{(k)}}{\sum_{k=1}^{s} \left(n_{11}^{(k)} + n_{10}^{(k)}\right)} = \frac{1}{n} \sum_{k=1}^{s} n_{11}^{(k)}.$$

The last steps follow from the fact that summing the number of cases belonging to (recall) or predicted to be in each class (precision) yields the total number of cases. Hence in this case precision and recall are identical and also identical to the classification accuracy. This identity renders micro-averaged precision and recall basically useless, since they do not add anything to the expressiveness of the straightforward accuracy measure. Strangely enough this identity, which causes me to neglect these micro-averages entirely, seems to be widely unrecognized.[2]

[2]This can be inferred from the fact that the last step in the above equalities is usually missing from the definition and it is also not pointed out in any other way that the denominators are actually equal to n, the total number of cases (see, for instance, [Sebastiani 2002], but also several other machine learning papers). It may be, though, that micro-averaging yields useful results if the classification is not exclusive, that is, if a data point may belong to (and may be assigned to) several classes at the same time. However, this is not the scenario in which one finds it usually applied.

Combined Measures

With (macro-)averaged precision and recall one still has two numbers that assess the quality of a classifier. A common way to combine them into one number is to compute the **$\mathbf{F_1}$ measure** [Rijsbergen 1979], which is the harmonic average (cf. Section A.10 in the appendix) of precision and recall:

$$F_1 = \frac{2}{\frac{1}{\pi} + \frac{1}{\rho}} = \frac{2\pi\rho}{\pi + \rho}.$$

In this formula precision and recall have the same weight. The generalized F measure [Rijsbergen 1979] introduces a mixing parameter. It can be found in several different, but basically equivalent versions. For example,

$$F(\alpha) = \frac{1}{\frac{\alpha}{\pi} + \frac{1-\alpha}{\rho}} = \frac{\pi\rho}{\alpha\rho + (1-\alpha)\pi}, \qquad \alpha \in [0,1],$$

or

$$F(\beta) = \frac{1+\beta^2}{\frac{1}{\pi} + \frac{\beta^2}{\rho}} = \frac{\pi\rho(1+\beta^2)}{\rho + \beta^2\pi}, \qquad \beta \in [0,\infty).$$

Obviously, the standard F_1 measure results from these formulae for $\alpha = \frac{1}{2}$ or $\beta = 1$, respectively. (The latter explains the index 1 of the F_1 measure.) By choosing α or β it can be controlled whether the focus should be more on precision ($\alpha > \frac{1}{2}$ or $\beta > 1$; for $\alpha = 1$ or $\beta = 0$ we have $F(\alpha) = \pi$) or more on recall ($\alpha < \frac{1}{2}$ or $\beta < 1$; for $\alpha = 0$ or $\beta \to \infty$ we have $F(\alpha) = \rho$). However, this possibility is rarely used, presumably, because precision and recall are usually considered to be equally important.

Apart from averaging precision and recall before computing the F_1 measure, we may also compute the F_1 measure for each category individually and then average the results weighted with the probability of the class:

$$F_{1,(\text{alt})} = \sum_{k=1}^{s} \frac{n_{1.}^{(k)}}{n} F_1^{(k)} \qquad \text{where} \qquad F_1^{(k)} = \frac{2\pi_k\rho_k}{\pi_k + \rho_k}.$$

Several other combined measures are really useful only for two class problems, but may, in principle, be used for multi-class problems by averaging them over all classes (as it was done with the F_1 measure above). Examples are **11-point average precision** and the **breakeven point** [Baeza-Yates and Ribeiro-Neto 1999]. Both of these measures exploit the fact that it is usually possible to tune the classifier in such as way that recall (for the class under consideration) takes the values 0.0, 0.1., ..., 0.9, 1.0. This may be

achieved, for example, by setting a threshold for the probability or membership degree that has to be exceeded in order to assign the class focused on. Then the precision (for the focus class) is computed for these 11 points and the results are averaged. (The choice of the step width 0.1, which leads to 11 points, is, of course, somewhat arbitrary; other step widths may be used too.) The breakeven point exploits a similar idea, but tries to find, by tuning the classifier, the recall value for which precision equals recall. This value is then used as a combined measure.

An even more sophisticated measure, though again really useful only for two-class problems, is the **power**, which is defined as twice the area between the **receiver operating characteristic curve** (ROC curve) and the diagonal of the unit square [Hanley and McNeil 1982]. However, a detailed explanation of this measure is beyond the scope of this thesis.

Penalized Measures

All measures discussed up to now are meant to be computed on a validation set, since they obviously overestimate the quality of the classifier if they are computed on the training data (cf. Section 7.1.1). However, there are also approaches that try to estimate the true error rate of the classifier (that is, the error rate on new data) from the error rate on the training data and a penalty term that is meant to account for possible overfitting. Usually this penalty term takes into account the flexibility of the model to fit the data, measured, for example, by the number of parameters that can be adapted. Several such methods have already been mentioned on pages 91ff in Section 4.3, for example, information criteria and the minimum description length principle (MDL) [Rissanen 1983, Rissanen 1987]. An alternative that was not mentioned is based on the Vapnik–Chervonenkis dimension (VC dimension) of a class of classifiers. Details can be found, for instance, in [Vapnik 1995, Vapnik 1998].

It should be noted, though, that cross validation is a clearly preferable approach, since all penalization approaches must conjecture how the true error rate grows with the model complexity [Scheffer and Joachims 1999]. This conjecture will inevitably fail for some learning problems, leading to a limited performance of such approaches [Kearns *et al.* 1997]. Nevertheless penalized approaches are very popular due to their simplicity and the fact that only a single learning process is necessary for each class of classifiers. In contrast to this, k-fold cross validation needs $k + 1$ learning executions of the learning algorithm (k for the k combinations of $k - 1$ folds and one for the final construction of the classifier from the entire data set).

An alternative to cross validation as well as to penalized measures that is worth mentioning has been developed by [Scheffer and Joachims 1999]. It is based on the idea to evaluate randomly selected classifiers from the given model class on the data in order to obtain an expected error rate. From this an indication of the best model class can be derived, which according to the authors is highly competitive to cross validation. A detailed discussion of this approach is, however, beyond the scope of this thesis.

7.2 Assessing the Clustering Quality

While assessing the quality of a classifier is fairly easy, because for each data point we have a target class the classifier should predict, assessing a cluster model is much more difficult, because there is no such reference value. All we have as a basis for an assessment is the somewhat vague goal that data points assigned to the same cluster should be as similar as possible, while data points assigned to different clusters should be as dissimilar as possible. In addition, we face the unpleasant problem that most clustering algorithms always produce a cluster model, for any data set. Hence it may be that the obtained cluster structure is only an artifact of the clustering algorithm and does not capture actual regularities in the data.

As a consequence, methods to assess the clustering quality have two purposes: in the first place, one would like to have means to select the most appropriate cluster model. In particular, this means choosing the number of clusters, because basically all algorithms discussed in this thesis (with the exception of some of the more sophisticated initialization methods, cf. Section 4.3) presuppose that the number c of clusters is known. Approaches in this direction are usually based on (internal) evaluation measures (cf. Section 7.2.1), which try to give precise meaning to the somewhat vague goal of clustering stated above. Alternatively, one may compare the clustering result to a reference, which is obtained from class labels assigned to the data points, background knowledge about the domain, or simply another clustering result with a different algorithm. How such comparisons can be done with (relative) evaluation measures is studied in Section 7.2.2.

Secondly, one would like to know whether the data set actually exhibits a cluster structure or whether the result is an artefact of the algorithm. Approaches to answer this question are traditionally based on *resampling* (cf. Section 7.2.3). However, resampling approaches may also be used for model selection, since, for example, the right choice of the number of clusters should lead to much more stable resampling results.

7.2.1 Internal Evaluation Measures

Suggestions for evaluation measures for cluster models abound and thus it is impossible to provide a full list. Overviews, which are necessarily incomplete themselves, though, can be found, for example, in [Pal and Bezdek 1995, Bezdek and Pal 1998, Bezdek *et al.* 1999, Höppner *et al.* 1999, Halkidi *et al.* 2002a, Halkidi *et al.* 2002b]. In this and the next section I review several of the better known indices and quality measures. This section focuses on **internal measures**, that is, measures that exploit only information from the data set and the clustering result. In contrast to this, the next section discusses **relative measures**, that is, measures that compare two groupings or a clustering result with a given class assignment.

Note that the term *relative measure* is often reserved for the comparison of two clustering results, while measures for the comparison of a clustering result with a given class assignment are called **external measures** [Halkidi *et al.* 2002a]. However, since these measures draw on the same ideas as long as they only consider assignments of the data points to clusters (or classes), I do not distinguish them here. It has to be conceded, though, that cluster models may also be compared by comparing their parameters. Such approaches are not captured in the term *external measure*.

Since classical clustering is concerned with a crisp partition of the data points, some of the measures reviewed in the following were originally developed for crisp clustering results. Fortunately, however, they usually become applicable to fuzzy and probabilistic clustering results if they are written with the elements of the partition matrix $\mathbf{U}$. Furthermore, not all of the measures can handle clusters that employ size and shape parameters, but rather presuppose (hyper-)spherical clusters of equal size. The main reason for this restriction is that with cluster-specific size and shape information the distance between clusters is undefined, because there is no uniform distance measure on the data space anymore. As a solution one may define the distance between two clusters by averaging the distance of the center of one as seen from the other [Timm 2002]. However, one has to check carefully whether this leads to an appropriate evaluation measure. As a consequence of these problems, I try to point out, as far as possible, the limits of applicability of the different evaluation measures.

Finally, it is recommended to compute the measures on a validation data set, which has not been used for building the cluster model, if the goal is model selection (e.g., choosing the number of clusters). The idea is the same as in model selection for classification (cf. Section 7.1.2): one tries to avoid overfitting to accidental properties of the training data.

Objective Function

As for evaluating classification results (cf. Section 7.1.3), it is immediately clear that all objective functions introduced in Chapter 3, like the (weighted) *sum of squared distances* (cf. Section 3.1) or the *likelihood* of the data (cf. Section 3.3), can be used to evaluate a cluster model. These are actually the most natural choices for evaluation measures, because the quality of a clustering result was originally defined by these objective functions.

It may even be argued that the quality of a clustering result is only defined by whether it is the optimum of the objective function [Bezdek *et al.* 1999]. However, this should not be interpreted as a general argument against other evaluation measures. Such measures can provide useful information about certain properties of the clustering result, which is not expressed as clearly by the objective function. In addition, most of the measures stated below cannot be used as objective functions, because it is not possible to optimize them without additional constraints.

Dunn Index / Separation Index

One of the oldest quality measures for (crisp) clustering is the *Dunn index* [Dunn 1973], which is also known as the *separation index*. Actually it is a family of indices, because it is generally defined as

$$Q_{\text{Dunn}}(\mathbf{C}, \mathbf{U}, \mathbf{X}) = \frac{\min_{1 \leq i < k \leq c} d(\mathbf{c}_i, \mathbf{c}_k; \mathbf{U}, \mathbf{X})}{\max_{1 \leq i \leq c} S(\mathbf{c}_i; \mathbf{U}, \mathbf{X})}$$

where $d(\mathbf{c}_i, \mathbf{c}_j; \mathbf{U}, \mathbf{X})$ is a cluster distance measure and $S(\mathbf{c}_l; \mathbf{U}, \mathbf{X})$ is a measure for the diameter of a cluster. Each choice of a measure for the cluster distance d and the cluster diameter S yields a member of this index family.

Obviously, the separation index is directly derived from the general goal of clustering. That data points from the same cluster should be as similar as possible is interpreted as: the (maximum) diameter of a cluster should be as small as possible; and that data points from different clusters should be as dissimilar as possible is interpreted as: the (minimum) pairwise distance between clusters should be as large as possible. By dividing the former by the latter one obtains a measure that should be maximized.

[Dunn 1973] originally used for the cluster distance measure

$$d(\mathbf{c}_i, \mathbf{c}_k; \mathbf{U}, \mathbf{X}) = \min_{\substack{1 \leq j, l \leq n: \\ u_{ij} u_{kl} = 1}} d(\vec{x}_j, \vec{x}_l),$$

that is, the smallest distance between two data points, one from each cluster.

The diameter of a cluster [Dunn 1973] defined as

$$S(\mathbf{c}_i; \mathbf{U}, \mathbf{X}) = \max_{\substack{1 \le j,l \le n: \\ u_{ij} u_{il} = 1}} d(\vec{x}_j, \vec{x}_l),$$

that is, as the largest distance between two data points from the cluster. Technically this presupposes that at least one cluster contains at least two data points, which is no relevant constraint in practice, though.

However, a serious problem of the original Dunn index is that it is highly sensitive to noise. Since both the cluster distance and the cluster diameter are defined through operations that neglect all but one distance, a single outlier can change the value of the Dunn index considerably.

A better, that is, much more robust version was suggested by [Bezdek *et al.* 1997]. Here the distance between two clusters is defined as

$$d(\mathbf{c}_i, \mathbf{c}_k; \mathbf{U}, \mathbf{X}) = \frac{1}{\left(\sum_{j=1}^{n} u_{ij}\right)\left(\sum_{j=1}^{n} u_{kj}\right)} \sum_{j=1}^{n} \sum_{l=1}^{n} u_{ij} u_{kl}\, d(\vec{x}_j, \vec{x}_l),$$

that is, as the average distance between two data points, one from each cluster. In a matching fashion, the diameter of a cluster is defined as

$$S(\mathbf{c}_i; \mathbf{U}, \mathbf{X}) = 2\, \frac{\sum_{j=1}^{n} u_{ij}\, d(\vec{x}_j, \vec{\mu}_i)}{\sum_{j=1}^{n} u_{ij}},$$

i.e., as twice the arithmetic mean of the data point distance from the cluster center, which may be seen as a kind of cluster radius (hence the factor two: diameter $= 2 \times$ radius). [Bezdek *et al.* 1997] reported very good and reliable experimental results with this version of the separation index.

It should be noted that up to now it was assumed that $u_{ij} \in \{0, 1\}$, $1 \le i \le c$, $1 \le j \le n$. Indeed, this is indispensable for the original version of the Dunn index as proposed by [Dunn 1973]. In the version by [Bezdek *et al.* 1997], however, we may just as well have $u_{ij} \in [0, 1]$, as this only leads to a weighting of the distances. Hence this version is also applicable to fuzzy or probabilistic clustering results, making it a very versatile index.

Even applying it to non-spherical clusters (that is, with distances based on a cluster-specific covariance matrix) is imaginable. The distance between two clusters may be computed, as mentioned before, by averaging the mutual distances (which may differ due to different covariance matrices), and the diameter of a (hyper-)ellipsoidal cluster may be defined as twice its isotropic radius (cf. Section 2.3). However, a careful study and experimental evaluation of this possibility seems to be missing yet.

Davies–Bouldin Index

Similar to the Dunn index, the *Davies–Bouldin index* [Davies and Bouldin 1979] is actually a family of indices. It is defined generally as

$$Q_{\mathrm{DB}}(\mathbf{C}, \mathbf{U}, \mathbf{X}) = \frac{1}{c} \sum_{i=1}^{c} \max_{\substack{1 \leq k \leq c \\ k \neq i}} \frac{S(\mathbf{c}_i; \mathbf{U}, \mathbf{X}) + S(\mathbf{c}_k; \mathbf{U}, \mathbf{X})}{d(\mathbf{c}_i, \mathbf{c}_k; \mathbf{U}, \mathbf{X})}$$

where d is a cluster distance measure and S measures the scatter within a cluster[3] and thus intuitively the size of a cluster. Depending on the choice of the measures d and S different indices from this family result.

Similar to the Dunn index, the Davies–Bouldin index relates the distances within a cluster (captured by the scatter within a cluster) to the distance between clusters. However, the Dunn index considers these two quantities independently, relating only global aggregates to each other. In contrast to this, the Davies–Bouldin index forms a ratio for each cluster, which is compared to every other cluster. The basic idea underlying this approach is that the assessment of the distance between two clusters should depend on the scatter within the two clusters. Large clusters (that is, clusters with a high scatter) should be further apart in order to be well separated, while small clusters (that is, clusters with a small scatter) can be closer together and still be well separated. With the maximum in the above formula the worst separation in this sense (that is, the separation from the most similar cluster) is determined for each cluster, which corresponds to a high value of the ratio. The full Davies–Bouldin index is the arithmetic average of these cluster-specific separation indicators. As a consequence, it is to be minimized for a good separation.

[Davies and Bouldin 1979] originally used for the cluster distance

$$d(\mathbf{c}_i, \mathbf{c}_k; \mathbf{U}, \mathbf{X}) = d(\vec{\mu}_i, \vec{\mu}_k) = \sqrt{(\vec{\mu}_i - \vec{\mu}_k)^\top (\vec{\mu}_i - \vec{\mu}_k)},$$

that is, the Euclidean distance of the cluster centers, and for the scatter within a cluster (or, intuitively, the size of a cluster)

$$S(\mathbf{c}_i; \mathbf{U}, \mathbf{X}) = \sqrt{\frac{\sum_{j=1}^{n} u_{ij}\, d^2(\vec{x}_j, \vec{\mu}_i)}{\sum_{j=1}^{n} u_{ij}}},$$

that is, the quadratic mean of the distances of the data points from the cluster center (which is equivalent to a maximum likelihood estimate of the

[3] Since measures for the diameter of a cluster may be seen as measures of the scatter within it and vice versa, I use the same letter S for both quantities.

variance of a normal distribution). However, the measures used by [Bezdek *et al.* 1997] for the separation index (see above) may just as well employed.

The Davies–Bouldin index was originally defined for crisp clustering results ($u_{ij} \in \{0, 1\}$), but there is nothing that hinders its application to fuzzy or probabilistic clustering result. In this case the membership degrees $u_{ij} \in [0, 1]$ weight the distances in the measure for the scatter within a cluster (and maybe also in the measure for the cluster distance, if some other measure than the Euclidean distance of the cluster centers is used).

Similar to the Dunn index, an application of the Davies–Bouldin index to non-spherical clusters is imaginable, but has not been studied in detail yet. However, in the subject area of semi-supervised learning [Klose 2004] suggested a modified Davies–Bouldin index that is applicable to non-spherical clusters and led to good results in this application domain.

Xie–Beni Index / Separation

The *Xie–Beni index* [Xie and Beni 1991] or *separation* (not to be confused with the *separation index*, see above) or draws on similar ideas as the preceding two measures, but was defined specifically for evaluating results of the fuzzy clustering, in particular fuzzy c-means clustering. It is defined as

$$Q_{\text{sep}}(\mathbf{C}, \mathbf{U}, \mathbf{X}) = \frac{\frac{1}{n} \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{w} \, d^2(\vec{x}_j, \vec{\mu}_i)}{\min_{1 \leq i < k \leq n} d^2(\vec{\mu}_i, \vec{\mu}_k)}.$$

Clearly, the numerator of this measure contains the standard objective function of fuzzy clustering (cf. page 51 in Section 3.1). This objective function is divided by n in order to remove a dependence on the number of data points. The result can be seen as an average (weighted) distance of the data points from the cluster centers and thus as a global measure of the scatter within the clusters, connecting it to the Davies–Bouldin index. This global scatter measure is finally divided by the minimum distance between two cluster centers, which indicates how well the clusters are separated. Since for a good clustering result the numerator should be as small as possible and the denominator as large as possible, this measure is to be minimized.

Note that the Xie–Beni index is similar to a reciprocal Dunn index, with the maximum diameter of a cluster replaced by the average distance to a cluster center. Note also that by replacing the Euclidean distance of the cluster centers in the denominator by a general cluster distance measure $d(\mathbf{c}_i, \mathbf{c}_k; \mathbf{U}, \mathbf{X})$ the Xie–Beni index may be turned into a family of quality indices, in analogy to the preceding two measures.

The Xie–Beni index is one of the most popular measures for assessing fuzzy clustering results, mainly due to the results of [Pal and Bezdek 1995], who found that is provided the best response over a wide range of choices for the number of clusters. Note that, although the Xie–Beni index was defined for fuzzy clustering, it may just as well be applied to crisp partitions. W.r.t. clusters with adaptable shape and size the same comments apply as for the preceding two measures (*Dunn index* and *Davies–Bouldin index*).

Fukuyama–Sugeno Index

The *Fukuyama–Sugeno index* [Fukuyama and Sugeno 1989] employs a fairly similar idea as the *Xie–Beni index*, but measures and incorporates the separation of the clusters differently. It is defined as

$$\begin{aligned} Q_{\mathrm{FS}}(\mathbf{C}, \mathbf{U}, \mathbf{X}) &= \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{w} \left(d^2(\vec{x}_j, \vec{\mu}_i) - d^2(\vec{\mu}_i, \vec{\mu}) \right) \\ &= \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{w} \, d^2(\vec{x}_j, \vec{\mu}_i) - \sum_{i=1}^{c} d^2(\vec{\mu}_i, \vec{\mu}) \sum_{j=1}^{n} u_{ij}^{w}, \end{aligned}$$

where $\vec{\mu}$ is the global mean of the data points, that is,

$$\vec{\mu} = \frac{1}{n} \sum_{j=1}^{n} \vec{x}_j.$$

Like the *Xie–Beni index*, this measure obviously contains the standard objective function of fuzzy clustering (cf. page 51 in Section 3.1). However, it modifies it by relating every single distance to a cluster center to the distance of this cluster center to the global mean. Alternatively, as can be seen from the second line in the above definition, we may say that it reduces the objective function by the sum of the distances of the clusters centers to the global mean, where each such distance enters with the weight of the data points assigned to it. This sum is the larger, the better separated the clusters are, while the objective function is to be minimized anyway. As a consequence, the Fukuyama–Sugeno index is to be minimized.

[Pal and Bezdek 1995] report that the Fukuyama–Sugeno index is unreliable for high as well as low values of the fuzzifier w. This is actually not too surprising, because the formula already indicates that it prefers large clusters far away from the center and thus imposes a certain bias on the cluster structure. Apart from this unfavorable evaluation, the same comments apply as for the *Xie–Beni index* (see above).

Partition Coefficient

The *partition coefficient* [Bezdek 1973, Bezdek 1981, Windham 1982] was introduced for assessing fuzzy clustering results and intuitively measures how crisp a partition is. It is defined on the fuzzy partition matrix alone as

$$Q_{\mathrm{PC}}(\mathbf{U}) = \frac{1}{n} \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^2.$$

From this formula it is immediately clear that this measure is entirely useless for assessing crisp partitions, because for $u_{ij} \in \{0, 1\}$, $1 \leq i \leq c$, $1 \leq j \leq n$, and with the constraint $\forall j; 1 \leq j \leq n : \sum_{i=1}^{c} u_{ij} = 1$ (that is, each data point is assigned to exactly one cluster), it is always $Q_{\mathrm{PC}}(\mathbf{U}) = 1$, independent of the number of cluster, the number of data points, and how the data points are assigned to the clusters.

For fuzzy partitions, however, it can yield useful results, which are the higher, the closer a fuzzy partition is to a crisp partition. The reason is that due to the squared membership degrees in this formula, the value of the partition coefficient is the smaller, the more uniformly the data points are assigned to all clusters. From this insight we can infer that its minimum value results for $u_{ij} = \frac{1}{c}$, $1 \leq i \leq c$, $1 \leq j \leq n$. In this case we have

$$Q_{\mathrm{PC}}(\mathbf{U}) = \frac{1}{n} \sum_{i=1}^{c} \sum_{j=1}^{n} \frac{1}{c^2} = \frac{1}{n} \frac{nc}{c^2} = \frac{1}{c}.$$

As a consequence we have $Q_{\mathrm{PC}}(\mathbf{U}) \in \left[\frac{1}{c}, 1\right]$ for all fuzzy clustering results.

From this lower bound on the value of the partition coefficient it is already clear that its value depends on the number of clusters, which is inconvenient if one wants to compare clustering results that were obtained with a different number of clusters. In order to amend this drawback, the *normalized partition coefficient* [Backer and Jain 1981, Stutz 1998] has been introduced. This variant of the partition coefficient is defined as

$$Q_{\mathrm{PCnorm}}(\mathbf{U}) = 1 - \frac{c}{c-1}(1 - Q_{\mathrm{PC}}(\mathbf{U})).$$

That this normalization removes the dependence on the number of clusters can be seen by analyzing its range of values as follows:

$$\begin{aligned} Q_{\mathrm{PC}}(\mathbf{U}) \in \left[\tfrac{1}{c}, 1\right] \quad &\Leftrightarrow \quad 1 - Q_{\mathrm{PC}}(\mathbf{U}) \in \left[0, \tfrac{c-1}{c}\right] \\ &\Leftrightarrow \quad \frac{c}{c-1}(1 - Q_{\mathrm{PC}}(\mathbf{U})) \in [0, 1] \\ &\Leftrightarrow \quad Q_{\mathrm{PCnorm}}(\mathbf{U}) \in [0, 1]. \end{aligned}$$

The rationale underlying both the partition coefficient and its normalized version is that a (fuzzy) cluster model that does not fit the data well should lead to ambiguous membership degrees, because the misfit will show itself in the inability of the cluster model to assign data points clearly to one cluster. However, although this view is very intuitive, the usefulness of the partition coefficient is limited in practice. The reason is that a cluster model may fit a large part of the data well, so that only a small number of data points receive ambiguous membership degrees, which may have only a very small impact on the total partition coefficient. In particular, if one tries to estimate the number of clusters with this measure, numbers of clusters that differ only little from the optimal number will also receive good assessments from this measure, rendering the decision at least problematic.

In addition, in fuzzy clustering with inverse squared distances, membership degrees tend to become more equal with a higher number of dimensions of the data space, because the average distance between data points tends to increase with the number of dimensions (cf. Section 2.4 for a detailed investigation of the dependence of the membership degrees on the distance). Hence one may conjecture that for fuzzy clustering algorithms the expressiveness of the partition coefficient does not scale well with the number of dimensions. This is indeed observed in experiments.

It is clear that both the partition coefficient and its normalized variant are applicable to non-spherical clusters and clusters of differing size, because they are defined only on the fuzzy partition matrix. Thus they do not make any assumptions about the cluster size or shape.

Partition/Classification Entropy

Like the partition coefficient, the *partition entropy* [Bezdek 1975, Bezdek 1981, Windham 1982], which is also known as the *classification entropy*, was introduced for assessing fuzzy clustering results. It is defined on the fuzzy partition matrix alone as

$$H(\mathbf{U}) = \frac{1}{n}\sum_{j=1}^{n} H(\vec{u}_j) = -\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{c} u_{ij}\log_2 u_{ij},$$

where $\vec{u}_j = (u_{1j}, \ldots, u_{cj})$ is the i-th row of the fuzzy partition matrix $\mathbf{U}$ and comprises the membership degrees of one data point to the different clusters and H denotes the Shannon entropy [Shannon 1948].

In general, the Shannon entropy of a (strictly positive) probability distribution over r values, given as a vector $\vec{p} = (p_1, \ldots, p_r)$ of probabilities

satisfying $\sum_{i=1}^{r} p_i = 1$, is defined as

$$H^{(\text{Shannon})}(\vec{p}) = -\sum_{i=1}^{r} p_i \log_2 p_i.$$

Intuitively, it measures the expected number of yes/no-questions one has to ask (with an optimal question scheme) in order to discover the obtaining value.[4] The Shannon entropy is always positive and it is the higher, the more uniform the probability distribution is and it is maximal for a completely uniform distribution $p_i = \frac{1}{r}$, $1 \leq i \leq r$, namely $\log_2 r$.

The idea of the partition entropy is to interpret the membership degrees to the different clusters, which formally resemble a probability distribution (if normalized to sum 1), as cluster assignment probabilities. This view is particularly justified, if the underlying model is actually probabilistic (for instance, a Gaussian mixture model) and thus the membership degrees are posterior probabilities. With this view the partition entropy computes the average entropy of the cluster probability distributions, which is the smaller, the closer a fuzzy or probabilistic partition is to crisp partition.

Since its underlying idea is similar to the idea of the partition coefficient, the partition entropy shares basically all properties of the partition coefficient (except that it is to be minimized while the partition coefficient is to be maximized): it is useless for comparing crisp partitions, it does not scale well with the number of dimensions of the data space, but it can be applied to cluster models with clusters of different shape and size.

It should be noted that the partition entropy may also be normalized, like the partition coefficient, by removing the dependence on the number of clusters and data points. However, in this case this normalization is achieved by dividing by the maximum value the partition entropy can have, that is, by $\log_2 c$. Therefore the normalized partition entropy is

$$H_{\text{norm}}(\mathbf{U}) = \frac{1}{n \log_2 c} \sum_{j=1}^{n} H(\vec{u}_j) = -\frac{1}{n \log_2 c} \sum_{j=1}^{n} \sum_{i=1}^{c} u_{ij} \log_2 u_{ij}.$$

Fuzzy Hyper-Volume

The *fuzzy hyper-volume* [Gath and Geva 1989] was introduced in order to evaluate the result of the fuzzy maximum likelihood algorithm, but may generally be used to assess and compare clustering results obtained with

[4] A detailed explanation of the ideas underlying Shannon entropy and its connection to coding theory can be found in [Borgelt and Kruse 2002].

adaptable cluster-specific sizes (with or without adaptable shape). It is defined as the total hyper-volume of all clusters, that is, as

$$Q_{\text{volume}}(\mathbf{C}) = \sum_{i=1}^{c} \sigma_i^m = \sum_{i=1}^{c} \sqrt{|\boldsymbol{\Sigma}_i|},$$

and thus depends only on the (final) cluster parameters. However, due to the discussion in Section 2.3, we know that the volume is only one way of measuring the size of a cluster. We may choose differently, using, for example, the isotropic radius. This gives rise to a generalized version of this measure, which may be defined as

$$Q_{\text{size}}(\mathbf{C}, \kappa) = \sum_{i=1}^{c} \sigma_i^\kappa = \sum_{i=1}^{c} |\boldsymbol{\Sigma}_i|^{\frac{\kappa}{2m}}.$$

From this generalized version the original form obviously results for $\kappa = m$. Other plausible choices are $\kappa = 1$ and $\kappa = 2$ (cf. also Section 6.1.3).

The idea underlying all forms of this measure is that a cluster model that fits the data well should lead to compact clusters and thus a low value of the fuzzy hyper-volume or any other measure for the size of a cluster. Note that it is not applicable if the cluster size is fixed, because then this measure is necessarily constant, regardless of the clustering result. However, it is not restricted to fuzzy partitions, but applies equally well to crisp partitions.

Partition Density

In the fuzzy hyper-volume, as it was discussed above, it is only indirectly taken into account how the data points are distributed in a cluster, namely only through the adaptation of the cluster volume/size in the clustering process. In addition, the relative location of the clusters is neglected (even though the partitioning property of a clustering algorithm in which the membership degrees are normalized to sum 1 limits the possible amount of overlap). To amend these drawbacks, [Gath and Geva 1989] suggested the *partition density*, which intuitively measures the data point density in the "core regions" of the clusters. Formally, the partition density is defined as

$$Q_{\text{PD}}(\mathbf{C}, \mathbf{U}, \mathbf{X}) = \frac{\sum_{i=1}^{c} \sum_{j \in I(i)} u_{ij}}{\sum_{i=1}^{c} \sqrt{|\boldsymbol{\Sigma}_i|}}$$

where

$$I(i) = \{j \mid 1 \leq j \leq n \wedge (\vec{x}_j - \vec{\mu}_i)^\top \boldsymbol{\Sigma}_i^{-1} (\vec{x}_j - \vec{\mu}_i) < 1\}.$$

That is, the "core region" of a cluster is defined as the interior of a (hyper-) sphere with radius one around the cluster center, which is defined by the Mahalanobis distance that is parameterized with the cluster-specific covariance matrix $\boldsymbol{\Sigma}_i$, $1 \leq i \leq c$ (cf. Section 2.3). Only the data points in these interior regions of each cluster are considered, and they are weighted with their degree of membership, so that data points closer to the cluster center have a higher influence. By dividing the result by a value that is proportional to the total size of these interior regions[5], one obtains a measure of the data point density in these interior regions. Obviously, for a good clustering result, this data point density should be as large as possible, so that many data points are covered by few, small clusters and in such a way that only few data points are outside the "core regions" of the clusters.

As an alternative to the above, one may compute first the data point densities for each cluster individually and then (arithmetically) average the results over the clusters. This leads to the so-called *average partition density* [Gath and Geva 1989], which is formally defined as

$$Q_{\mathrm{PDavg}}(\mathbf{C}, \mathbf{U}, \mathbf{X}) = \frac{1}{c} \sum_{i=1}^{c} \frac{\sum_{j \in I(i)} u_{ij}}{\sqrt{|\boldsymbol{\Sigma}_i|}}.$$

Note that the *partition density* and the *average partition density* may yield fairly different results, especially if the clusters are of considerably different size and cover substantially differing numbers of data points. Intuitively, small clusters with few data points have a much lesser influence on the *partition density* than on the *average partition density*.

Note that, although both versions of the partition density were developed for assessing fuzzy clustering results, they may just as well be applied to crisp clustering results, since no difficulties result from a restriction of the u_{ij}, $1 \leq i \leq c$, $1 \leq j \leq n$, to the set $\{0, 1\}$. Unlike the fuzzy hyper-volume, these measures may even be applied if the cluster sizes are not adaptable (and thus the volume of a cluster, as it is measured by $\sqrt{|\boldsymbol{\Sigma}_i|}$, is fixed).

Selecting the Number of Clusters

The general approach to select the number of clusters with an internal evaluation measure is to carry out the chosen clustering algorithm with all numbers of clusters in a certain, user-defined range, then to evaluate the result with an internal evaluation measure, and finally to select the

[5] Consult Section A.4 in the appendix for a detailed explanation why $\sqrt{|\boldsymbol{\Sigma}_i|}$ is proportional to the size of "core region" of a cluster.

number of clusters that yields the best value for this measure. Some indices, however, show a monotone behavior over the number of clusters, that is, they generally tend to decrease (increase) with a larger number of clusters (for example, partition coefficient or partition entropy). In this case one has to look not for minimum, but for a pronounced knee in the graph of the evaluation measure over the number of clusters.

Unfortunately, even though this simple and straightforward procedure sometimes leads to good results, thus identifying the "correct" (in an intuitive sense) number of clusters, the problem of choosing the right number of clusters is far from being solved. Often enough different indices indicate different numbers of clusters and thus the choices recommended are fairly unreliable. How one can reliably determine a good value for the number of clusters with an internal measure remains an open research problem.

7.2.2 Relative Evaluation Measures

Relative evaluation measures compare two partitions, one being a clustering result and the other either also a clustering result or given by a classification or a human expert defined grouping. In the latter case one also speaks of *external measures* [Halkidi *et al.* 2002a, Halkidi *et al.* 2002b], although the methods to compare the partitions are usually the same: in both cases we are given two partition matrices and have to determine how similar they are. Two clustering results, however, may also be compared based on the cluster parameters alone, although I do not discuss such methods here.

I usually assume that the two partition matrices to compare have the same number of rows, that is, refer to the same number of clusters or classes. However, it is also imaginable to compare matrices with different numbers of rows, although some measures may give misleading results in this case, since they are based on the assumption that it is possible to set up a bijective (i.e., one-to-one) mapping between the cluster/classes of the two partitions.

Regardless of whether the numbers of rows coincide or not, we face the general problem of relating the clusters/classes of the one partition to the clusters/classes of the other partition. There are basically three solutions to this problem: (1) for each cluster/class in the one partition we determine the *best fitting* cluster/class in the other, (2) we find the *best permutation* of the rows of one partition matrix, that is, the best one-to-one mapping of the clusters, or (3) we compare the partition matrices indirectly by first setting up a *coincidence matrix* for each of them, which records for each pair of data points whether they are assigned to the same cluster or not, and then compare the coincidence matrices.

The first alternative has the advantage of being quite efficient (time complexity $O(nc^2)$), but the severe disadvantage that we cannot make sure that we obtain a one-to-one relationship. Some clusters/classes in the second partition may not be paired with any cluster/class in the first, which also renders the approach asymmetric. The second alternative has the advantage that it definitely finds the best one-to-one relationship. Its disadvantage is the fairly high computational cost (time complexity $O(nc^2+c^3)$, see below). The third alternative has the disadvantages that it does not yield a direct indication of how to relate the clusters to each other and that it can have fairly high computational costs (time complexity $O(n^2c)$), especially for a large number of data points. However, the fact that it does not need an explicit mapping between the clusters can also be seen as an advantage, because it renders this method very flexible. In particular, this method is well suited to compare partitions with different numbers of clusters/classes.

Fuzzy or Probabilistic Membership Degrees

All approaches listed above were originally developed for crisp partitions, that is, for partition matrices $\mathbf{U}$, whose elements u_{ij}, $1 \leq i \leq c$, $1 \leq j \leq n$, are either 0 or 1. However, it is also possible to extend them to fuzzy or probabilistic partition matrices, namely by drawing on **fuzzy set theory** [Klir and Yuan 1997, Böhme 1994, Kruse *et al.* 1994] and in particular the theory of *t-norms*, *t-conorms*, and *fuzzy negations* for the combination of membership degrees or probabilities. The general idea of this extension is that the measures discussed in the following can be seen as based on counting how often some logical expression about the elements of the partition matrix or the coincidence matrix is true. Even the elements of the coincidence matrix itself can be seen as derived by logical expressions. Hence it is proper to briefly review the operators of fuzzy logic here.

A **fuzzy negation** is a function $\sim : [0,1] \to [0,1]$ that behaves like a logical "not". This condition already sets the frame conditions, namely that $\sim 0 = 1$ and $\sim 1 = 0$. In addition, it is natural to require the function to be monotonously decreasing, that is, $\forall a, b \in [0,1] : a < b \Rightarrow \sim a > \sim b$. In addition, one may require $\sim\sim a \equiv a$ and continuity. However, even though these conditions do not uniquely determine a fuzzy negation and several families of fuzzy negation operators have been proposed, in practice only the standard negation $\sim a \equiv 1 - a$ [Zadeh 1965] is used.

A **fuzzy conjunction** or **t-norm** is a function $\top : [0,1]^2 \to [0,1]$ that behaves like a logical "and". This already fixes the function values at the corner points, that is, $\top(1,1) = 1$ and $\top(0,0) = \top(0,1) = \top(1,0) = 0$.

In addition, $\top$ should be commutative, associative, and monotonously non-decreasing, that is, $\forall a, b, c \in [0,1] : a \leq b \Rightarrow \top(a,c) \leq \top(b,c)$. Together with the values at the corner points it follows from the monotony condition that $\forall a \in [0,1] : \top(0,a) = \top(a,0) = 0$. On the other two sides of the unit square it is usually defined that $\top(1,a) = \top(a,1) = a$. All conditions together define the class of t-norms. Further conditions, like continuity and sub-idempotency (that is, $\top(a,a) < a$) may be introduced to define specific subclasses (like *Archimedic t-norms* [Klir and Yuan 1997, Kruse *et al.* 1994]), but I do not consider those here. Well-known t-norms are

$$\begin{array}{lcll} \top_{\min}(a,b) & = & \min\{a,b\}, & \\ \top_{\text{prod}}(a,b) & = & ab, & \text{or} \\ \top_{\text{Luka}}(a,b) & = & \max\{0, a+b-1\}. & \end{array}$$

Note that in a probabilistic context $\top_{\text{prod}}$ is usually most appropriate, even though it implicitly assumes independence.

A **fuzzy disjunction** or **t-conorm** is a function $\perp : [0,1]^2 \to [0,1]$ that behaves like a logical "or". As for t-norms, this already fixes the function values at the corner points, that is, $\perp(1,1) = \perp(0,1) = \perp(1,0) = 1$ and $\perp(0,0) = 0$. In analogy to t-norms, $\perp$ should be commutative, associative, and monotonously non-decreasing, that is, $\forall a, b, c \in [0,1] : a \leq b \Rightarrow \perp(a,c) \leq \perp(b,c)$. Together with the values at the corner points it follows from the monotony condition that $\forall a \in [0,1] : \perp(1,a) = \perp(a,1) = 1$. On the other two sides of the unit square it is usually defined that $\perp(0,a) = \perp(a,0) = a$. All conditions together define the class of t-conorms. As for t-norms, further conditions, like continuity and super-idempotency (that is, $\perp(a,a) > a$) may be introduced to define specific subclasses (like *Archimedic t-conorms* [Klir and Yuan 1997, Kruse *et al.* 1994]), but again I do not consider those here. Well-known t-conorms are

$$\begin{array}{lcll} \perp_{\max}(a,b) & = & \max\{a,b\}, & \\ \perp_{\text{prod}}(a,b) & = & a+b-ab, & \text{or} \\ \perp_{\text{Luka}}(a,b) & = & \min\{1, a+b\}. & \end{array}$$

Note that in a probabilistic context $\perp_{\text{prod}}$ is usually most appropriate, even though it implicitly assumes independence.

In the following it is mostly fuzzy negation and fuzzy conjunction I need. I write the formulae generally with a product if a conjunction is needed, because this is the most natural, not only in a probabilistic, but also in a crisp scenario. However, it should be kept in mind that this product may be replaced by other t-norms. For the negation I generally use the standard negation $\sim a \equiv 1 - a$, which is predominant even in fuzzy set theory.

	$u_{kj}^{(2)} = 1$	$u_{kj}^{(2)} = 0$	Σ
$u_{ij}^{(1)} = 1$	$n_{11}^{(i,k)}$	$n_{10}^{(i,k)}$	$n_{1.}^{(i,k)}$
$u_{ij}^{(1)} = 0$	$n_{01}^{(i,k)}$	$n_{00}^{(i,k)}$	$n_{0.}^{(i,k)}$
Σ	$n_{.1}^{(i,k)}$	$n_{.0}^{(i,k)}$	n

Table 7.2: Contingency table for comparing two rows of two partition matrices.

Comparing Partition Matrices

The first two approaches outlined at the beginning of this section directly compare two $c \times n$ partition matrices $\mathbf{U}^{(1)}$ and $\mathbf{U}^{(2)}$. For both of them we need a measure that compares two rows, one from each matrix. Such measures are basically the same as those used in Section 7.1 for comparing a classifier output to the class labels of a data set. Formally, we set up a 2×2 contingency table for each pair of rows, one from each matrix. (cf. Table 7.2). This is done in basically the same way as in Section 7.1.3. That is, for each pair $(i, k) \in \{1, \ldots, c\}^2$ we compute

$$
\begin{aligned}
n_{11}^{(i,k)}\big(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}\big) &= \sum_{j=1}^{n} u_{ij}^{(1)} \cdot u_{kj}^{(2)}, \\
n_{01}^{(i,k)}\big(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}\big) &= \sum_{j=1}^{n} \Big(1 - u_{ij}^{(1)}\Big) \cdot u_{kj}^{(2)}, \\
n_{10}^{(i,k)}\big(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}\big) &= \sum_{j=1}^{n} u_{ij}^{(1)} \cdot \Big(1 - u_{kj}^{(2)}\Big), \\
n_{00}^{(i,k)}\big(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}\big) &= \sum_{j=1}^{n} \Big(1 - u_{ij}^{(1)}\Big) \cdot \Big(1 - u_{kj}^{(2)}\Big).
\end{aligned}
$$

As in Section 7.1.3, I drop the arguments $\mathbf{U}^{(1)}$ and $\mathbf{U}^{(2)}$ in the following to make the formulae easier to read. Likewise, it is convenient to introduce the following abbreviations for the row and column sums (cf. Table 7.1)

$$
n_{1.}^{(i,k)} = n_{11}^{(i,k)} + n_{10}^{(i,k)}, \qquad n_{0.}^{(i,k)} = n_{01}^{(i,k)} + n_{00}^{(i,k)},
$$
$$
n_{.1}^{(i,k)} = n_{11}^{(i,k)} + n_{01}^{(i,k)}, \qquad n_{.0}^{(i,k)} = n_{10}^{(i,k)} + n_{00}^{(i,k)}.
$$

(Note that all these numbers may also be computed from fuzzy or probabilistic membership degrees, even though I assume $u_{ij} \in \{0, 1\}$ here.)

From these number we may now compute the same measures as in Section 7.1.3. The only differences consist in how the rows are paired in the two approaches outlined at the beginning of this section. If we try to find the best matching row in the partition matrix $\mathbf{U}^{(2)}$ for each row in the partition matrix $\mathbf{U}^{(1)}$, we may compute, for example, (macro-averaged) precision as

$$\pi = \sum_{i=1}^{c} \max_{1 \le k \le c} \pi_{i,k} = \sum_{i=1}^{c} \max_{1 \le k \le c} \frac{n_{11}^{(i,k)}}{n_{.1}^{(i,k)}} = \sum_{i=1}^{c} \max_{1 \le k \le c} \frac{n_{11}^{(i,k)}}{n_{01}^{(i,k)} + n_{11}^{(i,k)}}$$

and (macro-averaged) recall as

$$\rho = \sum_{i=1}^{c} \max_{1 \le k \le c} \pi_{i,k} = \sum_{i=1}^{c} \max_{1 \le k \le c} \frac{n_{11}^{(i,k)}}{n_{1.}^{(i,k)}} = \sum_{i=1}^{c} \max_{1 \le k \le c} \frac{n_{11}^{(i,k)}}{n_{10}^{(i,k)} + n_{11}^{(i,k)}}.$$

However, since both precision and recall are needed for the assessment of how similar the two partitions are, such an approach is somewhat dubious: it may be that for a given row i of the first matrix precision and recall may draw on different rows k of the second matrix. In such a case the closeness of the partition would be overrated, because then there is no one-to-one mapping that achieves the computed values of precision and recall. Rather all assessments computed from one-to-one mappings would either have a lower precision or a lower recall. Therefore combining the values of the above two formulae into an F_1 measure (cf. page 207) is not advisable.

Unfortunately, this problem cannot be resolved by applying the second approach outlined at the beginning of this section, that is, by computing the maximum over all possible one-to-one mappings of the matrix rows. In this case (macro-averaged) precision would be computed as

$$\pi = \max_{\varsigma \in \Pi(c)} \sum_{i=1}^{c} \pi_{i,\varsigma(i)} = \max_{\varsigma \in \Pi(c)} \sum_{i=1}^{c} \frac{n_{11}^{(i,\varsigma(i))}}{n_{.1}^{(i,\varsigma(i))}} = \max_{\varsigma \in \Pi(c)} \sum_{i=1}^{c} \frac{n_{11}^{(i,\varsigma(i))}}{n_{01}^{(i,\varsigma(i))} + n_{11}^{(i,\varsigma(i))}}$$

and (macro-averaged) recall as

$$\rho = \max_{\varsigma \in \Pi(c)} \sum_{i=1}^{c} \pi_{i,\varsigma(i)} = \max_{\varsigma \in \Pi(c)} \sum_{i=1}^{c} \frac{n_{11}^{(i,\pi(i))}}{n_{1.}^{(i,\varsigma(i))}} = \max_{\varsigma \in \Pi(c)} \sum_{i=1}^{c} \frac{n_{11}^{(i,\varsigma(i))}}{n_{10}^{(i,\varsigma(i))} + n_{11}^{(i,\varsigma(i))}},$$

where $\Pi(c)$ is the set of all permutations of the c numbers $\{1, \ldots, c\}$.[6] Again we cannot be sure that the best precision value results for the same permutation as the best recall value. A useful result can only be obtained if a

[6] Note that with the so-called *Hungarian method* for solving optimum weighted bipartite matching problems [Papadimitriou and Steiglitz 1982] the time complexity of finding the maximum for given pairwise precision and recall values is only $O(c^3)$ and not $O(c!)$.

single assessment value is maximized over all permutations. Examples are the **$\mathbf{F_1}$ measure**

$$F_1 = \max_{\varsigma \in \Pi(c)} \sum_{i=1}^{c} \frac{2\,\pi_{i,\varsigma(i)}\rho_{i,\varsigma(i)}}{\pi_{i,\varsigma(i)} + \rho_{i,\varsigma(i)}},$$

(compare page 207) where class-specific precision and recall are

$$\pi_{i,k} = \frac{n_{11}^{(i,k)}}{n_{\cdot 1}^{(i,k)}} = \frac{n_{11}^{(i,k)}}{n_{01}^{(i,k)} + n_{11}^{(i,k)}} \qquad \text{and} \qquad \rho_{i,k} = \frac{n_{11}^{(i,k)}}{n_{1\cdot}^{(i,k)}} = \frac{n_{11}^{(i,k)}}{n_{10}^{(i,k)} + n_{11}^{(i,k)}},$$

and the **(cross-classification) accuracy**

$$Q_{\text{acc}}\big(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}\big) = \max_{\varsigma \in \Pi(c)} \frac{1}{n} \sum_{i=1}^{c} \left(n_{00}^{(i,\varsigma(i))} + n_{11}^{(i,\varsigma(i))} \right).$$

Generally two partition matrices $\mathbf{U}^{(1)}$ and $\mathbf{U}^{(2)}$ are the more similar, the higher the values of the F_1 measure or the cross-classification accuracy.

Comparing Coincidence Matrices

As an alternative to comparing the partition matrices directly, one may first compute from each of them an $n \times n$ **coincidence matrix**, also called a **cluster connectivity matrix** [Levine and Domany 2001], which states for each pair of data points whether they are assigned to the same cluster or not. Formally, a coincidence matrix $\mathbf{\Psi} = (\psi_{jl})_{1 \leq j,l \leq n}$ can be computed from a partition matrix $\mathbf{U} = (u_{ij})_{1 \leq i \leq c, 1 \leq j \leq n}$ by

$$\psi_{jl} = \sum_{i=1}^{c} u_{ij} u_{il}.$$

Note that this computation may also be carried out with fuzzy or probabilistic membership degrees, possibly replacing the product by another t-norm (cf. the general discussion on page 222).

After coincidence matrices $\mathbf{\Psi}^{(1)}$ and $\mathbf{\Psi}^{(2)}$ are computed from the two partition matrices $\mathbf{U}^{(1)}$ and $\mathbf{U}^{(2)}$, the comparison is carried out by computing statistics of the number of data point pairs that are in the same group in both partitions, in the same group in one, but in different groups in the other, or in different groups in both. The main advantage of this approach is, of course, that we are freed of the need to pair the groups of the two partitions. We rather exploit that data points that are considered (dis)similar by one partition should also be considered (dis)similar by the other.

	$\psi_{jl}^{(2)}=1$	$\psi_{jl}^{(2)}=0$	Σ
$\psi_{jl}^{(1)}=1$	N_{SS}	N_{SD}	$N_{S.}$
$\psi_{jl}^{(1)}=0$	N_{DS}	N_{DD}	$N_{D.}$
Σ	$N_{.S}$	$N_{.D}$	$N_{..}$

Table 7.3: Contingency table for the computation of several clustering comparison indices.

Formally, we compute a 2×2 contingency table (cf. Table 7.3) containing the numbers (which are basically counts of the different pairs $\left(\psi_{jl}^{(1)},\psi_{jl}^{(2)}\right)$)

$$\begin{aligned}
N_{SS}\left(\mathbf{\Psi}^{(1)},\mathbf{\Psi}^{(2)}\right) &= \sum_{j=2}^{n}\sum_{l=1}^{j-1}\psi_{jl}^{(1)}\psi_{jl}^{(2)},\\
N_{SD}\left(\mathbf{\Psi}^{(1)},\mathbf{\Psi}^{(2)}\right) &= \sum_{j=2}^{n}\sum_{l=1}^{j-1}\psi_{jl}^{(1)}\left(1-\psi_{jl}^{(2)}\right),\\
N_{DS}\left(\mathbf{\Psi}^{(1)},\mathbf{\Psi}^{(2)}\right) &= \sum_{j=2}^{n}\sum_{l=1}^{j-1}\left(1-\psi_{jl}^{(1)}\right)\psi_{jl}^{(2)},\\
N_{DD}\left(\mathbf{\Psi}^{(1)},\mathbf{\Psi}^{(2)}\right) &= \sum_{j=2}^{n}\sum_{l=1}^{j-1}\left(1-\psi_{jl}^{(1)}\right)\left(1-\psi_{jl}^{(2)}\right),
\end{aligned}$$

where the index S stands for "same group" and the index D stands for "different groups" and the two indices refer to the two partitions. To make the formulae easier to read, the arguments $\mathbf{\Psi}^{(1)}$ and $\mathbf{\Psi}^{(2)}$ are dropped in the following. In addition, it is convenient to define the following abbreviations

$$N_{S.}=N_{SS}+N_{SD},\qquad N_{D.}=N_{DS}+N_{DD},$$

$$N_{.S}=N_{SS}+N_{DS},\qquad N_{.D}=N_{SD}+N_{DD},$$

$$N_{..}=N_{SS}+N_{SD}+N_{DS}+N_{DD}=\frac{n(n-1)}{2}.$$

From these number a large variety of measures may be computed. Well-known examples include the **Rand statistic**

$$Q_{\text{Rand}}\left(\mathbf{\Psi}^{(1)},\mathbf{\Psi}^{(2)}\right)=\frac{N_{SS}+N_{DD}}{N_{..}}=\frac{N_{SS}+N_{DD}}{N_{SS}+N_{SD}+N_{DS}+N_{DD}},$$

which is a simple ratio of the number of data point pairs treated the same in both partitions to all data point pairs, and the **Jaccard coefficient**

$$Q_{\text{Jaccard}}\big(\mathbf{\Psi}^{(1)}, \mathbf{\Psi}^{(2)}\big) = \frac{N_{SS}}{N_{SS} + N_{SD} + N_{DS}},$$

which ignores negative information, that is, pairs that are assigned to different groups in both partitions. Both measures are to be maximized. Another frequently encountered measure is the **Folkes–Mallows index**

$$Q_{\text{FM}}\big(\mathbf{\Psi}^{(1)}, \mathbf{\Psi}^{(2)}\big) = \frac{N_{SS}}{\sqrt{N_{S\cdot}N_{\cdot S}}} = \sqrt{\frac{N_{SS}}{N_{SS} + N_{SD}} \cdot \frac{N_{SS}}{N_{SS} + N_{DS}}},$$

which can be interpreted as a cosine similarity measure, because it computes the cosine between two binary vectors, each of which contains all elements of one of the two coincidence matrices $\mathbf{\Psi}^{(1)}$ and $\mathbf{\Psi}^{(2)}$. Consequently, this measure is also to be maximized. A final example is the **Hubert index**

$$Q_{\text{Hubert}}\big(\mathbf{\Psi}^{(1)}, \mathbf{\Psi}^{(2)}\big) = \frac{N_{\cdot\cdot}N_{SS} - N_{S\cdot}N_{\cdot S}}{\sqrt{N_{S\cdot}N_{\cdot S}N_{D\cdot}N_{\cdot D}}},$$

which may either be interpreted as a product-moment correlation, computed from the set of pairs $\big(\psi_{jl}^{(1)}, \psi_{jl}^{(2)}\big)$, $1 \leq j, l \leq n$. Alternatively, it may be interpreted as the square root of the (normalized) χ^2 measure, as it can be computed from the 2×2 contingency table shown in Table 7.3. The χ^2 measure can be seen as measuring the strength of dependence between two random variables, one for each partition, which indicate for each data point pair whether the data points are in the same group or not.[7]

It should be clear that this list does not exhaust all possibilities. Basically all measures by which (binary) vectors and matrices can be compared are applicable, and these abound.

7.2.3 Resampling

Resampling [Good 1999] can be seen as a special **Monte Carlo method**, that is, as a method for finding solutions to mathematical and statistical problems by simulation [Everitt 1998, Halkidi *et al.* 2002a]. It has been applied to cluster estimation problems already fairly early [Jain and Moreau

[7] A detailed explanation of the general χ^2 measure can be found, for instance, in [Borgelt and Kruse 2002]. A derivation of the special formula for 2×2 contingency tables referred to here can be found in Section A.11 in the appendix.

1986, Breckenridge 1989] and it seems to have gained increased attention in this domain recently [Levine and Domany 2001, Roth *et al.* 2002, Law and Jain 2003]. Its main purpose in clustering is the validation of clustering results as well as the selection of an appropriate cluster model—in particular the choice of an appropriate number of clusters—by estimating the variability (or, equivalently, the stability) of the clustering result.

Resampling methods can be found with two sampling strategies. In the first place, one may use **subsampling**, that is, the samples are drawn without replacement from the given data set, so that each data point appears in at most one data subset. This strategy is usually applied in a cross validation style (cf. Section 7.1.2), that is, the given data set is split into a certain number of disjoint subsets. The alternative is **bootstrapping** [Efron and Tibshirani 1993], in which samples are drawn with replacement, so that a data point may even appear multiple times in the same data subset. There are good arguments in favor and against both approaches.

The general idea of applying resampling for cluster validation and model selection is as follows: a cluster model can usually be applied as a classifier with as many classes as there are clusters (i.e. one class per cluster). In the case of a prototype-based cluster model the natural approach is a *nearest prototype* or *maximum membership classifier* (cf. Section 2.5), in which each data point is assigned to the most similar prototype. In this way data points that have not been used to build the cluster model can be assigned to clusters (or the corresponding classes). Thus we obtain, with the same algorithm, two different groupings of the same set of data points. For example, one may be obtained by clustering the data set, the other by applying a cluster model that was built on another data set. These two groupings can be compared using, for example, one of the measures discussed in the preceding section. By repeating such comparisons with several samples drawn from the original data set, one can obtain an assessment of the variability of the cluster structure (or, more precisely, an assessment of the variability of the evaluation measure for the similarity of partitions).

Specific algorithms following this general scheme have been proposed by [Levine and Domany 2001, Roth *et al.* 2002, Law and Jain 2003]. The approaches by [Levine and Domany 2001] and [Law and Jain 2003] are basically identical. Both are based on a bootstrapping approach and work as follows: first the full given data set is clustered with the chosen algorithm. Formally, this may be seen as an estimate of the "average" partition [Law and Jain 2003]. Then a user-defined number of random samples of user-defined size are drawn (with replacement) from the data set and clustered as well. The cluster models obtained from the samples are applied to the full

data set, thus obtaining two groupings of this data set. These two groupings are compared by one of the relative evaluation measures based on coincidence matrices that were discussed in Section 7.2.2. Finally, the average of the evaluation measure values for each of these comparisons is taken as an assessment of the cluster variability. As an alternative, [Law and Jain 2003] mention that one may do without an estimate for the "average" partition (which is estimated by the cluster model obtained from the full data set) and rather assess the variability of the cluster structures by comparing all pairs of cluster models obtained from the samples on the full data set.

This resampling approach may be applied to select the most appropriate cluster model, in particular, the "best" number of clusters, by executing the above algorithm for different parameterizations of the clustering algorithm and then to select the one showing the lowest variability. Experimental results reported by [Law and Jain 2003] indicate that this approach is very robust and a fairly reliable way of choosing the number of clusters.

In contrast to the bootstrapping approaches, [Roth *et al.* 2002] rely on a (repeated) two-fold cross validation sampling scheme. In each step the given data set is split randomly into two parts of about equal size. Both parts are processed with the same clustering algorithm and the cluster model obtained on the second half of the data is applied to the first half. Thus one obtains two groupings for the first half of the data, which are compared with a risk-based evaluation measure. This (relative) measure is defined on the two partition matrices and thus has to find the best matching of the clusters of the two groupings (see above). However, in principle all relative measures discussed in the preceding section (including those based on coincidence matrices) may be applied (just as measures based on partition matrices may be applied in the bootstrapping approaches by [Levine and Domany 2001, Law and Jain 2003]). [Roth *et al.* 2002] report experimental results on several data sets, which show that the number of clusters can be selected in a fairly reliable way with this approach.

For applications of these resampling methods it should be noted that all approaches in this direction only assess the variability in the results obtained with some clustering algorithm. Although a low variability is surely a highly desirable property, it is not sufficient to guarantee a good clustering result. For example, a clustering algorithm that always yields the same partition of the data space, regardless of the data it is provided with, has no variability at all, but surely yields unsatisfactory clustering results [Law and Jain 2003]. Hence the clustering algorithms that are compared with such schemes should not differ too much in their flexibility, because otherwise the simpler and thus more stable algorithm may be judged superior without actually being.

Furthermore, [Halkidi *et al.* 2002a, Halkidi *et al.* 2002b] remark that the power of many such statistical tests, like the estimation of the variability of the clustering structure as it was discussed above, decreases quickly with increasing data dimensionality. This is not surprising, because due to what is usually called the *curse of dimensionality*, the data space necessarily is less and less densely populated, the more dimensions there are. In addition, the noise in the different dimensions tends to sum, which in combination with the tendency of larger average distances between the data points [Döring *et al.* 2005], makes it more and more difficult for a clustering algorithm to find reasonable groups in the data. This, of course, must lead to a higher variability in the clustering result. For low-dimensional data sets, however, resampling is a very powerful technique and seems to be the best available approach to determine the number of clusters.

Chapter 8

Experiments and Applications

Although data analysis methods should have a sound mathematical basis, the ultimate test of their quality is whether they yield useful results in practice. While the preceding chapters were all devoted to a (theoretical) review and development of clustering and classification methods, this chapter reports experiments on well-known data sets that were carried out with an implementation of these methods. In addition, I describe an application of the discussed methods to document clustering. This application needs some additional considerations concerning, in particular, how to code documents so that clustering algorithms become applicable. The needed theoretical background and preprocessing steps are reviewed briefly in Section 8.3.

All experiments reported here were carried out with the programs that can be found on my WWW page[1], which was already mentioned in Section 1.4. The `cluster` package contains programs for expectation maximization and fuzzy clustering, which also offer a learning vector quantization mode. In addition, the `lvq` package contains a special learning vector quantization program, which provides some special features (like, for example, the radius adaptation methods discussed at the beginning of Section 5.3.3). Finally, the `rbf` package contains programs for training radial basis function neural networks, which can be initialized with a clustering result. In addition, several of the simpler initialization methods described in Chapter 4 are available in all of the mentioned programs.

[1] `http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html`

8.1 Regularization

In this section I report experiments that demonstrate the usefulness of the regularization methods that I suggested in Section 6.1.2 and 6.1.3, respectively [Borgelt and Kruse 2004, Borgelt and Kruse 2005]. As test cases I use well-known data sets from the UCI machine learning repository [Blake and Merz 1998]. In all experiments each dimension of the data set is normalized to mean value 0 and standard deviation 1 in order to avoid any distortions that may be caused by a different scaling of the coordinate axes.

As a first illustrative example, I consider the result of clustering the *iris data* [Anderson 1935, Fisher 1936, Blake and Merz 1998] (excluding, of course, the class attribute), which was already mentioned in Section 4.2. This data set was clustered with the Gustafson–Kessel fuzzy clustering algorithm [2]—that is, using an alternating optimization approach—with the aim of finding three clusters: one cluster per class. In this algorithm each cluster is endowed with a covariance matrix describing its shape (cf. Section 5.2.3). The size of the clusters (measured as the isotropic radius, that is, $\kappa = 1$, cf. Section 2.3) was set to a fixed value of 0.4 (since all dimensions are normalized to mean value 0 and standard deviation 1, 0.4 is a good size of a cluster if three clusters are to be found). Note, however, that this size only influences how the clusters are described numerically, but not the degrees of membership a data point has to the different clusters. The reason is that in the formula for computing the membership degrees a possible size factor cancels if the Cauchy function with $a = 2$ and $b = 0$ is used (cf. Section 5.2.3). As a consequence the same membership degrees result independent of the chosen cluster size. Hence I use such a size value only to obtain a better visualization of the clustering result.

The result without shape regularization is shown in Figure 8.1 on the left. Due to a few data points located in a thin diagonal cloud on the upper right border on the figure, the middle cluster is drawn into a fairly long ellipsoid. Although this shape minimizes the objective function, it may not be a desirable result, because the cluster structure is not compact enough. Another argument supporting this assessment is that it does not capture the class structure well (cf. Figure 4.2 on page 80, which shows the classes of the data points in a similar figure). Computing the reclassification accuracy[3]

[2] The options used for the clustering program `cli` are `-Vc3I0.4`.

[3] The reclassification accuracy can be computed by executing the cluster model with the program `clx` from the `cluster` package, using the option `-c` to obtain a hard classification. The resulting output table is then processed with the program `xmat` from the `table` package, using the option `-c` to find the best assignment of clusters to classes.

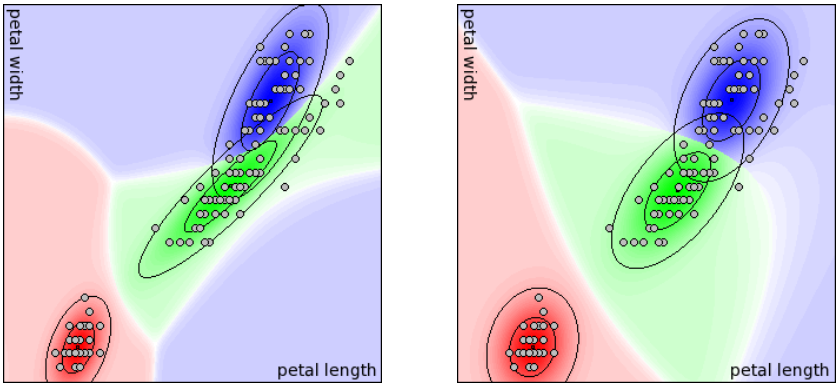

Figure 8.1: Result of the Gustafson-Kessel algorithm on the iris data with fixed equal cluster size of $\sigma = 0.4$ without (left) and with shape regularization (right, method 2 with $r = 4$). Clustering was done on all four attributes (that is, petal length and width and sepal length and width).

confirms this visual impression: a classifier based on this cluster structure yields 15 misclassifications (10%), whereas a Bayes classifier yields only 6 (4%, naïve Bayes classifier—axes-parallel ellipsoids) or 3 misclassifications (2%, full Bayes classifier—general ellipsoids, cf. page 80).

Using shape regularization method 2 (cf. page 185 in Section 6.1.2) with $r = 4$ the cluster structure shown on the right in Figure 8.1 is obtained. In this result the clusters are more compact and resemble the class structure of the data set. This impression is confirmed by the reclassification accuracy: with this cluster model, if it is used as a classifier, only 3 cases are misclassified, and thus the same quality is achieved as with a full Bayes classifier. This is particularly remarkable if one considers that the clustering approach does not use any class information about the data points.

As another example let us consider the result of clustering the *wine data* [Blake and Merz 1998], which describes 178 Italian wines from three different cultivation regions by 13 numerical attributes obtained from a chemical analysis. This data set was clustered with the fuzzy maximum likelihood estimation (FMLE) algorithm using three clusters of variable shape, size, and weight. In this experiment I used attributes 7, 10, and 13, which are among the most informative w.r.t. the class assignments, where the classes are the cultivation areas. One result without size regularization is shown

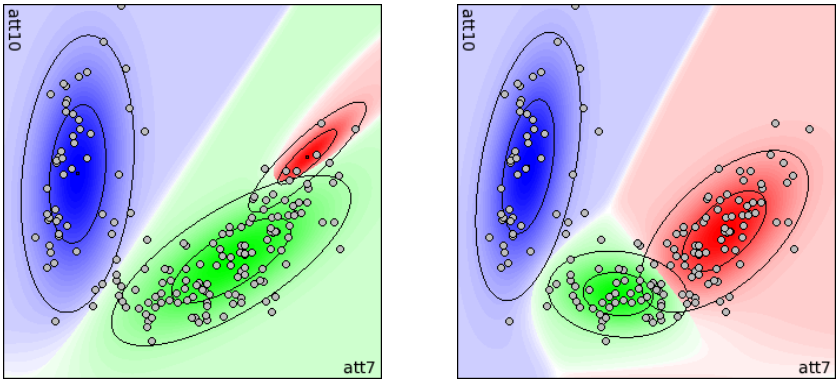

Figure 8.2: Result of the fuzzy maximum likelihood estimation (FMLE) algorithm on the wine data without (left) and with size regularization (right, method 3 with $r = 2$). Clustering was done on attributes 7, 10, and 13.

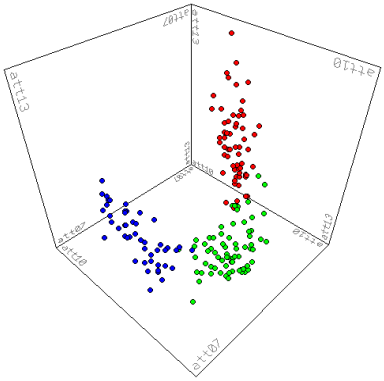

Figure 8.3: A three-dimensional scatter plot of the wine data in the space spanned by the attributes 7, 10, and 13. In this view it is well visible that the three classes have a fairly clear cluster structure, even though the classes are not cleanly separated from each other.

in Figure 8.2 on the left.[4] However, the algorithm is much too unstable to present a unique result. Sometimes clustering fails completely, because one cluster collapses to a single data point—an effect that is due to the steepness of the Gaussian probability density function. Only with a good initialization an acceptable clustering structure is obtained with high probability (which is why it is usually recommended to initialize the fuzzy maximum likelihood estimation (FMLE) algorithm it with the fuzzy c-means algorithm).

[4]The options used for the clustering program `cli` are `-wZVGNc3`.

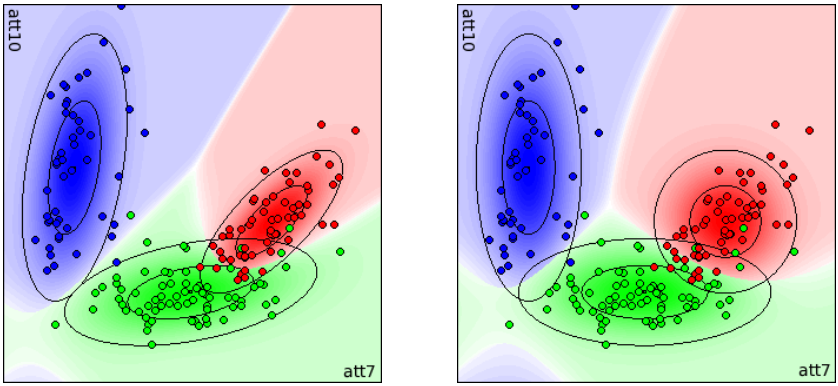

Figure 8.4: The probability density functions used by a full Bayes classifier (general ellipses, left) and a naïve Bayes classifier (axis-parallel ellipses, right) for the wine data. The ellipses are the 1σ- and 2σ-boundaries.

The result also does not capture the class structure well, even though the classes form fairly clear clusters, as can easily be seen in a three-dimensional scatter plot (cf. Figure 8.3). However, Figure 8.4, in which the labeled data points are shown in the same space as the clustering results (attributes 7 and 10), makes it easier to compare the clustering result to the class assignments. The visual impression obtained by this comparison is confirmed by computing the reclassification accuracy, which yields 61(!) errors. In contrast to this, a naïve Bayes classifier misclassifies only 8 cases (cf. Figure 8.4 on the right), a full Bayes classifier even only 6 (cf. Figure 8.4 on the left).

The situation is improved with size regularization, the most frequent result of which (which sometimes, with a fortunate initialization, can also be achieved without) is shown on the right in Figure 8.2. It was obtained with method 3 with $r = 2$. Using this result as a classifier yields 21 misclassifications, which is still not too good compared to the Bayes classifier results, but considerably better than the result shown in Figure 8.2 on the left. Another result that is also often obtained (though less frequently than the one shown in Figure 8.2 on the right) is depicted in Figure 8.5. It captures the class structure very well and misclassifies only 7 cases, which is competitive to the Bayes classifier results. Again one should bear in mind that the cluster structure is computed without any information about the class labels. Therefore this is fairly remarkable result.

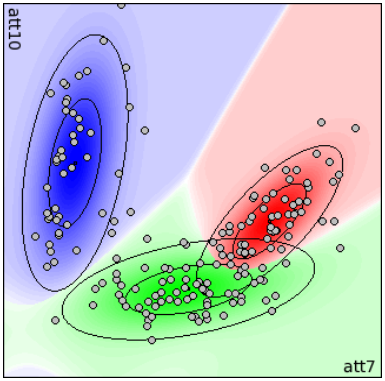

Figure 8.5: Alternative result of the fuzzy maximum likelihood estimation (FMLE) algorithm on the wine data, which can be obtained with and without regularization. It captures the class structure even better than the most frequent result.

Summarizing, it is justified to say that, although the results are still not unique and sometimes clusters still focus on very few data points, the fuzzy maximum likelihood estimation (FMLE) algorithm becomes more stable by using size regularization and reasonable results are obtained more often than without size regularization. Hence we can conclude that size regularization considerably improves the robustness of this algorithm.

Even better results can be obtained if size regularization is combined with weight regularization. The reason is that, depending on the data distribution, equalizing the sizes of the clusters not necessarily also equalizes their weights, so that the effect on the full cluster model is too small. Combined size and weight regularization turn the fuzzy maximum likelihood estimation (FMLE) algorithm into a fairly stable clustering method.

The expectation maximization algorithm, on the other hand, does not profit as much from such modifications, mainly because it is already fairly stable without. Results like the one shown in Figure 8.2 on the left almost never occur, as experiments reported in [Döring *et al.* 2004] indicate. The reason seems to be that the *weighting exponent* or *fuzzifier* that is used in the fuzzy maximum likelihood estimation (FMLE) algorithm, has a deteriorating effect, in particular w.r.t. the cluster size and weight, which is mitigated by regularization. Expectation maximization, which does not use such an exponent, seems to have a lesser tendency towards extreme cluster sizes or weights. However, size and weight regularization with method 3 or 2, respectively, do not introduce any changes as long as the size and weight relations are within the user-specified bounds, and only interfere with the clustering process when extreme results are about to develop. Hence they are harmless and thus generally recommendable modifications.

8.2 Acceleration

In this section I present experimental results of fuzzy clustering algorithms modified with the acceleration techniques that I discussed in Section 6.2 [Borgelt and Kruse 2003]. As in the preceding section I tested these methods on well-known data sets from the UCI machine learning repository [Blake and Merz 1998]: abalone (physical measurements of abalone clams), breast (Wisconsin breast cancer, tissue and cell measurements), iris (petal and sepal length and width of iris flowers, see also Section 4.2), and wine (chemical analysis of Italian wines from different regions, see also the preceding section). In order to avoid scaling effects, all data was normalized, so that in each dimension the mean value was 0 and the standard deviation 1.

Since the listed data sets are originally classified (although I did not use the class information in the clustering process), one knows the number of clusters to find (abalone: 3 classes, breast: 2 classes, iris: 3 classes, wine: 3 classes). Therefore I ran the clustering algorithms using these numbers. In addition, I ran the algorithms with 6 clusters for the abalone and the wine data set, for which these numbers also yield reasonable results.

In all experiments the iterative updating was terminated when a (normal, that is, unmodified alternating optimization) update step changed no center coordinate by more than 10^{-6}. That is, regardless of the update step modification, I used the normal update step to define the termination criterion in order to make the results comparable over the different acceleration methods. Note that for the Gustafson–Kessel algorithm (normal and axes-parallel version) I did not consider the change of the elements of the covariance matrices for the termination criterion. Doing so would have introduced difficulties when comparing the results obtained with it to the results obtained with the fuzzy c-means algorithm. It should also be noted that I used shape regularization (cf. Section 6.1.2) method 2 with $r = 4$ to stabilize the results obtained with the Gustafson–Kessel algorithm.

Most of the methods I explored in Section 6.2 have parameters that influence their behavior. My experiments showed that the exact values of these parameters are important only for the step width expansion and the momentum term, for which the best value seems to depend on the data set. Therefore I ran the clustering algorithm for different values of the expansion factor η (cf. Section 6.2.1) and the momentum factor β (cf. Section 6.2.2). In the following I report only the best results I obtained; the parameter values I employed are listed in Table 8.1. For the self-adaptive learning rate as well as for the resilient approach I used a growth factor of 1.2 and a shrink factor of 0.5, which are very common choices for these parameters.

	fuzzy c-means		axis-parallel GK		Gustafson–Kessel	
dataset	expand	moment	expand	moment	expand	moment
abalone 3	1.50	0.30	1.60	0.45	1.80	0.60
abalone 6	1.80	0.50	1.80	0.70	1.90	0.65
breast 2	1.20	0.05	1.60	0.25	1.50	0.25
iris 3	1.40	0.15	1.60	0.30	1.60	0.25
wine 3	1.40	0.10	1.70	0.30	1.90	0.70
wine 6	1.90	0.75	2.00	0.50	1.90	0.70

Table 8.1: Best parameter values for the different datasets.

In order to remove the influence of the random initialization of the clusters (they are initialized by sampling from the data set, cf. Section 4.2), I ran each method 20 times for each data set and averaged the number of iterations needed. The standard deviation of the individual results from these averages is fairly small, though. A shell script with which the experiments can be repeated is available on my WWW page.[5] Note that the results differ somewhat from the results reported in [Borgelt and Kruse 2003], due to certain changes of the implementation and the experimental setup.

Note also that the modifications of the update steps only affect the recomputation of the cluster centers and not the distance computations for the data points, the latter of which accounts for the greater part of the computational costs of fuzzy clustering. Therefore the increase in computation time for one iteration is negligible and consequently it is justified to compare the different approaches by simply comparing the number of iterations needed to reach a given accuracy of the cluster parameters. Hence all result tables show average numbers of iteration steps.

Table 8.2 shows the results for the fuzzy c-means algorithm (spherical clusters of equal size). Each column shows the results for a method, each line the results for a dataset. The most striking observation to be made about this table is that the analogs of resilient backpropagation and quick backpropagation almost always *increase* the number of iterations needed. Step width expansion, momentum term, and self-adaptive learning rate, however, yield very good results, in some cases even cutting down the number of iterations to less than half of what the standard algorithm needs. Judging from the numbers in this table, the momentum term approach and the self-adapting learning rate appear to perform best. However, it should

[5] `http://fuzzy.cs.uni-magdeburg.de/~borgelt/cluster.html`.

dataset	standard	expand	moment	adaptive	resilient	quick
abalone 3	46.2	31.9	27.5	28.8	56.9	56.2
abalone 6	114.8	72.5	53.1	64.6	87.6	144.8
breast 2	13.2	9.9	11.4	10.6	46.8	13.8
iris 3	26.3	17.8	18.1	16.1	48.1	27.9
wine 3	27.4	20.7	20.6	17.6	52.0	28.9
wine 6	490.2	269.7	112.8	259.4	306.9	430.9

Table 8.2: Results for the fuzzy c-means algorithm.

dataset	standard	expand	moment	adaptive	resilient	quick
abalone 3	51.2	32.5	43.9	50.9	222.9	50.8
abalone 6	140.6	61.8	103.5	139.8	484.8	141.9
breast 2	30.6	21.2	18.7	19.2	83.8	30.6
iris 3	27.1	26.4	26.7	26.6	100.8	27.2
wine 3	32.2	29.1	29.1	29.1	115.9	32.4
wine 6	406.0	359.4	360.6	359.3	failed	384.8

Table 8.3: Results for the axes-parallel Gustafson–Kessel algorithm.

be kept in mind that the best value of the momentum factor β is not known in advance and that these results were obtained using the optimal value. Hence the self-adaptive learning rate approach, which does not need such tuning, has a clear edge over the momentum term approach.

Table 8.3 shows the results for the axes-parallel Gustafson–Kessel algorithm (axes-parallel (hyper-)ellipsoids of equal size) and Table 8.4 the results for the normal Gustafson–Kessel algorithm (equally sized (hyper-)ellipsoids in general orientation). Here the picture is basically the same. The analogs of resilient and quick backpropagation are clear failures (the entry "failed" means that at least in some runs no stable clustering result could be reached even with 10000 iterations). Step width expansion, momentum term, and self-adaptive learning rate, however, yield very good results. In particular for the normal Gustafson–Kessel algorithm the momentum term approach seems to be the clear winner. However, it has to be taken into account that the optimal value of the momentum factor β is not known in advance. Therefore one should prefer the self-adaptive learning rate approach, which consistently leads to significant gains in the number of iterations.

dataset	standard	expand	moment	adaptive	resilient	quick
abalone 3	146.3	83.2	63.0	74.8	205.2	172.3
abalone 6	268.2	136.5	77.2	135.1	failed	247.7
breast 2	37.8	25.9	24.5	21.1	54.1	37.1
iris 3	44.7	26.9	28.4	27.1	308.8	55.6
wine 3	333.8	174.4	92.2	175.0	333.8	325.4
wine 6	358.6	187.5	103.0	185.8	358.6	339.1

Table 8.4: Results for the normal Gustafson–Kessel algorithm.

Summarizing, one has to say that the experimental results show that there is indeed some potential for accelerating the clustering process, especially if a self-adaptive learning rate or a momentum term are used. With these approaches the number of iterations needed until convergence can sometimes be reduced to less than half the number of the standard approach. I recommend the self-adaptive learning rate, but the step width expansion and the momentum term approach also deserve strong interest, especially, because they are so simple to implement. Their drawback is, however, that the best value for their parameter depends on the dataset.

8.3 Clustering Document Collections

In the area of **information retrieval** [Frakes and Baeza-Yates 1992, Jones 1997, Baeza-Yates and Ribeiro-Neto 1999, Grossman and Frieder 2004] one often faces the task to structure a given document collection with the aim of finding groups of similar documents. Such a grouping, once it is determined, can be used to guide a user in his/her search for documents that are relevant for a given task. One way to approach this task is to represent the given text documents by numerical feature vectors that are clustered using, for example, one of the methods discussed in this thesis. In the following I describe experiments in this direction, which were presented first in the sequence of papers [Borgelt and Nürnberger 2004a, Borgelt and Nürnberger 2004b, Borgelt and Nürnberger 2004c]. In these papers we explored whether cluster specific variances (axes-parallel ellipsoids) are useful when clustering document collections, to what extent term re-weighting influences the clustering performance, and whether initial—global or cluster specific—term re-weighting can be used to bias or improve the performance.

8.3.1 Preprocessing the Documents

For both searching in large document collections as well as for structuring them it is necessary to preprocess the documents and to store the resulting features in a data structure that is more appropriate for further processing than unstructured text files. The currently predominant approaches to document representation are the **vector space model** [Salton *et al.* 1975], the **probabilistic model** [Robertson 1977], the **logical model** [Rijsbergen 1986], and the **Bayesian network model** [Turtle and Croft 1990].

Although it lacks explicit semantic information and treats a document as a *bag of words* rather than a *sequence of words*, the vector space model allows for very efficient analysis of huge document collections due to its simple structure. As a consequence it is used in most of the currently available document retrieval systems. The most popular retrieval methods that are based on this model are—besides the direct use of the vector description in an inverse index—**latent semantic indexing** (LSI) [Deerwester *et al.* 1990], **random projection** [Kaski 1998] and **independent component analysis** (ICA) [Isbell and Viola 1998]. In the following I briefly describe the vector space model as well as corresponding document encoding techniques, which are based on filtering and stemming the words appearing in the documents to encode and on selecting appropriate index terms.

The Vector Space Model

In the vector space model text documents are represented as vectors in an m-dimensional space. That is, each document D_j, $1 \leq j \leq n$, is described by a numerical feature vector $\vec{x}_j = (x_{j1}, \ldots, x_{jm})$. Each element of this vector represents a word of the document collection, i.e., the size of the vector is given by the total number m of words in the complete document collection. For a given document D_j, $1 \leq j \leq n$, the so-called **term weight** x_{jl} specifies the importance of the word W_l, $1 \leq l \leq m$, in this document w.r.t. the given document collection. The general idea is to assign large weights to terms that are frequent in relevant documents but rare in the whole document collection [Salton and Buckley 1988]. To achieve this, the weight of a term W_l in document D_j is computed as the **term frequency** tf_{jl} times the **inverse document frequency** idf_l, where the former is simply the number of occurrences of term W_l in document D_j. The latter is defined as $\mathrm{idf}_l = \log(\frac{n}{n_l})$, where n is the size of the document collection and n_l, $1 \leq l \leq m$, is the number of documents that contain term W_l, Hence it describes the specificity of term W_l within the document collection.

This weighting scheme has meanwhile proven its usability in practice. However, if used directly in this simple form, the vectors $\vec{x}_{jl}$ may have different lengths for different documents. In particular, the longer a document—that is, the more words it contains—the more often a given term can appear in it. As a consequence the frequency and thus the weight of this term are higher, which inevitably increases the length of the representing vector. This can lead to the undesired effect that in retrieval systems longer documents have better chances of being retrieved than shorter ones—independent of their actual relevance for a given task. To counteract this undesired behavior, [Salton *et al.* 1994] add a length normalization factor, which ensures that all documents have equal chances of being retrieved independent of their lengths. That is, the term weights are computed as

$$x_{jl} = \frac{\mathrm{tf}_{jl} \log \frac{n}{n_l}}{\sqrt{\sum_{k=1}^{m} \left(\mathrm{tf}_{jk} \log \frac{n}{n_k}\right)^2}}.$$

Another way of justifying the introduction of this normalization factor is through the most common measure for the similarity of two documents, which is the cosine of the angle between the two vectors representing the documents. Obviously, this cosine is 1 if the two documents are equivalent w.r.t. the set of considered terms (same or proportional term frequency for all words) and thus the two vectors representing them point into the same direction. On the other hand, it is 0 if the two documents do not have any word in common and thus the two vectors are perpendicular to each other. This behavior captures our intuition of document similarity fairly well.

The cosine of the angle between two vectors is most easily computed through the scalar product of these vectors, that is, as

$$\cos\varphi(\vec{x}_j, \vec{x}_k) = \frac{\vec{x}_j^{\top} \vec{x}_k}{|\vec{x}_j| \cdot |\vec{x}_k|}.$$

Hence it is advisable to normalize the vectors to length 1, because then the division by the lengths of the two vectors can be eliminated. Normalizing the vectors can also be seen as a preprocessing step to accelerate computations, without affecting the result of retrieval or clustering.

Note that for normalized vectors the scalar product is basically equivalent to the Euclidean distance, since it is [Borgelt and Nürnberger 2004a]

$$\frac{\vec{x}_j^{\top} \vec{x}_k}{|\vec{x}_j| \cdot |\vec{x}_k|} = 1 - \frac{1}{2} d^2\!\left(\frac{\vec{x}_j}{|\vec{x}_j|}, \frac{\vec{x}_k}{|\vec{x}_k|}\right).$$

Strangely enough, this simple geometrical relation seems to have been overlooked by many researchers, as even special versions of, for example, the fuzzy c-means algorithm have been developed that are based a scalar product derived distance [Klawonn and Keller 1999, Mendes and Sacks 2001, Frigui and Nasraoui 2003]. However, due to the above relation between the scalar product and the Euclidean distance, no such special consideration is actually necessary. Effectively, the mentioned version of the fuzzy c-means algorithm only introduces a factor of $\sqrt{2}$ into the distance measure, since the papers cited above define (for vectors normalized to length 1)

$$d^2_{\text{doc}}(\vec{x}_j, \vec{x}_k) = 1 - \vec{x}_j^{\top} \vec{x}_k.$$

It has to be conceded, though, that in addition to a scalar product based distance measure the corresponding, so-called **hyperspherical fuzzy clustering algorithm** introduces the constraint that the center vectors of the clusters must have unit length, so that they can be treated in the same way as the vectors representing documents. However, it should be immediately clear that basically the same result may also be obtained by using the normal update rule for the cluster centers and renormalizing them to length 1 afterwards. Indeed, a formal derivation (carried out by incorporating the constraint into the objective function with the help of a Lagrange multiplier [Mendes and Sacks 2001, Mendes and Sacks 2003], also cf. Section 5.2.3 for similar cases) leads to exactly this update rule: compute the weighted sum of the data points and normalize the result to length 1. As a consequence, we may conclude that the improvements of the clustering results reported by [Mendes and Sacks 2001] are due to the fact that the cluster centers are forced to lie on the m-dimensional (hyper-)sphere with radius 1, and not due to real changes of the objective function or the update rules.

Another point in favor of the scalar product based distance, which can be important in practice, is that it can be much faster to compute. The main reason is that vectors representing documents are usually sparse (that is, differ from zero in only few elements), because only a (small) subset of all words will appear in a given document. If this is exploited by a sparse representation of the vectors, only a small subset of all coordinates may have to be accessed in order to compute a needed distance. On the other hand, it may be a disadvantage of the scalar product based distance that it enforces spherical clusters. For ellipsoidal clusters we need the Mahalanobis distance, which is a generalization of the Euclidean distance (cf. Section 2.1).

For a further, more detailed discussion of the vector space model and term weighting schemes, in particular for information retrieval purposes, see, for instance, [Salton *et al.* 1975, Salton and Buckley 1988, Greiff 1998].

Filtering and Stemming

It is usually advisable to try to reduce the number of used terms, that is, the size of the dictionary of words describing the documents, and thus to minimize the dimensionality of the vector space description of the document collection. Such a reduction is possible, because usually only a (small) subset of all terms appearing in the documents of the given collection are relevant for characterizing or for distinguishing them. The main techniques for this task are filtering so-called *stop words* and stemming.

The idea of **stop word filtering** is to remove words that, in general, bear little or no content information, like articles, conjunctions, prepositions etc. Eliminating such words is harmless, because they appear frequently in basically every text document and thus give no hints how we may structure them. More generally, any word that occurs extremely often in the whole text collection can be assumed to provide only little information that can be used to distinguish between the documents [Frakes and Baeza-Yates 1992]. For example, if all texts in the given collection are about a specific subject area, it is to be expected that basically all texts contain certain words that describe key concepts of this subject area. These words, however, are of little use when it comes to structuring the documents, since they only tell us that all texts under consideration are from this subject area—a redundant information. Similarly, words that occur very rarely are likely to be of no (statistical) relevance and thus may be removed from the dictionary [Frakes and Baeza-Yates 1992]. If only very few documents (say, less than five) contain a given word, this word is useless for finding broader structures in the data: the subset of documents containing it is simply too small.

Stemming methods try to build the basic forms of words, i.e., strip the plural "s" from nouns, the "ing" from verbs, or other affixes. Linguistically, a stem is a natural group of words with the same (or very similar) meaning. After the stemming process, every word is represented by its stem in the vector space description. For example, the stem "notifi" stands for words like "notify", "notifies", "notified", "notifying", "notification" etc. As a consequence, a feature x_{jl} of a document vector $\vec{x}_j$ now describes a group of words rather than a single word and thus the vector length can be reduced considerably. A more sophisticated analysis would even consider irregularities, especially in verb forms, and map, for example, "went" to the stem "go". A well-known and widely applied stemming algorithm has been proposed originally by [Porter 1980]. It consists of a set of production rules to iteratively transform (English) words into their stems. Although it is far from perfect, it has the advantage of being simple and thus efficient.

Index Term/Keyword Selection

To further reduce the number of terms used in the vector description of a document collection, indexing or keyword selection algorithms may be applied (see, for example, [Deerwester *et al.* 1990, Witten *et al.* 1999]). In this case only the selected keywords are used to describe the documents, which not only speeds up the processing, but can also enhance the quality of the description, provided that appropriate keywords are chosen.

A common approach is to select the keywords based on (statistical) measures that try to capture how well a word can distinguish between documents. An example of such a measure the so-called **entropy** of a word W_l, $1 \leq l \leq m$, which is defined as [Lochbaum and Streeter 1989]

$$H_l = 1 + \frac{1}{\log_2 n} \sum_{j=1}^{n} p_{jl} \log_2 p_{jl} \qquad \text{with} \qquad p_{jl} = \frac{\mathrm{tf}_{jl}}{\sum_{k=1}^{n} \mathrm{tf}_{kl}},$$

where tf_{jl} is the frequency of the word W_l in document D_j, $1 \leq j \leq n$ (see above). Unfortunately, the name "entropy" for this measure, although it is common in information retrieval, is a bit misleading, since it differs from the so-called **Shannon entropy** [Shannon 1948] of the probability distribution of the word W_l over the document collection. Shannon entropy is actually only the negated sum appearing in the above formula[6]:

$$H_l^{(\mathrm{Shannon})} = -\sum_{j=1}^{n} p_{jl} \log_2 p_{jl}.$$

However, by dividing the Shannon entropy by $\log_2 n$, it is set in relation to the maximally possible Shannon entropy, which results for a uniform distribution (all probabilities $p_{jl} = \frac{1}{n}$), and thus normalized to the range $[0, 1]$. In addition, it is subtracted from 1, so that high rather than low entropy values H_l indicate good keywords. (Note that low entropy means high Shannon entropy, which in turn means that the word occurs with roughly the same frequency in all documents and is thus of little value for discriminating between documents. High entropy, on the other hand, means low Shannon entropy, which in turn means that the word is frequent in some, but infrequent in other documents and hence distinguishes well between them.) Intuitively, the entropy states how well a word is suited to separate documents from the collection by a keyword search. It may also be seen as a measure of the importance of a word in the given domain context.

[6] A detailed explanation of the ideas underlying Shannon entropy and its connection to coding theory can be found in [Borgelt and Kruse 2002].

Given such a quality measure for words, one may select keywords by simply ranking all words that could possibly be used (after stop word filtering and stemming) and selecting the m top ranked ones, where m is to be chosen by a user. However, this usually favors rare words, since they tend to have a higher entropy, independent of their actual relevance. Hence a better approach consists in selecting words that have a high entropy relative to their inverse document frequency, thus penalizing infrequent words. Empirically this approach has been found to yield a set of relevant words that are well suited to serve as index terms [Klose *et al.* 2000].

However, even such a strategy does not necessarily ensure that the set of selected keywords covers the document collection well. Therefore one should also take care to select index terms in such a way that for each document there are at least some terms actually occurring in it. This is particularly important if the number of keywords that shall be selected is limited. Otherwise too many documents will be represented by null vectors, because none of the chosen index terms appears in them.

In order to obtain a fixed number of index terms that appropriately cover the documents, we applied a greedy strategy when preparing the documents for the experiments described below [Borgelt and Nürnberger 2004a, Borgelt and Nürnberger 2004b, Borgelt and Nürnberger 2004c]: from the first document (w.r.t. an arbitrary, but fixed order) in the collection select the term with the highest relative entropy as an index term. Mark this document and all other documents containing this term. From the first of the remaining unmarked documents select again the term with the highest relative entropy as an index term. Again mark this document and all other documents containing this term. Repeat this process until all documents are marked, then unmark all of them and start over. The process is terminated when the desired number of index terms have been selected.

8.3.2 Clustering Experiments

The problem of finding descriptive weights for terms in document collections in order to improve retrieval performance has been studied extensively in the past (see, for instance, [Salton *et al.* 1975, Salton and Buckley 1988, Greiff 1998]). To achieve an improved classification or clustering performance for a given text collection, it is usually necessary to select a subset of all describing features (that is, keywords) and/or to re-weight the features w.r.t. a specific classification or clustering goal. Consequently, several studies were conducted in this direction. For example, it was explored how to select keywords based on statistical and information theoretical measures

Label	Dataset Category	Associated Theme
A	Commercial Banks	Banking & Finance
B	Building Societies	Banking & Finance
C	Insurance Agencies	Banking & Finance
D	Java	Programming Languages
E	C / C++	Programming Languages
F	Visual Basic	Programming Languages
G	Astronomy	Science
H	Biology	Science
I	Soccer	Sport
J	Motor Racing	Sport
K	Sport	Sport

Table 8.5: Categories and Themes of a benchmark data set of web pages.

[Foreman 2003, Mladenic 2001, Yang and Pedersen 1997] or how to combine clustering and keyword weighting techniques [Frigui and Nasraoui 2003] in order to improve the clustering performance. With the clustering experiments that are reported below we[7] investigated the effects of global and cluster-specific term weighting in clustering [Borgelt and Nürnberger 2004a, Borgelt and Nürnberger 2004b, Borgelt and Nürnberger 2004c]. Cluster-specific term weights were introduced through (adaptable or precomputed) cluster-specific variances to describe the shape of the cluster, while global term weights were introduced through a re-scaling of the vector space.

Web Page Collection

For our experimental studies we chose the collection of web page documents that has also been used in [Sinka and Corne 2002].[8] This data set consists of 11,000 web pages that are classified into 11 categories each of which contains about 1,000 web documents. In addition, the categories are also grouped into four disjoint themes, namely *Banking & Finance*, *Programming Languages*, *Science*, and *Sport*, which are assigned as shown in Table 8.5. [Sinka and Corne 2002] reported baseline classification rates using c-means clustering and different preprocessing strategies (alternatively stop word filtering and/or stemming, different numbers of index terms).

[7] The pronoun "we" used in the following refers to Borgelt and Nürnberger.

[8] This web page collection is available for download at the URL http://www.pedal.rdg.ac.uk/banksearchdataset/.

In order to obtain a well-covering, but nevertheless compact vector space description of this document collection, we first applied the preprocessing methods described in the preceding section. After stemming and stop word filtering we had 163,860 different words. This set was further reduced by removing terms that are shorter than 4 characters and that occur less then 15 times or more than $11,000/12 \approx 917$ times in the whole collection. With the latter condition we tried to ensure that in the describing vectors no words are referred to that perfectly separate one class from another. Such words would make clustering, in a way, too "easy", with the consequence that the results could convey a wrong impression about the quality of the approaches. From the remaining 10,626 words we chose 400 words by applying the greedy index term selection approach described in the preceding section. For our clustering experiments we selected finally subsets of the 50, 100, 150, ..., 350, 400 most frequent words in the data subset to be clustered. Based on these words we determined vector space descriptions for each document (see the preceding section for explanations) that we used in our clustering experiments. All document vectors were normalized to unit length (*after* selecting the subset of keywords).

For our experiments we used the same data sets and clustering tasks as in [Sinka and Corne 2002]. That is, we clustered the union of the semantically fairly dissimilar categories A and I, and the semantically more similar categories B and C. The average reclassification accuracies reported in [Sinka and Corne 2002] using stemming, stop word filtering, and hard c-means clustering on the resulting vector representation are 67.5% for data sets A and I and 75.4% for data sets B and C.[9] In a third set of experiments we used all classes and tried to find clusters describing the four main themes, i.e. *banking*, *programming languages*, *science*, and *sport*.

The algorithms we applied are (hard) c-means clustering, fuzzy c-means clustering and (fuzzy) learning vector quantization.[10] The (fuzzy) learning vector quantization algorithm updated the cluster parameters once for every 100 documents with the approach described in Section 5.3.2. To evaluate the clustering results we always applied the cluster model as a *nearest prototype* or *maximum membership classifier* (cf. Section 2.5) and computed the reclassification accuracy (cf. Section 7.2 for details).

[9] These results are surprising, since A and I are much easier to separate than B and C. Maybe the sets of keywords chosen by [Sinka and Corne 2002] were not well suited.

[10] All experiments were carried out with a program written in C and compiled with gcc 3.3.3 on a Pentium 4C 2.6GHz system with 1GB of main memory running S.u.S.E. Linux 9.1. The program and its sources can be downloaded free of charge at the already mentioned URL http://fuzzy.cs.uni-magdeburg.de/~borgelt/cluster.html.

Clustering using Adaptable Variances

In a first set of experiments we ran the clustering algorithms on a standard vector space description of the documents as it resulted from the preprocessing described above (experiments with a rescaled vector space are discussed below). We executed the algorithms with and without cluster centers normalized to unit length, with and without variances (i.e., spherical clusters and axes-parallel ellipsoids—diagonal covariance matrices—of equal size), and with the inverse squared distance (special Cauchy function) or the Gaussian function for the activation/membership degree. For the experiments with adaptable variances we restricted the maximum ratio of the variances (cf. Section 6.1.2) to $1.2^2 : 1 = 1.44 : 1$ if cluster centers were normalized to length 1, and to a maximum ratio of $8^2 : 1 = 64 : 1$ for unnormalized cluster centers. In preliminary runs these values had produced the best average results over all three clustering tasks described above.[11]

The results for some parameterizations of the algorithms are shown in Figures 8.6 to 8.8. All results are the mean values of ten runs, which differed in the initial cluster positions and the order in which documents were processed. The dotted horizontal lines show the default accuracy (that is, the accuracy that is obtained if all documents are assigned to the majority class). The grey horizontal lines in each block, which are also marked by diamonds to make them more easily visible, show the average reclassification accuracy in percent (left axis, computed from a confusion matrix by permuting the columns so that the minimum number of errors results, cf. Section 7.2), while the black crosses indicate the performance of individual experiments. The grey dots and lines close to the bottom show the average execution times in seconds (right axis), while the smaller grey dots and lines at the top of each diagram show the performance of a naïve Bayes classifier trained on a labeled version of the data set. The latter can be seen as an upper limit, while the default accuracy is a lower baseline.

The diagrams in the top row of each figure show the results for spherical clusters (fixed uniform variances) with cluster centers normalized to unit length (cf. Section 8.3.1 for a discussion of this approach). In this case we used a fixed cluster radius of $\frac{1}{3}$ and a standard Gaussian membership/activation function.[12] Results obtained with the membership/activa-

[11] For individual experiments, however, other values seem to be better. For example, for unnormalized centers and categories A versus I a larger maximum ratio than $8^2 : 1$ is preferable, while for categories B versus C a lower maximum ratio yields better results.

[12] Note that with an inverse squared distance membership/activation, the radius has no effect, because of the normalization to sum 1.

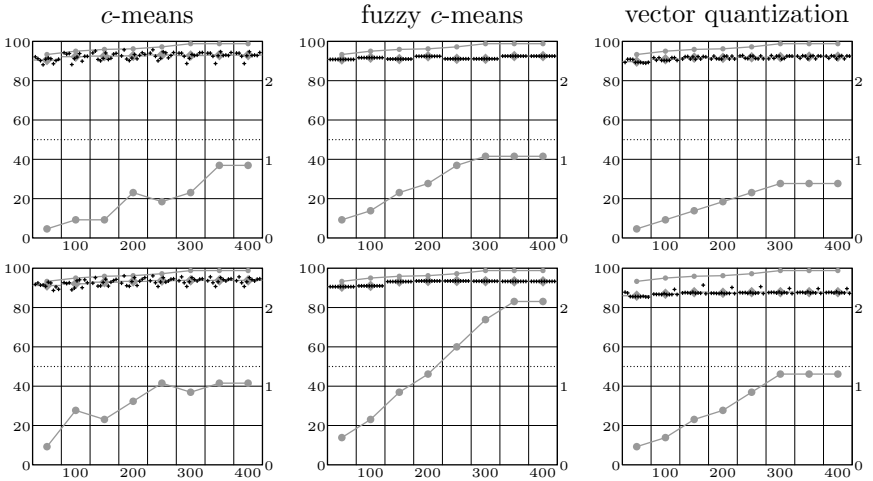

Figure 8.6: Reclassification accuracy over number of keywords on commercial banks (category A) versus soccer (category I). Top row: normalized centers; bottom row: ditto, but with adaptable variances.

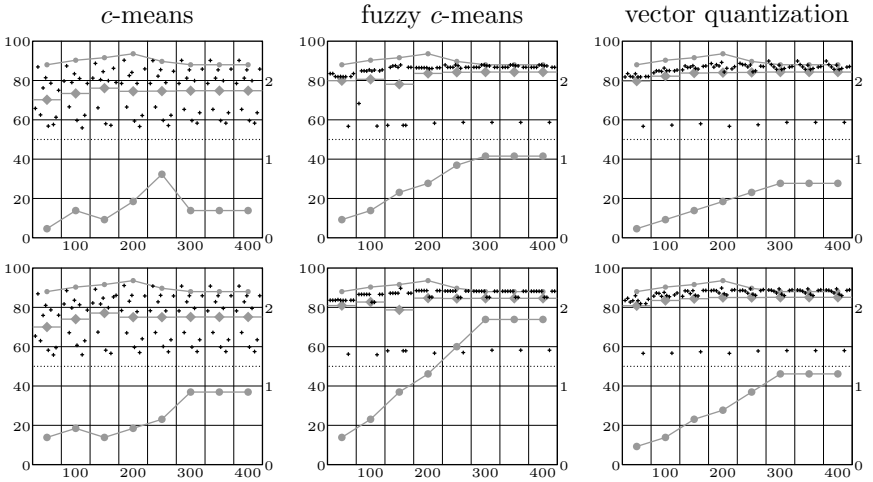

Figure 8.7: Reclassification accuracy over number of keywords on building companies (category B) versus insurance agencies (category C). Top row: normalized centers; bottom row: ditto, but with adaptable variances.

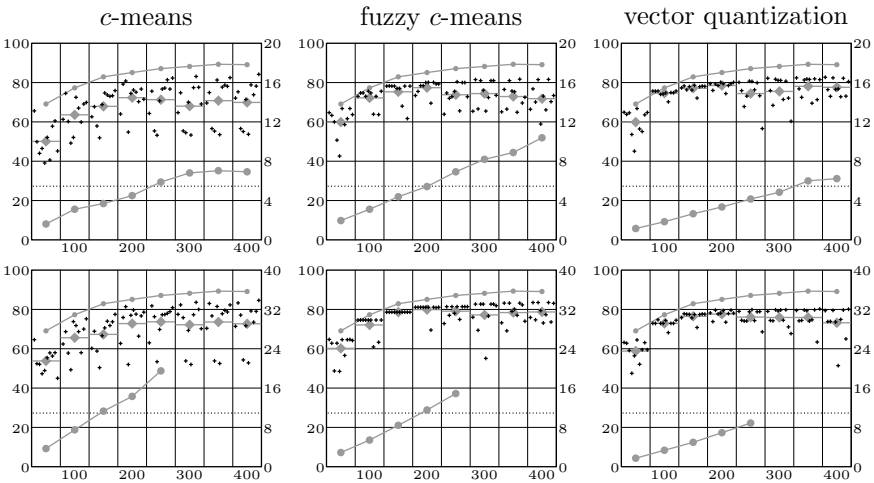

Figure 8.8: Reclassification accuracy over number of keywords on major themes (four clusters). Top row: normalized centers; bottom row: ditto, but with adaptable variances. (Note the time scale in the bottom row!)

tion computed as the inverse squared distance lead to considerably worse results for fuzzy clustering and (fuzzy) learning vector quantization. Therefore I omit an explicit discussion of these results here.

The diagrams show that the introduction of variances increases the performance of fuzzy c-means clustering in all cases. However, the performance for (hard) c-means clustering is only improved for the two class problem with data sets A and I and the four class problem, while the performance of (fuzzified) learning vector quantization is minimally improved for the semantically more similar data sets B and C and the four class problem. For the data sets A and I, however, performance deteriorated significantly.

It is remarkable that in several experiments we carried out (some of which are not reported explicitely here) we found that using ellipsoidal clusters often slightly *deteriorates* performance if the cluster centers are normalized (see Figures 8.6 to 8.8), but slightly *improves* performance for free cluster centers (i.e., no normalization to unit length, results not shown in diagrams). Furthermore, shape regularization (as it was described in Section 6.1.2) was mandatory in all experiments with adaptable variances. Without such regularization, the clusters tended to degenerate into very long and thin ellipsoids, leading to useless cluster models.

The diagrams also show that for all data sets the clustering process of fuzzy c-means clustering and (fuzzy) learning vector quantization is much more stable than that of (hard) c-means clustering. (Although for the semantically similar data sets B and C all methods seem to switch between two strong local minima, the variation around the corresponding reclassification accuracies is considerably smaller for the fuzzy approaches.) This observation confirms general experience with fuzzy clustering approaches compared to their hard or crisp counterparts, namely that they are more robust and less likely to get stuck in local optima.

It may be a little surprising that learning vector quantization not only outperforms fuzzy clustering w.r.t. the execution time, but can even compete with (hard) c-means clustering in this respect. Note, however, that the execution time curves are less smooth for (hard) c-means clustering due to the crisp behavior of the algorithm, which leads to a considerable variation in the number of update iterations needed until convergence. Therefore individual runs of hard c-means may be considerably faster while others may be considerably slower. Note also that the execution times level off for 350 and 400 words for all algorithms in Figures 8.6 and 8.7. This effect occurs, because only 316 and 300 words appear in these document subsets, respectively, while the remaining words occur only in other categories than A and I or B and C, respectively.

Keyword Weighting

The experiments reported above indicate that using cluster prototypes with adaptable variances (which may be seen as cluster-specific keyword weights) can sometimes improve the clustering performance, though only slightly. In the following I consider a related approach, in which term weights are not found by the clustering algorithm through the adaptation of variances, but are introduced into the clustering process by external considerations. Such externally determined weights may be better able to provide the necessary information for a good division into clusters, because they can exploit, for example, a user's preferences about how to structure the documents. Term weights determined in this way are either used as (fixed) cluster-specific variances or as a means to rescale the document space.

For our experiments we determined term weights from the labeled documents (that is, the data points with their associated category), because our goal is a cluster model that captures the categories well. The term weights were computed as *information gain* or the χ^2 *measure* between presence or absence of a term in a document and the associated classes.

Information gain (also known as **mutual (Shannon) information** or **(Shannon) cross entropy**), which is very popular in decision tree induction [Quinlan 1986, Quinlan 1993], measures the average or expected entropy reduction of the probability distribution over the classes that results from observing the value of a (nominal) attribute.[13] In text categorization one considers one binary attribute for each term, which indicates whether the term is present in a document or not. Intuitively, information gain then measures how well a term can be used to classify a document. Formally, the information gained from a term W_l, $1 \leq l \leq m$, about a given set $\{1, \ldots, s\}$ of s classes is defined as

$$\begin{aligned} I_{\text{gain}}(W_l) \quad = \quad & -\sum_{k=1}^{s} p_Z(k) \log_2 p_Z(k) \\ & +p_{X_l}(1) \sum_{k=1}^{s} p_{Z|X_l}(k|1) \log_2 p_{Z|X_l}(k|1) \\ & +p_{X_l}(0) \sum_{k=1}^{s} p_{Z|X_l}(k|0) \log_2 p_{Z|X_l}(k|0), \end{aligned}$$

where Z is a random variables that has the possible classes as its values, while X_l is a random variable that indicates whether the term W_l is contained in a given document ($X_l = 1$) or not ($X_l = 0$). The information is the larger, the better a term is suited to distinguish between the classes.

The χ^2 **measure**, on the other hand, which is well known in statistics, but has also been used for decision tree induction [Breiman *et al.* 1984], uses the difference of expected and observed occurrences of attributes (here: terms) to measure the statistical significance of a (nominal) attribute (here: a term) with respect to another (nominal) attribute (here: the class). The χ^2 measure can be computed based on the contingency table of a term W_l, $1 \leq l \leq m$, and the classes k, $1 \leq k \leq s$. Formally, it can be defined as

$$\begin{aligned} \chi^2(W_l) \quad = \quad & n \sum_{k=1}^{s} \left(\frac{(p_Z(k) p_{X_l}(0) - p_{X_l,Z}(0,k))^2}{p_Z(k) p_{X_l}(0)} \right. \\ & \left. + \frac{(p_Z(k) p_{X_l}(1) - p_{X_l,Z}(1,k))^2}{p_Z(k) p_{X_l}(1)} \right), \end{aligned}$$

where n is the total number of documents in the collection.

[13] A detailed discussion of the information gain measure and its interpretation can be found, for example, in [Borgelt and Kruse 2002]. The same holds for the χ^2 measure.

Intuitively, $p_{X_l,Z}(x,k)$, $1 \leq k \leq s$, is the probability that a document in class k contains the term W_l ($x = 1$) or not ($x = 0$). $p_Z(k)p_{X_l}(x)$ is the same probability, but under the assumption that the events that a document belongs to class k and that the word W_l occurs in it are independent. Hence the χ^2 measure assesses how much the true joint distribution differs from an independent one by summing the (weighted) pointwise squared differences of the two distributions. As a consequence, the χ^2 measure is the larger, the more the two distributions differ. Thus, like information gain, it is the larger, the better a term is suited to distinguish between the classes.

Rescaling the Document Space

In order to study the effects of keyword weighting, we used information gain as well as the χ^2 measure to compute the "importance" of each of the selected 400 keywords for the classification of a document. These importance values were then used to re-weight the terms in each document according to the simple linear transformation

$$x'_{jl} = x_{jl} \cdot (Q(W_l) + o),$$

where the measure Q is either I_{gain} or χ^2. The offset o is computed as

$$o = \frac{\max_{l=1}^{m} Q(W_l) - r \cdot \min_{l=1}^{m} Q(W_l)}{r - 1},$$

where r is a user-specified maximum ratio of the scaling factors for different terms and m is the total number of terms considered. From several preliminary experiments we conducted it seems that values of r must be small (close to 1) in order not to spoil the performance completely. For the experiments reported below we chose $r = 1.5$. After the re-weighting the document vectors are re-normalized to unit length, resulting in a re-scaling of the document space with respect to the importance of a keyword.

Since the results using the χ^2 measure are almost identical to those obtained with information gain, I confine myself to the experiments with information gain. The results of these experiments are shown in the top rows of Figures 8.9 to 8.11. As can be seen, no gains result in any of the cases (compared to Figures 8.6 to 8.8). The accuracy rather deteriorates slightly, an effect that gets stronger with higher values of r as we observed in other experiments. Hence we can conclude that re-scaling the document space in the way described does not lead to an improved performance.

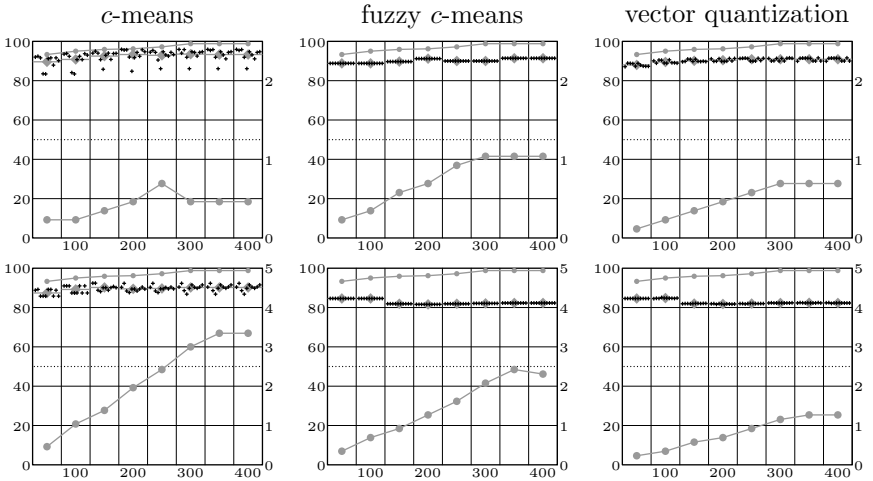

Figure 8.9: Reclassification accuracy over number of keywords on commercial banks (category A) versus soccer (category I). Top row: document space rescaled, spherical clusters; bottom row: fixed cluster-specific variances.

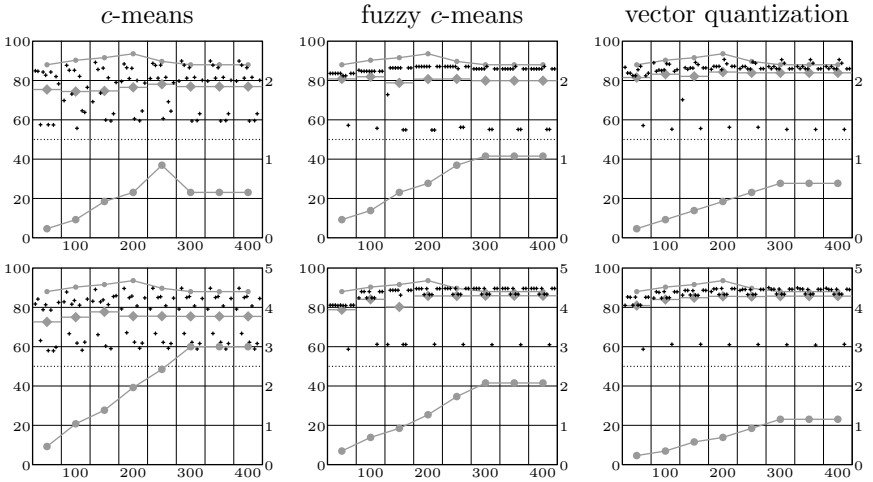

Figure 8.10: Reclassification accuracy over number of keywords on building companies (B) versus insurance agencies (C). Top row: document space rescaled, spherical clusters; bottom row: fixed cluster-specific variances.

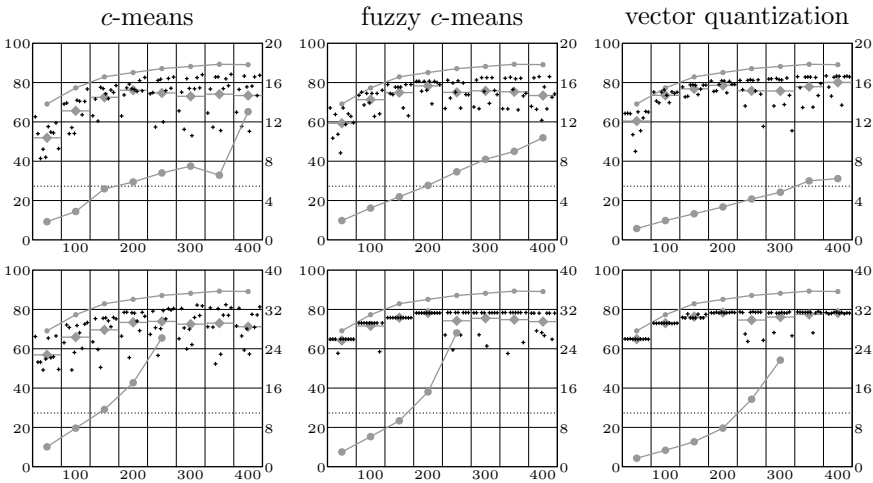

Figure 8.11: Reclassification accuracy over number of keywords on major themes (four clusters). Top row: document space rescaled, spherical clusters; bottom row: fixed cluster-specific variances.

Cluster-Specific Keyword Weights

Instead of using information gain or the χ^2 measure to rescale the document space, one may also add shape parameters (i.e., variances) to the cluster prototypes. These variances are then initialized according to the "importance" of a term. This has the advantage that term weights can be cluster-specific, since each cluster may use a different set of variances.

To evaluate this approach, we proceeded as follows: in a first step we clustered the documents with randomly chosen starting points for the cluster centers (normalized to length 1) and without variances. By evaluating the confusion matrix of a classification with a nearest prototype classifier based on the cluster model, we obtained the "best" assignment of classes to these clusters. Then the cluster prototypes were enhanced with cluster-specific variances, which were computed as the product of the term frequency in the class and the information gain or χ^2 value of the term w.r.t. a separation of the class assigned to the cluster from all other classes. The maximum ratio of the variances was restricted to $1.2^2 : 1 = 1.44 : 1$ (compare the experiments with adaptable variances reported above). Other values for this maximum ratio (higher as well as lower) led to worse results. Especially

larger values considerably worsened the performance. Finally, in a second clustering run, these enhanced cluster prototypes were optimized without changing the variances (only the cluster centers were adapted).

The results of these experiments are shown in the bottom rows of Figures 8.9 to 8.11. As can be seen when these figures are compared to Figures 8.6 to 8.8, the cluster-specific variances stabilize the results for the four cluster problem and—though only very slightly—improve the performance for the two cluster problems. Thus we can conclude that cluster-specific variance may provide some means for term weighting. However, the approach seems to be very sensitive to the parameter settings. Furthermore, the computational costs are fairly high.

Choosing Initial Cluster Centers

As mentioned in the discussion of the clustering experiments with adaptable variances, all clustering methods seem to switch between local minima depending on the initial cluster centers chosen. This is in fact a well known clustering problem, especially for the less robust (hard) c-means algorithm, which is prone to get stuck in local optima easily (cf. Section 5.2.2). In order to find means to cope with this problem we studied a simple initialization approach: for each class we sorted the index terms w.r.t. the product of the term frequency in the class and the information gain of the term w.r.t. a separation of the class from all other classes (see the preceding paragraph). Then we selected the first k words in these lists and initialized the cluster center using the same value for each selected word and zero for all others, finally normalizing the vector to unit length. Even for fairly small values of k (i.e. few selected words), this initialization results in a very stable clustering performance. Hence we can conclude that known describing keywords can be used to initialize the clustering process. In this way undesired local minima may be avoided and the results may be stabilized.

Keyword Extraction for Cluster Labeling

Once a set of documents are clustered, one would like to obtain an expressive description of the found clusters. A simple approach in this direction is to select keywords that characterize a cluster well. In order to obtain such characteristic terms for each cluster, we compared different keyword rankings for several clustering experiments on the four major themes using 200 terms. I selected a typical result for this brief discussion. In Table 8.6 the terms are sorted descendingly by their information gain $I_{\text{gain}}(W_l)$ w.r.t.

the found clusters (second column). Columns 3 to 6, each of which corresponds to a cluster, show the probability that term W_l, $1 \leq l \leq m$, appears in the corresponding cluster, with the highest probability per row set in bold face. Going down a column and collecting the highlighted terms yields a ranked keyword list for the corresponding cluster. Using $\chi^2(W_l)$ for the ranking leads to very similar results and thus I omit a separate table.

As an alternative, Table 8.7 shows the keyword ranking obtained by selecting the terms with the highest weights (largest center coordinates) for each cluster prototype. As can be seen, the order of the keywords is very similar to that of Table 8.6 (the rank differences are fairly small, at least in the shown top part of the list). Thus, to describe a cluster the terms can simply be selected from an ordered list of the center coordinates of each cluster, which is, of course, much more efficient than the computation of an information measure. At least for this example, the descriptive quality seems to be very similar.

8.3.3 Conclusions

The experiments with adaptable variances [Borgelt and Nürnberger 2004a, Borgelt and Nürnberger 2004b] show that fuzzy clustering and (fuzzy) learning vector quantization with shape parameters, as it was developed in Section 5.3.3, can be useful for clustering collections of documents. The latter leads to shorter execution times and may be even slightly more robust than fuzzy clustering, which, in turn, is known to be considerably more robust than crisp clustering. In addition, (fuzzy) learning vector quantization enables "online" clustering, since only a fraction of the documents is needed for each update. Shape regularization, as it was developed in Section 6.1.2, turned out to be mandatory in this application domain.

The experiments concerning term weighting [Borgelt and Nürnberger 2004b, Borgelt and Nürnberger 2004c] show that including prior information about the "importance" or "goodness" of a keyword for a desired class or cluster can, in principle, improve the clustering performance. However, it is fairly difficult to find a good way of scaling the documents or enhancing the cluster prototypes in an appropriate way. Scaling the document space does not yield any improvement at all. On the other hand, cluster-specific variances derived from the "importance" of index terms can slightly improve and stabilize the clustering performance. However, the gains are marginal and the approaches seem to be fairly sensitive to parameter settings. The most substantial gains, at least w.r.t. robustness, result from an initialization of the cluster centers w.r.t. the importance of keywords.

term W	$I_{\text{gain}}(W)$	$p(W\|1)$	$p(W\|2)$	$p(W\|3)$	$p(W\|4)$
footbal	0.143	0.001	**0.299**	0.007	0.004
championship	0.120	0.001	**0.242**	0.001	0.002
galaxi	0.118	0.003	0.002	0.001	**0.270**
repay	0.112	0.004	0.000	**0.212**	0.000
genet	0.106	0.003	0.005	0.002	**0.253**
season	0.100	0.011	**0.277**	0.018	0.027
theori	0.094	0.017	0.012	0.004	**0.266**
applet	0.089	**0.178**	0.001	0.001	0.004
pension	0.087	0.004	0.004	**0.177**	0.001
astronom	0.084	0.002	0.003	0.000	**0.196**
energi	0.080	0.010	0.022	0.009	**0.249**
premium	0.080	0.006	0.006	**0.183**	0.008
button	0.080	**0.234**	0.018	0.030	0.011
sequenc	0.079	0.049	0.018	0.003	**0.254**
borrow	0.078	0.006	0.005	**0.180**	0.013
detect	0.076	0.073	0.005	0.005	**0.235**
telescop	0.076	0.003	0.003	0.003	**0.193**
cosmolog	0.075	0.003	0.002	0.000	**0.178**
coach	0.074	0.001	**0.173**	0.008	0.003
distanc	0.074	0.012	0.058	0.005	**0.239**
chequ	0.063	0.005	0.005	**0.140**	0.001
arrai	0.057	**0.148**	0.009	0.001	0.077
script	0.055	**0.150**	0.005	0.010	0.013
javascript	0.052	**0.147**	0.002	0.016	0.017
browser	0.051	**0.180**	0.011	0.025	0.050
gross	0.050	0.004	0.012	**0.134**	0.015
activex	0.050	**0.109**	0.001	0.002	0.002
liabil	0.050	0.005	0.006	**0.123**	0.006
surfac	0.050	0.020	0.028	0.004	**0.178**
default	0.049	**0.167**	0.009	0.030	0.020
input	0.049	**0.157**	0.016	0.008	0.053
purchas	0.048	0.035	0.035	**0.186**	0.019
⋮	⋮			⋮	

Table 8.6: Table of keywords ordered by information gain (second column). The values in columns 3 to 6 are the probabilities that the term W appears in the given cluster c, $c \in \{1, 2, 3, 4\}$.

term W	μ_{1W}
applet	0.592
button	0.331
javascript	0.319
script	0.263
browser	0.188
password	0.153
thread	0.150
arrai	0.148
⋮	⋮

term W	μ_{2W}
footbal	0.723
coach	0.329
championship	0.298
season	0.242
david	0.132
basketbal	0.128
round	0.122
scotland	0.100
⋮	⋮

term W	μ_{3W}
repay	0.495
borrow	0.324
discount	0.312
pension	0.281
gross	0.226
chequ	0.225
premium	0.222
purchas	0.194
⋮	⋮

term W	μ_{4W}
galaxi	0.626
genet	0.436
sequenc	0.342
theori	0.181
distanc	0.171
cosmolog	0.170
telescop	0.164
energi	0.156
⋮	⋮

Table 8.7: Tables of keywords for each cluster obtained by sorting the terms of each cluster prototype by their weight (center coordinate).

Chapter 9

Conclusions and Future Work

The study of prototype-based clustering and classification methods carried out in this thesis showed that these methods can be organized into a nicely uniform framework. Although this possibility was widely recognized, a rigid and comprehensive exposition, which not only lists a variety of related approaches, but structures and combines them into a coherent scheme, was still missing. I hope to have closed this gap with this thesis, even though it is, of course, impossible to exhaustively cover all prototype-based approaches to classification and clustering that have been suggested.

The developed framework of prototype-based methods consists of the following building blocks, which have been treated in individual chapters:

- **prototypes: properties and interaction** (Chapter 2)
  Prototypes are described by a (typical) reference point (that is, a cluster center) and a possibly prototype-specific similarity function that is defined as a radial function over a distance measure. Prototypes may be endowed with a size and a shape by using a Mahalanobis distance that is parameterized with a (cluster-specific) covariance matrix. Since a cluster model or a classifier consist of several prototypes, their interaction can have a considerable influence on the model and its behavior. This interaction as well as the limiting behavior far away from the cluster centers is controlled by the choice of how membership degrees to clusters are transformed and normalized. Finally, there are several methods of turning a cluster model into a classifier.

- **objective functions** (Chapter 3)

  There are two main approaches to define an objective function for classification and clustering: the objective function may either be based on (squared) deviations from some reference, or it is defined with the help of a probabilistic model of the data generation process. This gives rise to four types of objective functions, two for clustering (minimum sum of (squared) distances and maximum likelihood of the data set) and two for classifier construction (minimum sum of (squared) errors and maximum likelihood ratio). The possibility to categorize the different approaches in this way provides an appealing framework.

- **initialization methods** (Chapter 4)

  Since the optimum of the objective functions for prototype-based classification and clustering usually cannot be determined directly, one has to rely on an iterative improvement scheme. In order to obtain a starting point, the prototypes and, for a classifier, the parameters of the classification functions have to be initialized. For this initialization there is a ladder of methods, ranging from very simple procedures, which exploit only very basic information about the given data set, to highly sophisticated techniques, which can even be seen as clustering or classifier construction methods in their own right.

- **update methods** (Chapter 5)

  For updating the parameters of the prototypes iteratively, there are basically three core methods, namely gradient ascent or descent, alternating optimization, and competitive learning[1]. As emphasized in Chapter 5, these update methods are closely related and thus give rise to a transfer of extensions and improvements between them. An example is the introduction of shape and size parameters into learning vector quantization as it was developed in Section 5.3.3. In addition, the three update methods can fairly freely be combined with the four objective functions, thus adding to the appeal of this framework.

- **update modifications** (Chapter 6)

  By introducing modifications into the update procedure, it is possible to enhance both the robustness and the speed of the training process. Outliers in the data set and the increased flexibility of models in which prototypes are enhanced with size and shape information often lead to an unstable behavior of the construction procedure. While the former

[1] Genetic or evolutionary algorithms, which were considered at the end of Chapter 5 as an alternative, turn out to be clearly inferior to these three.

can effectively be handled with known approaches like noise clustering, I introduced shape and size regularizations techniques to cope with the latter. Furthermore, by transferring more sophisticated neural network learning techniques to other update schemes, I obtained effective methods for accelerating the training process.

- **evaluation methods** (Chapter 7)

  Even though the objective functions already provide direct and natural means to evaluate a trained classifier or cluster model, several other evaluation measures have been suggested over the years in order to capture other desirable properties and to validate the results. Furthermore, one has to pay attention to the data on which the trained models are evaluated, since an evaluation on the training data may convey a wrong impression due to overfitting effects. The main techniques to cope with this problem are cross validation and resampling.

That the developed improvements mentioned in the above summary are actually relevant for practical purposes was demonstrated with experimental results reported in Chapter 8. The quality of the clustering results and the speed with which they could be obtained is indeed improved considerably. In the application to document structuring it even turned out that shape and size regularization are mandatory in this domain, because otherwise the cluster shapes tend to degenerate in basically every run.

Since the research area of prototype-based methods is vast, with an abundance of slightly different approaches and extensions, it is natural that I could not treat all possible variations and enhancements that have been suggested. However, among those that I encountered during the work on this thesis, I identified mainly three interesting lines, which my future research in this area may follow:

- **merging and splitting prototypes**

  All methods discussed in this thesis keep the number of prototypes constant during the update process. However, even the old ISODATA algorithm [Ball and Hall 1966] contains rules to split, merge, add, or delete prototypes in order to achieve a better fit to the data. Newer approaches in this direction can be found in [Krishnapuram and Freg 1992, Frigui and Krishnapuram 1997, Stutz 1998]. Such strategies can be an efficient way to find an appropriate number of prototypes, even though in their current state these approaches are inferior to approaches based on cross validation or resampling.

- **coupled prototypes**

  In the models considered in this thesis the prototypes are basically unrelated and any coupling between them is brought about only by the membership normalization. However, there are several approaches that introduce a stronger coupling of the prototypes, organizing them, for example, in some grid. Examples in this direction include *self-organizing maps* [Kohonen 1995], and *topographic clustering* [Graepel and Obermayer 1998, Lesot *et al.* 2003]. With such approaches so-called *topology preserving mappings* can be learned, which are highly useful for data visualization and dimensionality reduction.

- **feature selection**

  Throughout this thesis it was implicitly assumed that the data space is fixed, that is, that the attributes used to describe the cases or objects under consideration have already been selected. In practice, however, it is often not known which features are most appropriate. Of course, selecting the right features may also be seen as a model selection problem and thus be attacked by approaches like cross validation and resampling. However, due to the usually high number of possible selections of features, they suffer from very high computational costs. An alternative are methods that try to exploit information gathered during the clustering or classification process in order to dynamically select from or weight the available features. Examples for clustering include the approaches in [Roth and Lange 2003, Law *et al.* 2004].

Summarizing, I think it is proper to say that prototype-based methods are already a very powerful tool for data analysis, but that they nevertheless still offer several challenging problems for future research.

Appendix A

Mathematical Background

In this appendix I briefly review the mathematical background of some techniques used in this thesis. In Section A.1 I discuss basic derivatives w.r.t. vectors and matrices, in Section A.2 I consider some properties of radial functions, in Section A.8 I treat the method of Lagrange multipliers, and in Sections A.3 to A.6 I recall linear algebra methods needed for the interpretation and initialization of (sets of) clusters.

A.1 Basic Vector and Matrix Derivatives

Since in Chapter 5 and in the subsequent section the derivatives of certain expressions (which are functions of vectors and matrices) w.r.t. a vector or a matrix are needed, I review in this section some basics about such derivatives. In particular, I compute frequently occurring derivatives like

$$\nabla_{\vec{v}}\, \vec{v}^\top \mathbf{A} \vec{v}, \qquad \nabla_{\mathbf{A}}\, \vec{v}^\top \mathbf{A} \vec{v}, \qquad \text{and} \qquad \nabla_{\mathbf{A}}\, |\mathbf{A}|,$$

where $\vec{v}$ is an m-dimensional vector, $\mathbf{A}$ is an $m \times m$ matrix, and

$$\nabla_{\vec{v}} = \begin{pmatrix} \frac{\partial}{\partial v_1} \\ \vdots \\ \frac{\partial}{\partial v_m} \end{pmatrix} \qquad \text{and} \qquad \nabla_{\mathbf{A}} = \begin{pmatrix} \frac{\partial}{\partial a_{11}} & \cdots & \frac{\partial}{\partial a_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial a_{m1}} & \cdots & \frac{\partial}{\partial a_{mm}} \end{pmatrix},$$

are differential operators, which are treated formally like a vector and a matrix, respectively, and are "multiplied" with the function following them.

To compute the abovementioned derivatives, we write out $\vec{v}^\top \mathbf{A}\vec{v}$ as

$$\vec{v}^\top \mathbf{A}\vec{v} = \sum_{i=1}^{m}\sum_{j=1}^{m} v_i a_{ij} v_j.$$

Then we have $\forall k; 1 \leq k \leq m$:

$$\frac{\partial}{\partial v_k}\vec{v}^\top \mathbf{A}\vec{v} = \frac{\partial}{\partial v_k}\sum_{i=1}^{m}\sum_{j=1}^{m} v_i a_{ij} v_j = \sum_{i=1}^{m}(a_{ki} + a_{ik})v_i$$

and therefore

$$\nabla_{\vec{v}}\, \vec{v}^\top \mathbf{A}\vec{v} = \left(\sum_{i=1}^{m}(a_{1i} + a_{i1})v_1, \ldots, \sum_{i=1}^{m}(a_{1m} + a_{im})v_m\right)^\top = \left(\mathbf{A} + \mathbf{A}^\top\right)\vec{v}.$$

Alternatively, we may derive this formula by applying the product rule as

$$\nabla_{\vec{v}}\, \vec{v}^\top \mathbf{A}\vec{v} = \left(\nabla_{\vec{v}}\, \vec{v}\right)^\top \mathbf{A}\vec{v} + \left(\vec{v}^\top (\nabla_{\vec{v}}\, \mathbf{A}\vec{v})\right)^\top = \mathbf{A}\vec{v} + \left(\vec{v}^\top \mathbf{A}\right)^\top = \left(\mathbf{A} + \mathbf{A}^\top\right)\vec{v}.$$

As special cases of this result we have immediately

$$\nabla_{\vec{v}}\, \vec{v}^\top \mathbf{A}\vec{v} = 2\mathbf{A}\vec{v}$$

if $\mathbf{A}$ is symmetric[1] and thus $\mathbf{A} = \mathbf{A}^\top$ and

$$\nabla_{\vec{v}}\, \vec{v}^\top \vec{v} = 2\vec{v}$$

if $\mathbf{A}$ is the unit matrix $\mathbf{1}$. These formulae are very intuitive when compared to the standard rules for derivatives w.r.t. scalar, real-valued quantities.

Similarly, for the matrix derivative we have $\forall k, l; 1 \leq k, l \leq m$:

$$\frac{\partial}{\partial a_{kl}}\vec{v}^\top \mathbf{A}\vec{v} = \frac{\partial}{\partial a_{kl}}\sum_{i=1}^{m}\sum_{j=1}^{m} v_i a_{ij} v_j = v_k v_l$$

and therefore

$$\nabla_{\mathbf{A}}\, \vec{v}^\top \mathbf{A}\vec{v} = \begin{pmatrix} v_1 v_1 & \cdots & v_1 v_m \\ \vdots & \ddots & \vdots \\ v_m v_1 & \cdots & v_m v_m \end{pmatrix} = \vec{v}\,\vec{v}^\top.$$

The expression $\vec{v}\,\vec{v}^\top$ appearing on the right is the so-called **outer product** or **matrix product** of the vector $\vec{v}$ with itself, in contrast to the so-called **inner product** or **scalar product** $\vec{v}^\top \vec{v}$.

[1] An $m \times m$ matrix $\mathbf{A}$ is called **symmetric** iff $\forall i, j; 1 \leq i, j \leq m : a_{ij} = a_{ji}$ or equivalently iff $\mathbf{A} = \mathbf{A}^\top$, that is, if $\mathbf{A}$ equals its transpose.

Finally, I consider $\nabla_{\mathbf{A}} |\mathbf{A}|$, which is slightly more complicated. It is best computed by referring to the definition of the determinant $|\mathbf{A}|$ through an **expansion by minors** [Bronstein *et al.* 1995, p. 228], i.e., by exploiting

$$\forall i; 1 \leq i \leq m: \quad |\mathbf{A}| = \sum_{i=1}^{m} a_{ij}\, \mathbf{a}^*_{ij}$$

(expansion by the i-th row) or

$$\forall j; 1 \leq j \leq m: \quad |\mathbf{A}| = \sum_{j=1}^{m} a_{ij}\, \mathbf{a}^*_{ij}$$

(expansion by the j-th column), where the $\mathbf{a}^*_{ij}$, $1 \leq i, j \leq m$, are the so-called **cofactors** of the matrix elements a_{ij}. They are defined as

$$\forall i, j; 1 \leq i, j \leq m: \quad \mathbf{a}^*_{ij} = (-1)^{i+j} \mathbf{a}_{ij},$$

with the $\mathbf{a}_{ij}$ being the so-called **minors**, i.e. the determinants $|\mathbf{A}_{ij}|$ of the matrices $\mathbf{A}_{ij}$, which result from the matrix $\mathbf{A}$ by removing the i-th row and the j-th column. Note that consequently $\mathbf{a}_{ij}$ and thus $\mathbf{a}^*_{ij}$ are independent of all a_{kl}, $1 \leq k, l \leq m$, with $k = i$ or $l = j$, a fact that is needed below. Note also that the cofactors $\mathbf{a}^*_{ij}$ satisfy [Bronstein *et al.* 1995, p. 229]

$$\forall i, j; 1 \leq i, j \leq m, i \neq j: \quad \sum_{k=1}^{m} a_{ik}\, \mathbf{a}^*_{jk} = 0,$$

that is, if in an expansion by minors w.r.t. the i-th row the cofactors are replaced by those of another row j, $j \neq i$, the result is zero.

The matrix $\mathbf{A}^*$ built from all cofactors of $\mathbf{A}$, that is,

$$\mathbf{A}^* = \begin{pmatrix} \mathbf{a}^*_{11} & \cdots & \mathbf{a}^*_{1m} \\ \vdots & \ddots & \vdots \\ \mathbf{a}^*_{m1} & \cdots & \mathbf{a}^*_{mm} \end{pmatrix},$$

is called the **cofactor matrix** and its transpose $\mathbf{A}^{*\top}$ is called the **adjoint matrix**[2] of the matrix $\mathbf{A}$ [Bronstein *et al.* 1995, p. 229]. For the following it is worth noting that the inverse $\mathbf{A}^{-1}$ relates to the adjoint matrix by

$$\mathbf{A}^{-1} = \frac{1}{|\mathbf{A}|} \mathbf{A}^{*\top} \qquad \text{or equivalently} \qquad \mathbf{A}^{*\top} = |\mathbf{A}|\, \mathbf{A}^{-1},$$

[2] Not to be confused with the adjoint matrix (also called adjugate matrix or conjugate transpose) of a complex matrix $\mathbf{C}$, which is the transpose of the complex conjugate of $\mathbf{C}$.

as can easily be verified by computing

$$\mathbf{A}\,\mathbf{A}^{-1} = \mathbf{A}\,\frac{1}{|\mathbf{A}|}\mathbf{A}^{*\top} = \frac{1}{|\mathbf{A}|}\begin{pmatrix} \sum_{k=1}^{m} a_{1k}\mathbf{a}_{1k}^{*} & \cdots & \sum_{k=1}^{m} a_{1k}\mathbf{a}_{mk}^{*} \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^{m} a_{mk}\mathbf{a}_{1k}^{*} & \cdots & \sum_{k=1}^{m} a_{mk}\mathbf{a}_{mk}^{*} \end{pmatrix}$$
$$= \frac{1}{|\mathbf{A}|}\begin{pmatrix} |\mathbf{A}| & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & |\mathbf{A}| \end{pmatrix} = \mathbf{1}.$$

We are now equipped with all necessary ingredients to compute the derivative $\nabla_{\mathbf{A}}\,|\mathbf{A}|$. Using for each derivative w.r.t. a matrix element an expansion by the row of this element, we have $\forall i, j; 1 \leq i, j \leq m$:

$$\frac{\partial}{\partial a_{ij}}|\mathbf{A}| = \frac{\partial}{\partial a_{ij}}\sum_{k=1}^{m} a_{kj}\,\mathbf{a}_{kj}^{*} = \mathbf{a}_{ij}^{*},$$

since all $\mathbf{a}_{kj}^{*}$, $1 \leq k \leq m$, are independent of a_{ij} (see above), and therefore

$$\nabla_{\mathbf{A}}\,|\mathbf{A}| = \mathbf{A}^{*\top} = |\mathbf{A}|\,\mathbf{A}^{-1}.$$

A.2 Properties of Radial Functions

This section is concerned with some properties of the generalized Cauchy and Gaussian radial functions (cf. Section 2.2). I compute the normalization factors needed to scale them to a unit integral and their derivatives as they are needed for the gradient methods (cf. Section 5.1).

A.2.1 Normalization to Unit Integral

In order to determine the normalization factor that scales a given radial function to a unit integral on the m-dimensional space $\mathbb{R}^m$ (so that it can be interpreted as a probability density function on this space), the radial function is integrated over (hyper-)sphere shells of infinitesimal thickness $\mathrm{d}r$. A core quantity in this integration is the surface area A_m of such an m-dimensional (hyper-)sphere as a function of its radius r, which is given by the Jacobi formula [Bronstein *et al.* 1996, 0.1.6, p. 15]:

$$A_m(r) = \frac{2\pi^{\frac{m}{2}}\,r^{m-1}}{\Gamma(\frac{m}{2})}, \qquad m \geq 2.$$

This formula is used as a starting point in the following two paragraphs.

Generalized Cauchy Function

In order to determine the normalization factor for the generalized Cauchy function, we have to compute the integral

$$\int_0^\infty \underbrace{\frac{1}{r^a+b}}_{\substack{\text{generalized}\\ \text{Cauchy}\\ \text{function}}} \underbrace{\frac{2\pi^{\frac{m}{2}} r^{m-1}}{\Gamma\left(\frac{m}{2}\right)}}_{\substack{\text{surface area of the } m\text{-}\\ \text{dimensional (hyper-)}\\ \text{sphere with radius } r}} \underbrace{\mathrm{d}r.}_{\substack{\text{thickness of}\\ \text{the (hyper-)}\\ \text{sphere shell}}}$$

Note that this formula works also for $m = 1$, although the Jacobi formula (see above) is valid only for $m \geq 2$. For $m = 1$ the fraction evaluates to 2, which takes care of the fact that the integral has to be doubled in order to take the other side (i.e. from $-\infty$ to 0) into account. We get

$$\int_0^\infty \frac{2\pi^{\frac{m}{2}} r^{m-1}}{\Gamma\left(\frac{m}{2}\right)} \frac{1}{r^a+b}\,\mathrm{d}r \quad = \quad \frac{2\pi^{\frac{m}{2}}}{b\Gamma\left(\frac{m}{2}\right)} \int_0^\infty \frac{r^{m-1}}{\frac{1}{b}r^a+1}\,\mathrm{d}r.$$

In order to reduce this integral to the known formula

$$\int_0^\infty \frac{x^{\alpha-1}}{1+x^\beta}\mathrm{d}x = \frac{\pi}{\beta \sin\frac{\alpha\pi}{\beta}}, \qquad \alpha,\beta \in \mathbb{R}, \quad 0 < \alpha < \beta,$$

[Bronstein *et al.* 1996, 0.9.6, No. 53, p. 189], we substitute x for $rb^{-\frac{1}{a}}$.

$$\begin{aligned} x = rb^{-\frac{1}{a}} \quad &\Rightarrow \quad r = xb^{\frac{1}{a}}, \\ &\Rightarrow \quad r^{m-1} = \left(xb^{\frac{1}{a}}\right)^{m-1} = x^{m-1}b^{\frac{m-1}{a}}, \\ &\Rightarrow \quad \frac{\mathrm{d}x}{\mathrm{d}r} = b^{-\frac{1}{a}} \quad \Rightarrow \quad \mathrm{d}r = b^{\frac{1}{a}}\mathrm{d}x. \end{aligned}$$

Note that the integration bounds—from 0 to ∞—do not change. By inserting the above equations into the integral formula we arrive at

$$\begin{aligned} \frac{2\pi^{\frac{m}{2}}}{b\Gamma\left(\frac{m}{2}\right)} \int_0^\infty \frac{r^{m-1}}{\frac{1}{b}r^a+1}\,\mathrm{d}r \quad &= \quad \frac{2\pi^{\frac{m}{2}}}{b\Gamma\left(\frac{m}{2}\right)} \int_0^\infty \frac{x^{m-1}b^{\frac{m-1}{a}}}{x^a+1} b^{\frac{1}{a}}\mathrm{d}x \\ &= \quad \frac{2\pi^{\frac{m}{2}} b^{\frac{m}{a}}}{b\Gamma\left(\frac{m}{2}\right)} \int_0^\infty \frac{x^{m-1}}{1+x^a}\,\mathrm{d}x \\ &= \quad \frac{2\pi^{\frac{m}{2}} b^{\frac{m}{a}-1}}{\Gamma\left(\frac{m}{2}\right)} \frac{\pi}{a \sin\frac{m\pi}{a}} \\ &= \quad \frac{2\pi^{\frac{m}{2}+1} b^{\frac{m}{a}-1}}{a\Gamma\left(\frac{m}{2}\right) \sin\frac{m\pi}{a}}. \end{aligned}$$

Therefore the desired normalization factor is

$$\gamma^*_{\text{Cauchy}}(a,b,m) = \frac{a\Gamma\left(\frac{m}{2}\right)\sin\frac{m\pi}{a}}{2\pi^{\frac{m}{2}+1}b^{\frac{m}{a}-1}}.$$

Since the integral formula used above is valid for $\beta > \alpha > 0$, we obtain that the Cauchy function can be scaled to a unit integral provided $a > m > 0$, the second half of which necessarily holds, since $m \geq 1$. The other parameter b must be positive, i.e. $b > 0$ (negative values are ruled out by the requirement that a probability density function must be non-negative).

Note that for the standard Cauchy function (that is, for $a = 2$ and $b = 1$), where due to the requirement $a > m$ it must be $m = 1$, we obtain

$$\gamma^*_{\text{Cauchy}}(2,1,1) = \frac{1}{\pi}.$$

Generalized Gaussian Function

In order to determine the normalization factor for the generalized Gaussian function, we have to compute the integral

$$\int_0^\infty \underbrace{e^{-\frac{1}{2}r^a}}_{\substack{\text{generalized}\\ \text{Gaussian}\\ \text{function}}} \underbrace{\frac{2\pi^{\frac{m}{2}}r^{m-1}}{\Gamma\left(\frac{m}{2}\right)}}_{\substack{\text{surface area of the } m\text{-}\\ \text{dimensional (hyper-)}\\ \text{sphere with radius } r}} \underbrace{\mathrm{d}r}_{\substack{\text{thickness of}\\ \text{the (hyper-)}\\ \text{sphere shell}}}.$$

As in the preceding section this formula also works for $m = 1$. We get

$$\int_0^\infty \frac{2\pi^{\frac{m}{2}}r^{m-1}}{\Gamma\left(\frac{m}{2}\right)} e^{-\frac{1}{2}r^a}\mathrm{d}r \quad = \quad \frac{2\pi^{\frac{m}{2}}}{\Gamma\left(\frac{m}{2}\right)}\int_0^\infty r^{m-1}e^{-\frac{1}{2}r^a}\mathrm{d}r.$$

In order to reduce this integral to the known formula

$$\int_0^\infty x^\beta e^{-\alpha x}\mathrm{d}x = \frac{\Gamma(\beta+1)}{\alpha^{\beta+1}}, \qquad \alpha,\beta \in \mathbb{R}, \quad \alpha > 0, \quad \beta > -1,$$

[Bronstein *et al.* 1996, 0.9.6, No. 1, p. 184], we substitute x for r^a.

$$\begin{aligned} x = r^a \quad &\Rightarrow \quad r = \sqrt[a]{x}, \\ &\Rightarrow \quad r^{m-1} = (\sqrt[a]{x})^{m-1} = x^{\frac{m-1}{a}}, \\ &\Rightarrow \quad \frac{\mathrm{d}x}{\mathrm{d}r} = ar^{a-1} \quad \Rightarrow \quad \mathrm{d}r = \frac{1}{a\,(\sqrt[a]{x})^{a-1}}\,\mathrm{d}x = \frac{1}{a}x^{\frac{1-a}{a}}\,\mathrm{d}x. \end{aligned}$$

Note that the integration bounds—from 0 to ∞—do not change. By inserting the above equations into the integral formula we arrive at

$$\begin{aligned}
\frac{2\pi^{\frac{m}{2}}}{\Gamma\left(\frac{m}{2}\right)} \int_0^\infty r^{m-1} e^{-\frac{1}{2}r^a} \,\mathrm{d}x &= \frac{2\pi^{\frac{m}{2}}}{\Gamma\left(\frac{m}{2}\right)} \int_0^\infty x^{\frac{m-1}{a}} e^{-\frac{1}{2}x} \frac{1}{a} x^{\frac{1-a}{a}} \mathrm{d}x \\
&= \frac{2\pi^{\frac{m}{2}}}{a\Gamma\left(\frac{m}{2}\right)} \int_0^\infty x^{\frac{m}{a}-1} e^{-\frac{1}{2}x} \mathrm{d}x \\
&= \frac{2\pi^{\frac{m}{2}}}{a\Gamma\left(\frac{m}{2}\right)} \cdot \frac{\Gamma\left(\frac{m}{a}\right)}{\left(\frac{1}{2}\right)^{\frac{m}{a}}} \\
&= \frac{2^{\frac{m}{a}+1}\pi^{\frac{m}{2}}\Gamma\left(\frac{m}{a}\right)}{a\Gamma\left(\frac{m}{2}\right)}.
\end{aligned}$$

Therefore the desired normalization factor is

$$\gamma^*_{\text{Gauss}}(a, b, m) = \frac{a\Gamma\left(\frac{m}{2}\right)}{2^{\frac{m}{a}+1}\pi^{\frac{m}{2}}\Gamma\left(\frac{m}{a}\right)}.$$

Since the integral formula used above is valid for $\beta > 0$, we obtain that the Gaussian function can be scaled to a unit integral provided $\frac{m}{a} - 1 > -1$, or equivalently $\frac{m}{a} > 0$. Since necessarily $m \geq 1$, the condition on the parameter is $a > 0$. The other parameter b is introduced here only to obtain an interface compatible with the Cauchy function.

Note that for $a = 2$ we obtain the well-known special case

$$\gamma^*_{\text{Gauss}}(2, 0, m) = \frac{1}{(2\pi)^{\frac{m}{2}}}.$$

Mahalanobis Distance

The above derivations, by using the Jacobi formula for the surface area of a (hyper-)sphere, implicitly assume that the Euclidian distance is used to measure the radius. However, they can easily be extended to the Mahalanobis distance, where the transformation brought about by the covariance matrix $\boldsymbol{\Sigma}$ has to be taken into account. Of this transformation only the scaling, captured—w.r.t. the volume of the infinitesimally thin (hyper-)sphere shells—in $\sqrt{|\boldsymbol{\Sigma}|}$ (see Section A.4 for details), is relevant for the normalization. That is, the normalization factor changes to

$$\gamma(a, b, m, \boldsymbol{\Sigma}) = \frac{\gamma^*(a, b, m)}{\sqrt{|\boldsymbol{\Sigma}|}} = \gamma^*(a, b, m) \cdot |\boldsymbol{\Sigma}|^{-\frac{1}{2}},$$

where $\gamma^*(a, b, m)$ is the normalization factor as it was computed above.

A.2.2 Derivatives

For the gradient methods the derivatives of the radial functions w.r.t. the cluster parameters are needed. Here I consider only the most general case, in which a full covariance matrix $\mathbf{\Sigma}$ is used for the Mahalanobis distance. All other possibilities can be seen as special cases, namely that the covariance matrix is a diagonal matrix (for axes-parallel (hyper-)ellipsoids) or the unit matrix (for the standard Euclidean distance). That is, I consider the case in which the argument r (radius) of the radial function is computed as

$$r = d(\vec{x}, \vec{\mu}; \mathbf{\Sigma}) = \sqrt{(\vec{x} - \vec{\mu})^\top \mathbf{\Sigma}^{-1} (\vec{x} - \vec{\mu})},$$

where $\vec{\mu}$ is the mean vector and $\mathbf{\Sigma}$ the covariance matrix of the cluster.

Generalized Cauchy Function

Inserting the expression from the beginning of this section for the argument r into the generalized Cauchy function yields

$$f_{\text{Cauchy}}(d(\vec{x}, \vec{\mu}; \mathbf{\Sigma}), a, b) = \frac{1}{((\vec{x} - \vec{\mu})^\top \mathbf{\Sigma}^{-1} (\vec{x} - \vec{\mu}))^{\frac{a}{2}} + b}.$$

Taking the derivative of this expression w.r.t. the mean vector $\vec{\mu}$ yields

$$\begin{aligned}
&\nabla_{\vec{\mu}}\, f_{\text{Cauchy}}(d(\vec{x}, \vec{\mu}; \mathbf{\Sigma}), a, b) \\
&= \nabla_{\vec{\mu}}\, \frac{1}{((\vec{x} - \vec{\mu})^\top \mathbf{\Sigma}^{-1} (\vec{x} - \vec{\mu}))^{\frac{a}{2}} + b} \\
&= -\frac{1}{(((\vec{x} - \vec{\mu})^\top \mathbf{\Sigma}^{-1} (\vec{x} - \vec{\mu}))^{\frac{a}{2}} + b)^2} \cdot \nabla_{\vec{\mu}} ((\vec{x} - \vec{\mu})^\top \mathbf{\Sigma}^{-1} (\vec{x} - \vec{\mu}))^{\frac{a}{2}} \\
&= -\frac{a}{2} f^2_{\text{Cauchy}}(d(\vec{x}, \vec{\mu}; \mathbf{\Sigma}), a, b) \cdot ((\vec{x} - \vec{\mu})^\top \mathbf{\Sigma}^{-1} (\vec{x} - \vec{\mu}))^{\frac{a}{2} - 1} \\
&\qquad\qquad \cdot \nabla_{\vec{\mu}}\, (\vec{x} - \vec{\mu})^\top \mathbf{\Sigma}^{-1} (\vec{x} - \vec{\mu}) \\
&= -\frac{a}{2} f^2_{\text{Cauchy}}(d(\vec{x}, \vec{\mu}; \mathbf{\Sigma}), a, b) \cdot (d(\vec{x}, \vec{\mu}; \mathbf{\Sigma}))^{a-2} \cdot (-2) \mathbf{\Sigma}^{-1} (\vec{x} - \vec{\mu}) \\
&= a\, f^2_{\text{Cauchy}}(d(\vec{x}, \vec{\mu}; \mathbf{\Sigma}), a, b) \cdot (d(\vec{x}, \vec{\mu}; \mathbf{\Sigma}))^{a-2} \cdot \mathbf{\Sigma}^{-1} (\vec{x} - \vec{\mu}),
\end{aligned}$$

where the semi-last step follows from the fact that a covariance matrix $\mathbf{\Sigma}$ and consequently also its inverse $\mathbf{\Sigma}^{-1}$ are symmetric, i.e., $\mathbf{\Sigma}^{-1} = \mathbf{\Sigma}^{-1\top}$, and the minus sign stems from the inner derivative $\nabla_{\vec{\mu}}\, (\vec{x} - \vec{\mu})$ (see Section A.1

for a general consideration of derivatives w.r.t. a vector). For the important special case $a = 2$ and $b = 0$ we have

$$\begin{aligned}&\nabla_{\vec{\mu}}\, f_{\text{Cauchy}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),2,0)\\ &= \quad 2\, f^2_{\text{Cauchy}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),2,0))\cdot\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu}).\end{aligned}$$

For the other parameter of the distance measure, the covariance matrix $\mathbf{\Sigma}$, we consider the derivative w.r.t. its inverse $\mathbf{\Sigma}^{-1}$. Since the derivative is needed in Section 5.1 to compute a change of the covariance matrix $\mathbf{\Sigma}$, the derivative w.r.t. its inverse $\mathbf{\Sigma}^{-1}$ suffices, because we can obtain the new covariance matrix by changing its inverse with the help of the above derivative and then inverting this new inverse matrix.

For the derivative w.r.t. the inverse $\mathbf{\Sigma}^{-1}$ we get

$$\begin{aligned}&\nabla_{\mathbf{\Sigma}^{-1}}\, f_{\text{Cauchy}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),a,b)\\ &= \quad \nabla_{\mathbf{\Sigma}^{-1}}\, \frac{1}{((\vec{x}-\vec{\mu})^\top\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu}))^{\frac{a}{2}}+b}\\ &= \quad -\frac{a}{2}f^2_{\text{Cauchy}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),a,b)\cdot((\vec{x}-\vec{\mu})^\top\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu}))^{\frac{a}{2}-1}\\ &\qquad\qquad\qquad\qquad\cdot\nabla_{\mathbf{\Sigma}^{-1}}\,(\vec{x}-\vec{\mu})^\top\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu})\\ &= \quad -\frac{a}{2}f^2_{\text{Cauchy}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),a,b)\cdot(d(\vec{x},\vec{\mu};\mathbf{\Sigma}))^{a-2}\cdot(\vec{x}-\vec{\mu})(\vec{x}-\vec{\mu})^\top.\end{aligned}$$

(See Section A.1 for a general consideration of derivatives w.r.t. a matrix, especially the derivative of $\vec{v}^\top\,\mathbf{A}\vec{v}$, which leads to the outer product.)

For the important special case $a = 2$ and $b = 0$

$$\begin{aligned}&\nabla_{\mathbf{\Sigma}^{-1}}\, f_{\text{Cauchy}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),2,0)\\ &= \quad -f^2_{\text{Cauchy}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),2,0)\cdot(\vec{x}-\vec{\mu})(\vec{x}-\vec{\mu})^\top.\end{aligned}$$

Generalized Gaussian Function

Inserting the expression from the beginning of this section, i.e.,

$$r = d(\vec{x},\vec{\mu};\mathbf{\Sigma}) = \sqrt{(\vec{x}-\vec{\mu})^\top\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu})},$$

for the argument r into the generalized Gaussian function yields

$$f_{\text{Gauss}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),a,b) = \exp\left(-\frac{1}{2}\left((\vec{x}-\vec{\mu})^\top\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu})\right)^{\frac{a}{2}}\right).$$

Taking the derivative of this expression w.r.t. the mean vector $\vec{\mu}$ yields

$$\begin{aligned}
&\nabla_{\vec{\mu}}\, f_{\text{Gauss}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),a,b) \\
&= \nabla_{\vec{\mu}}\, \exp\left(-\frac{1}{2}((\vec{x}-\vec{\mu})^\top\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu}))^{\frac{a}{2}}\right) \\
&= -\frac{1}{2}\exp\left(-\frac{1}{2}\left((\vec{x}-\vec{\mu})^\top\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu})\right)^{\frac{a}{2}}\right)\cdot\nabla_{\vec{\mu}}((\vec{x}-\vec{\mu})^\top\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu}))^{\frac{a}{2}} \\
&= -\frac{a}{4}\,f_{\text{Gauss}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),a,b)\;\cdot((\vec{x}-\vec{\mu})^\top\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu}))^{\frac{a}{2}-1} \\
&\qquad\qquad\qquad\qquad\cdot\nabla_{\vec{\mu}}\,(\vec{x}-\vec{\mu})^\top\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu}) \\
&= -\frac{a}{4}\,f_{\text{Gauss}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),a,b)\;\cdot(d(\vec{x},\vec{\mu};\mathbf{\Sigma}))^{a-2}\cdot(-2)\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu}) \\
&= \frac{a}{2}\,f_{\text{Gauss}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),a,b)\;\cdot(d(\vec{x},\vec{\mu};\mathbf{\Sigma}))^{a-2}\cdot\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu})
\end{aligned}$$

(compare the derivations in the preceding paragraph as well as the considerations in Section A.1) and thus for the special case $a = 2$ and $b = 0$:

$$\begin{aligned}
&\nabla_{\vec{\mu}}\, f_{\text{Gauss}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),2,0) \\
&\quad= f_{\text{Gauss}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),2,0)\cdot\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu}).
\end{aligned}$$

For the other parameter of the distance measure, the covariance matrix $\mathbf{\Sigma}$, we consider again the derivative w.r.t. its inverse $\mathbf{\Sigma}^{-1}$:

$$\begin{aligned}
&\nabla_{\mathbf{\Sigma}^{-1}}\, f_{\text{Gauss}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),a,b) \\
&= \nabla_{\mathbf{\Sigma}^{-1}}\, \exp\left(-\frac{1}{2}((\vec{x}-\vec{\mu})^\top\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu}))^{\frac{a}{2}}\right) \\
&= -\frac{a}{4}f_{\text{Gauss}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),a,b)\;\cdot((\vec{x}-\vec{\mu})^\top\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu}))^{\frac{a}{2}-1} \\
&\qquad\qquad\qquad\qquad\cdot\nabla_{\mathbf{\Sigma}^{-1}}\,(\vec{x}-\vec{\mu})^\top\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu}) \\
&= -\frac{a}{4}f_{\text{Gauss}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),a,b)\;\cdot(d(\vec{x},\vec{\mu};\mathbf{\Sigma}))^{a-2}\cdot(\vec{x}-\vec{\mu})(\vec{x}-\vec{\mu})^\top
\end{aligned}$$

(compare again the derivations in the preceding paragraph as well as the considerations in Section A.1) and thus for the special case $a = 2$ and $b = 0$:

$$\begin{aligned}
&\nabla_{\mathbf{\Sigma}^{-1}}\, f_{\text{Gauss}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),2,0) \\
&\quad= -\frac{1}{2}f_{\text{Gauss}}(d(\vec{x},\vec{\mu};\mathbf{\Sigma}),2,0)\cdot(\vec{x}-\vec{\mu})(\vec{x}-\vec{\mu})^\top.
\end{aligned}$$

Normalization Factor

If the radial function is required to have unit integral over the data space so that it can be interpreted as a probability density function, we have to consider an additional factor in the derivatives, namely the normalization factor $\gamma^*(a, b, m)$ or $\gamma(a, b, m, \mathbf{\Sigma})$ that was computed in Section A.2.1. If we do not use a Mahalanobis distance, this factor depends only on the parameters a and b of the radial function and the number m of dimensions of the data space, all of which are constants. Hence $\gamma^*(a, b, m)$ is a constant and thus simply reproduces when taking the derivative.

If, however, we use a Mahalanobis distance, things are slightly more complicated. Although the normalization factor does not depend on the center vector $\vec{\mu}$ and thus simply reproduces when computing the derivative w.r.t. $\vec{\mu}$, it does depend on the covariance matrix $\mathbf{\Sigma}$. As a consequence, we have to apply the product rule, for which we need the derivative of the normalization factor. This can be obtained generally, as it is

$$\gamma(a, b, m, \mathbf{\Sigma}) = \frac{\gamma^*(a, b, m)}{\sqrt{|\mathbf{\Sigma}|}} = \gamma^*(a, b, m) \cdot |\mathbf{\Sigma}|^{-\frac{1}{2}}$$

(see Section A.2.1), where $\gamma^*(a, b, m)$ is constant as argued above.

Since we are considering the derivative w.r.t. the inverse covariance matrix, it is best to exploit $|\mathbf{\Sigma}^{-1}| = |\mathbf{\Sigma}|^{-1}$ and hence to write the above as

$$\gamma(a, b, m, \mathbf{\Sigma}) = \gamma^*(a, b, m) \cdot |\mathbf{\Sigma}^{-1}|^{\frac{1}{2}}.$$

Then we have (exploiting $\nabla_{\mathbf{A}}|\mathbf{A}| = |\mathbf{A}|\mathbf{A}^{-1}$ as derived in Section A.1)

$$\begin{aligned}
\nabla_{\mathbf{\Sigma}^{-1}} \gamma(a, b, m, \mathbf{\Sigma}) &= \gamma^*(a, b, m) \cdot \nabla_{\mathbf{\Sigma}^{-1}} |\mathbf{\Sigma}^{-1}|^{\frac{1}{2}} \\
&= -\frac{\gamma^*(a, b, m)}{2} \cdot |\mathbf{\Sigma}^{-1}|^{-\frac{1}{2}} \cdot \nabla_{\mathbf{\Sigma}^{-1}} |\mathbf{\Sigma}^{-1}| \\
&= -\frac{\gamma^*(a, b, m)}{2} \cdot |\mathbf{\Sigma}^{-1}|^{-\frac{1}{2}} \cdot |\mathbf{\Sigma}^{-1}| \cdot \mathbf{\Sigma} \\
&= -\frac{\gamma^*(a, b, m)}{2} \cdot |\mathbf{\Sigma}^{-1}|^{\frac{1}{2}} \cdot \mathbf{\Sigma} \\
&= -\frac{\gamma^*(a, b, m)}{2} \cdot |\mathbf{\Sigma}|^{-\frac{1}{2}} \cdot \mathbf{\Sigma} \\
&= -\frac{\gamma(a, b, m, \mathbf{\Sigma})}{2} \cdot \mathbf{\Sigma}.
\end{aligned}$$

Compared to standard differentiation rules, this is an intuitive result.

A.3 Cholesky Decomposition

For one-dimensional data the variance σ^2 is not a very intuitive measure of the dispersion of the data around the mean value, because it is a kind of average *squared* distance to the mean value.[3] A more intuitive measure is the square root of the variance, that is, the standard deviation $\sigma = \sqrt{\sigma^2}$, because it has the same unit as the data and the mean value.

If we are working with multi-dimensional data, their dispersion around a mean vector $\vec{\mu}$ is described in the general case by a covariance matrix $\mathbf{\Sigma}$, which is also a quadratic measure and thus a bit difficult to interpret. It would be convenient if we could do the same as in the one-dimensional case, that is, if we could compute a "square root" of $\mathbf{\Sigma}$ in order to obtain a more intuitive measure. Fortunately, this is indeed possible. Formally we try to find a lower (or left) triangular[4] matrix $\mathbf{L}$, such that

$$\mathbf{\Sigma} = \mathbf{L}\mathbf{L}^\top,$$

where $\top$ denotes transposition. This is the so-called **Cholesky decomposition**, which exists for any symmetric and positive definite[5] matrix. Writing out the above equation in individual matrix elements, one easily obtains the following two formulae [Press *et al.* 1992]:

$$\forall i; 1 \leq i \leq m: \qquad l_{ii} = \sqrt{\sigma_i^2 - \sum_{k=1}^{i-1} l_{ik}^2},$$

and

$$\forall i, j; 1 \leq i < j \leq m: \qquad l_{ji} = \frac{1}{l_{ii}} \left(\sigma_{ij} - \sum_{k=1}^{i-1} l_{ik} l_{jk} \right).$$

In order to understand the meaning of the matrix $\mathbf{L}$, let us consider the two-dimensional case. Then we have

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix} \qquad \text{and} \qquad \mathbf{L} = \begin{pmatrix} \sigma_x & 0 \\ \frac{\sigma_{xy}}{\sigma_x} & \frac{1}{\sigma_x}\sqrt{\sigma_x^2\sigma_y^2 - \sigma_{xy}^2} \end{pmatrix}.$$

[3] It is exactly the average squared distance to the mean if the maximum likelihood estimator $\hat{\sigma}^2 = \frac{1}{n}\sum_{j=1}^{n}(x_j - \mu_j)^2$ is used. However, it is more common to rely on the unbiased estimator $\hat{\sigma}^2 = \frac{1}{n-1}\sum_{j=1}^{n}(x_j - \mu_j)^2$, which is not exactly an average. This distinction is not relevant for the considerations in this section, though.

[4] An $m \times m$ matrix is called **lower** or **left triangular** iff all elements above its diagonal are zero and **upper** or **right triangular** iff all elements below its diagonal are zero.

[5] An $m \times m$ matrix $\mathbf{A}$ is called **positive definite** iff for all m-dimensional vectors $\vec{v} \neq \vec{0}$, it is $\vec{v}^\top \mathbf{A} \vec{v} > 0$.

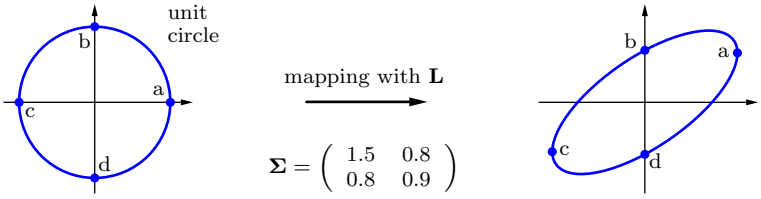

Figure A.1: Intuitive interpretation of the result $\mathbf{L}$ of the Cholesky decomposition of a 2×2 covariance matrix $\mathbf{\Sigma}$.

This matrix $\mathbf{L}$ describes a mapping that transforms a unit circle into an ellipse with general orientation. An example is shown in figure A.1. This ellipse is the set of all points that have distance 1 to the origin of the coordinate system w.r.t. the Mahalanobis distance based on $\mathbf{\Sigma}$, that is, for all points $\vec{v}$ on this ellipse it is $\sqrt{\vec{v}^\top \mathbf{\Sigma}^{-1} \vec{v}} = 1$.

Consequently, if $\mathbf{\Sigma}$ is used as a parameter of a multi-dimensional probability density function (for instance, the Gaussian function), this ellipse is a contour line, that is, all points on it have the same probability density.

Besides providing an intuitive interpretation of a covariance matrix $\mathbf{\Sigma}$, Cholesky decomposition is very convenient if one needs to compute the determinant $|\mathbf{\Sigma}|$ or the inverse matrix $\mathbf{\Sigma}^{-1}$. For the determinant, one only has to notice that

$$|\mathbf{\Sigma}| = |\mathbf{L}\mathbf{L}^\top| = |\mathbf{L}||\mathbf{L}^\top| = |\mathbf{L}|^2$$

and that the determinant $|\mathbf{L}|$ can be computed as the product of the diagonal elements of $\mathbf{L}$, since $\mathbf{L}$ is a triangular matrix.

To compute the inverse $\mathbf{\Sigma}^{-1}$, one starts from

$$\mathbf{1} = \mathbf{\Sigma}\mathbf{\Sigma}^{-1} = \mathbf{L}\mathbf{L}^\top \mathbf{\Sigma}^{-1},$$

where $\mathbf{1}$ is the $m \times m$ unit matrix, and obtains the solution by first doing a forward substitution[6] with $\mathbf{L}$ (recall that $\mathbf{L}$ is lower triangular) and the column vectors of the unit matrix $\mathbf{1}$ to get

$$\mathbf{L}^{-1}\mathbf{1} = \mathbf{L}^{-1} = \mathbf{L}^\top \mathbf{\Sigma}^{-1}$$

[6] In a forward substitution one finds the original of a vector $\vec{v}$ w.r.t. a lower (or left) triangular $m \times m$ matrix $\mathbf{L}$ (that is, the vector $\vec{z}$ satisfying $\mathbf{L}\vec{z} = \vec{v}$ and thus $\vec{z} = \mathbf{L}^{-1}\vec{v}$) by solving the equations $v_i = \sum_{j=1}^{i} l_{ij} z_j$ in ascending order of i, exploiting that the values of z_j with $j < i$ have already been determined in preceding steps and thus z_i can be computed from known values as $z_i = \frac{1}{l_{ii}} \left(v_i - \sum_{j=1}^{i-1} l_{ij} z_j \right)$.

and then a backward substitution[7] with $\mathbf{L}^\top$ (note that $\mathbf{L}^\top$ is upper triangular) to arrive at

$$(\mathbf{L}^\top)^{-1}\mathbf{L}^{-1} = \mathbf{\Sigma}^{-1}.$$

For such an inversion, Cholesky decomposition is about a factor of 2 faster than the more general **LU decomposition** with forward/backward substitution [Press *et al.* 1992].

A.4 Eigenvalue Decomposition

Cholesky decomposition, as it was discussed in the preceding section, yields a "square root" of a covariance matrix $\mathbf{\Sigma}$ and thus helps to interpret it. An even better way of computing such an analog of standard deviation is **eigenvalue decomposition**. It yields a matrix $\mathbf{T}$ that is composed of a rotation matrix and a diagonal matrix and that satisfies

$$\mathbf{\Sigma} = \mathbf{T}\mathbf{T}^\top.$$

Such a matrix $\mathbf{T}$ can be found by computing the eigenvalues and eigenvectors of $\mathbf{\Sigma}$ and by writing the resulting decomposition of $\mathbf{\Sigma}$ as a product of two matrices, which are transposes of each other.

Let us start by recalling that an $m \times m$ matrix $\mathbf{A}$ is said to have an **eigenvalue** λ and corresponding **eigenvector** $\vec{v}$ if

$$\mathbf{A}\vec{v} = \lambda\vec{v}.$$

Since obviously any multiple of an eigenvector $\vec{v}$ is also an eigenvector, one usually considers only normalized eigenvectors, i.e. vectors $\vec{v}$ with $|\vec{v}| = 1$. Furthermore, it is clear that the above equation can hold only if

$$|\mathbf{A} - \lambda\mathbf{1}| = 0,$$

where $\mathbf{1}$ is the $m \times m$ unit matrix. The left side of this equation is a polynomial of degree m in λ, the roots of which are the eigenvalues (necessary condition for the equation system to have a (unique) solution). Consequently, an $m \times m$ matrix always has m (not necessarily distinct) eigenvalues and thus also m corresponding (not necessarily distinct) eigenvectors.

[7] In a backward substitution one finds the original of a vector $\vec{v}$ w.r.t. an upper (or right) triangular $m \times m$ matrix $\mathbf{U}$ (that is, the vector $\vec{z}$ satisfying $\mathbf{U}\vec{z} = \vec{v}$ and thus $\vec{z} = \mathbf{U}^{-1}\vec{v}$) by solving the equations $v_i = \sum_{j=i}^{m} u_{ij}z_j$ in descending order of i, exploiting that the values of z_j with $j > i$ have already been determined in preceding steps and thus z_i can be computed from known values as $z_i = \frac{1}{u_{ii}}\left(v_i - \sum_{j=i+1}^{m} u_{ij}z_j\right)$.

This enables us to form an $m \times m$ matrix $\mathbf{R}$, the columns of which are the eigenvectors of $\mathbf{A}$, for which it is obviously (because it is only a way of writing all m eigenvalue/eigenvector equations in one formula)

$$\mathbf{A}\mathbf{R} = \mathbf{R}\,\mathrm{diag}(\lambda_1, \ldots, \lambda_m),$$

where $\lambda_1, \ldots, \lambda_m$ are the m eigenvalues of $\mathbf{A}$ and $\mathrm{diag}(\ldots)$ denotes a matrix with the given values on its diagonal (from left top to right bottom) and all other elements zero. Hence we can write the matrix $\mathbf{A}$ as

$$\mathbf{A} = \mathbf{R}\,\mathrm{diag}(\lambda_1, \ldots, \lambda_m)\,\mathbf{R}^{-1}.$$

For a real, symmetric[8] matrix it can be shown that all eigenvalues are real, and if the matrix is also positive definite[9], that all eigenvalues are positive. Furthermore, for a real, symmetric matrix all eigenvectors are real and orthonormal[10] (provided that orthogonal eigenvectors are chosen for multiple—also called **degenerate**—eigenvalues, which is always possible) [Press *et al.* 1992]. In this case the matrix $\mathbf{R}$ is orthogonal[11], so we obtain

$$\mathbf{A} = \mathbf{R}\,\mathrm{diag}(\lambda_1, \ldots, \lambda_m)\,\mathbf{R}^{\top}.$$

If $\mathbf{A}$ is positive definite and thus the eigenvalues $\lambda_1, \ldots, \lambda_m$ are all positive (see above), this formula may also be written as

$$\begin{aligned}
\mathbf{A} &= \mathbf{R}\,\mathrm{diag}\left(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_m}\right)\,\mathrm{diag}\left(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_m}\right)\,\mathbf{R}^{\top} \\
&= \mathbf{R}\,\mathrm{diag}\left(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_m}\right)\,\left(\mathbf{R}\,\mathrm{diag}\left(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_m}\right)\right)^{\top} \\
&= \mathbf{T}\,\mathbf{T}^{\top},
\end{aligned}$$

with $\mathbf{T} = \mathbf{R}\,\mathrm{diag}\left(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_m}\right)$. This matrix $\mathbf{T}$ is easily interpreted: the diagonal matrix $\mathrm{diag}\left(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_m}\right)$ describes a scaling of the axes and the matrix $\mathbf{R}$ effects a rotation of the coordinate system. (Note that, since the eigenvectors are normalized, it is $|\mathbf{R}| = 1$ and thus $\mathbf{R}$ does not lead to any further scaling, but describes a pure rotation.)

For a covariance matrix $\mathbf{\Sigma}$ the scaling factors, i.e. the square roots of the eigenvalues of $\mathbf{\Sigma}$, are the standard deviations in the eigensystem of $\mathbf{\Sigma}$ (i.e.,

[8] See footnote 1 on page 268 for a definition.
[9] See footnote 5 on page 278 for a definition.
[10] That is, for any two (normalized) eigenvectors $\vec{v}_1$ and $\vec{v}_2$, $\vec{v}_1 \neq \vec{v}_2$, it is $\vec{v}_1^{\top}\vec{v}_2 = 0$.
[11] A matrix $\mathbf{A}$ is called **orthogonal** if its transpose equals its inverse, that is, if $\mathbf{A}^{\top}\mathbf{A} = \mathbf{A}\mathbf{A}^{\top} = \mathbf{1}$.

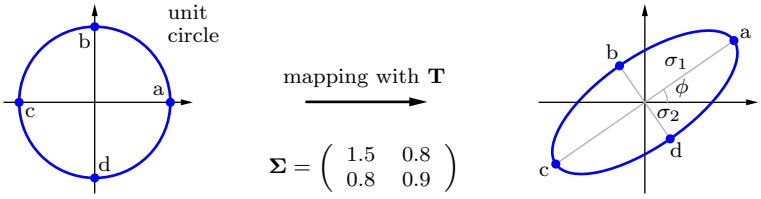

Figure A.2: Intuitive interpretation of the result $\mathbf{T}$ of the eigenvalue decomposition of a 2×2 covariance matrix $\mathbf{\Sigma}$: scaling and rotation.

the coordinate system formed by the eigenvectors of $\mathbf{\Sigma}$). To understand this better, let us consider the two-dimensional case, where we have

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix} \quad \text{and} \quad \mathbf{T} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix},$$

$$\text{with} \quad \phi = \frac{1}{2} \arctan \frac{2\sigma_{xy}}{\sigma_x^2 - \sigma_y^2} \qquad \text{(rotation angle)},$$

$$\sigma_1 = \sqrt{\sigma_x^2 \cos^2\phi + \sigma_y^2 \sin^2\phi + 2\sigma_{xy} \sin\phi \cos\phi},$$

$$\sigma_2 = \sqrt{\sigma_x^2 \sin^2\phi + \sigma_y^2 \cos^2\phi - 2\sigma_{xy} \sin\phi \cos\phi}.$$

Like the result matrix $\mathbf{L}$ of Cholesky decomposition (cf. Section A.3), the matrix $\mathbf{T}$ describes a mapping that transforms a unit circle into an ellipse with general orientation: $\mathrm{diag}(\sigma_1, \sigma_2)$ describes a scaling of the axes with the standard deviations in the eigensystem of $\mathbf{\Sigma}$, so that we obtain an axes-parallel ellipse, which is then rotated by a mapping with $\mathbf{R}$. An example, which uses the same covariance matrix $\mathbf{\Sigma}$ as the example for Cholesky decomposition in Figure A.1 on page 279, is shown in Figure A.2. Note that individual points are mapped differently with $\mathbf{T}$ than with the matrix $\mathbf{L}$ of Cholesky decomposition, but that the resulting ellipses are identical.

The above eigenvalue decomposition for two-dimensional covariance matrices was computed with **Jacobi transformation**, which is the most convenient method for low-dimensional matrices, in particular two-dimensional ones. For bigger matrices, however, Jacobi transformation is computationally less efficient than a combination of **Householder transformation** to tridiagonal form and an application of the **QR algorithm**. Details about all of these methods can be found, for example, in [Press *et al.* 1992].

Since the result matrix $\mathbf{T}$ of eigenvalue decomposition consists of a diagonal matrix describing a scaling and a rotation matrix, it is even better interpretable than the result matrix $\mathbf{L}$ of Cholesky decomposition. Furthermore, once we have the matrices $\mathbf{R}$ and $\mathrm{diag}(\lambda_1, \ldots, \lambda_m)$, we can also (and even more easily) compute the determinant $|\mathbf{\Sigma}|$, namely as

$$|\mathbf{\Sigma}| = |\mathbf{R}|\,|\mathrm{diag}(\lambda_1, \ldots, \lambda_m)|\,|\mathbf{R}^\top| = |\mathrm{diag}(\lambda_1, \ldots, \lambda_m)| = \prod_{i=1}^{m} \lambda_i$$

(since $|\mathbf{R}| = |\mathbf{R}^\top| = 1$), and the inverse $\mathbf{\Sigma}^{-1}$, namely as

$$\mathbf{\Sigma}^{-1} = \mathbf{R}\,\mathrm{diag}\left(\frac{1}{\lambda_1}, \ldots, \frac{1}{\lambda_m}\right)\mathbf{R}^\top.$$

However, if we do not need the eigenvalues of $\mathbf{\Sigma}$ explicitely, Cholesky decomposition is clearly preferable for implementations. The main disadvantage of eigenvalue decomposition is that it is computationally even more expensive than **LU decomposition** with forward/backward substitution, which, as remarked above, is already slower than Cholesky decomposition by roughly a factor of 2 [Press *et al.* 1992]. Therefore eigenvalue decomposition should only be used if absolutely necessary (like, for instance, for the alternative version of shape regularization described in Section 6.1.2, which limits the ratio of the largest to the smallest eigenvalue to a user-specified value).

Nevertheless, the above formulae are useful in another way: we saw that the eigenvalues λ_i, $1 \leq i \leq m$, are the squared lengths of the semi-major axes of the (hyper-)ellipsoid a unit (hyper-)sphere is mapped to by the transformation matrix $\mathbf{T}$ derived above (cf. Figure A.2). Therefore the formula for the determinant shows that $\sqrt{|\mathbf{\Sigma}|}$ is proportional to the volume of this (hyper-)ellipsoid. More precisely and more generally, it shows that

$$V_m(r) = \frac{\pi^{\frac{m}{2}} r^m}{\Gamma\left(\frac{m}{2}+1\right)} \sqrt{|\mathbf{\Sigma}|}$$

is the volume of the m-dimensional (hyper-)ellipsoid a (hyper-)sphere with radius r is mapped to. Note that if $\mathbf{\Sigma}$ is the unit matrix and thus $|\mathbf{\Sigma}| = 1$, this formula simplifies to the Jacobi formula for the volume of the m-dimensional (hyper-)sphere [Bronstein *et al.* 1996, 0.1.6, p. 15]. Note also that this formula justifies the modified normalization factor for the Mahalanobis distance given in Section A.2.1: since the volume of the (hyper-) ellipsoid (and thus the volume of a (hyper-)ellipsoid shell) is $\sqrt{|\mathbf{\Sigma}|}$ times the volume of the original (hyper-)sphere (shell), we have to divide by $\sqrt{|\mathbf{\Sigma}|}$.

A.5 Singular Value Decomposition

Singular value decomposition is very similar to eigenvalue decomposition in as far as it also yields a decomposition of a given matrix into three matrices, one of them diagonal. In contrast to eigenvalue decomposition, however, it is not restricted to symmetric matrices, although for these it coincides with eigenvalue decomposition. Singular value decomposition is based on the following theorem from linear algebra [Golub and Van Loan 1989, Press *et al.* 1992]: Any $n \times m$ matrix $\mathbf{A}$ with n (the number of rows) greater than or equal to m (the number of columns) can be written as the product of an $n \times m$ column-orthogonal matrix $\mathbf{U}$, an $m \times m$ diagonal matrix $\mathbf{W} = \operatorname{diag}(w_1, \ldots, w_m)$, and the transpose of an $m \times m$ orthogonal matrix $\mathbf{V}$:

$$\mathbf{A} = \mathbf{U}\mathbf{W}\mathbf{V}^\top = \mathbf{U} \operatorname{diag}(w_1, \ldots, w_m)\, \mathbf{V}^\top.$$

Both matrices $\mathbf{U}$ and $\mathbf{V}$ are orthogonal in the sense that their columns are orthonormal, i.e. pairwise orthogonal and of unit length. Formally, it is

$$\mathbf{U}^\top\mathbf{U} = \mathbf{V}^\top\mathbf{V} = \mathbf{1},$$

where $\mathbf{1}$ is the $m \times m$ unit matrix. Note that since $\mathbf{V}$ is a square matrix ($m \times m$), it is also row-orthonormal, i.e. $\mathbf{V}\mathbf{V}^\top = \mathbf{1}$.

The elements $w_1, \ldots, w_m$ of the diagonal matrix are the so-called *singular values*, from which this method derives its name. The reason for this name is that if singular value decomposition is applied to a square matrix, the values $w_1, \ldots, w_m$ provide an indication of "how singular" a matrix is, that is, how close numerically a matrix is to a singular matrix, so that computations on a finite precision machine may mistake it for an actually singular matrix [Press *et al.* 1992]. This indication is provided by the so-called *condition number* of a square matrix, which is the ratio

$$c = \frac{\max_{i=1}^{m} |w_i|}{\min_{i=1}^{m} |w_i|}.$$

Its value is infinity if the matrix is actually singular, because then at least one w_i vanishes. If c is very large, the matrix is called *ill-conditioned*.

Singular value decomposition is one of the most robust methods to handle least squares problems (see next section), because of its ability to detect and handle ill-conditioned matrices [Press *et al.* 1992]. Details about this method and a stable algorithm to compute the singular value decomposition of a matrix can be found, for example, in [Golub and Van Loan 1989]. An implementation of this algorithm in C is given in [Press *et al.* 1992].

A.6 Multilinear Regression

Multilinear regression is a data analysis method that is based on the assumption that a so-called **response variable** y is a linear function of a set $x_1, \ldots, x_c$ of so-called **regressor variables**, i.e.,

$$y = \theta + \sum_{i=1}^{c} w_i \, x_i.$$

The coefficients w_i and the offset θ (also known as *bias value* or *intercept term*) are to be determined in such a way as to fit a given data set. If this data set is represented by

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1m} \\ \vdots & & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{nm} \end{pmatrix} \quad \text{and} \quad \vec{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix},$$

where x_{ij}, $1 \leq i \leq n$, $1 \leq j \leq m$, is the value of the j-th regressor variable in the i-th example case and y_i, $1 \leq i \leq n$, is the value of the response variable in the i-th example case, we can state the task of multilinear regression as the problem to find an approximate solution of the (usually overdetermined) linear equation system

$$\mathbf{X}\vec{w} = \vec{y},$$

where the vector $\vec{w} = (\theta, w_1, \ldots, w_c)^\top$ represents the parameters to be determined. (Note that the first column of the matrix $\mathbf{X}$, which is set to 1, corresponds to the offset θ.) The approximation quality is measured by the sum of squared errors, i.e. the squared differences between the value of y as it is computed from the above function and the corresponding value of y in the data set. This sum of squared errors is to be minimized w.r.t. $\vec{w}$, i.e.

$$(\mathbf{X}\vec{w} - \vec{y})^\top (\mathbf{X}\vec{w} - \vec{y}) = \min.$$

Since it is a necessary condition for a minimum that the partial derivatives of this functional w.r.t. the elements of $\vec{w}$ vanish, we have

$$\nabla_{\vec{w}} \, (\mathbf{X}\vec{w} - \vec{y})^\top (\mathbf{X}\vec{w} - \vec{y}) = \vec{0},$$

where the differential operator $\nabla_{\vec{w}}$ is defined as

$$\nabla_{\vec{w}} = \left(\frac{\partial}{\partial \theta}, \frac{\partial}{\partial w_1}, \ldots, \frac{\partial}{\partial w_c} \right)^\top.$$

The derivative w.r.t. $\vec{w}$ can easily be computed if one remembers that the differential operator $\nabla_{\vec{w}}$ behaves formally like a vector that is "multiplied" from the left to the sum of squared errors. Alternatively, one may write out the sum of squared errors and take the derivatives componentwise. Here we rely on the former, much more convenient method and obtain

$$\begin{aligned}
\vec{0} &= \nabla_{\vec{w}} (\mathbf{X}\vec{w} - \vec{y})^\top (\mathbf{X}\vec{w} - \vec{y}) \\
&= \left(\nabla_{\vec{w}} (\mathbf{X}\vec{w} - \vec{y})\right)^\top (\mathbf{X}\vec{w} - \vec{y}) + \left((\mathbf{X}\vec{w} - \vec{y})^\top \left(\nabla_{\vec{w}} (\mathbf{X}\vec{w} - \vec{y})\right)\right)^\top \\
&= \left(\nabla_{\vec{w}} (\mathbf{X}\vec{w} - \vec{y})\right)^\top (\mathbf{X}\vec{w} - \vec{y}) + \left(\nabla_{\vec{w}} (\mathbf{X}\vec{w} - \vec{y})\right)^\top (\mathbf{X}\vec{w} - \vec{y}) \\
&= 2\mathbf{X}^\top(\mathbf{X}\vec{w} - \vec{y}) \\
&= 2\mathbf{X}^\top\mathbf{X}\vec{w} - 2\mathbf{X}^\top\vec{y},
\end{aligned}$$

from which the linear system of so-called **normal equations**

$$\mathbf{X}^\top\mathbf{X}\vec{w} = \mathbf{X}^\top\vec{y}$$

follows. This system may be solved by standard methods from linear algebra, for instance, by computing the inverse of $\mathbf{X}^\top\mathbf{X}$, which leads to

$$\vec{w} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\vec{y}.$$

The expression $(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$ occurring here is well known as the (Moore-Penrose) **pseudo inverse** of the matrix $\mathbf{X}$ [Albert 1972].

For implementations it is worth noting that $\frac{1}{n}\mathbf{X}^\top\mathbf{X}$ is the covariance matrix[12] of the extended input data (extended in the sense that there is an artificial extra regressor variable $x_0 \equiv 1$, which corresponds to the offset θ—see the first column of $\mathbf{X}$) and thus Cholesky decomposition may be used to compute the inverse $(\mathbf{X}^\top\mathbf{X})^{-1}$ efficiently (cf. Section A.3). An alternative is singular value decomposition (cf. Section A.5), which is more robust and can find a solution even if $\mathbf{X}^\top\mathbf{X}$ is ill-conditioned.[13]

Ill-conditioned matrices may also be handled by so-called *Tikhonov regularization* [Tikhonov and Arsenin 1977], also known as *ridge regression* [Cristianini and Shawe-Taylor 2000]. It consists in computing $\vec{w}$ as

$$\vec{w} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{1})^{-1}\mathbf{X}^\top\vec{y},$$

[12] Computed by maximum likelihood estimation. For the unbiased estimator the fraction $\frac{1}{n}$ has to be replaced by $\frac{1}{n-1}$.

[13] See Section A.5 for a general explanation of the *condition number* of a matrix. Here it is ratio of the largest to the smallest eigenvalue of the matrix $\mathbf{X}^\top\mathbf{X}$. A matrix is called *ill-conditioned* if its condition number is very large.

where $\lambda \in \mathbb{R}^+$ is a user-defined regularization parameter. The idea underlying this approach is that adding a λ-multiple of a $(c+1) \times (c+1)$ unit matrix $\mathbf{1}$ to $\mathbf{X}^\top\mathbf{X}$ "shifts" the eigenvalues of $\mathbf{X}^\top\mathbf{X}$ by λ and thus improves the condition number of the matrix (i.e., makes it "less singular" or "more regular"). This can easily be seen by considering the defining equations of the eigenvalues and eigenvectors of an $m \times m$ matrix $\mathbf{A}$, i.e. $\mathbf{A}\vec{v}_i = \lambda_i\vec{v}_i$, $i = 1, \ldots, m$, and replacing $\mathbf{A}$ by $\mathbf{A}' = \mathbf{A} + \lambda\mathbf{1}$. Then we get

$$\begin{aligned} \mathbf{A}'\vec{v}_i = \lambda_i'\vec{v}_i \quad &\Leftrightarrow \quad (\mathbf{A} + \lambda\mathbf{1})\vec{v}_i = \lambda_i'\vec{v}_i \\ &\Leftrightarrow \quad \mathbf{A}\vec{v}_i + \lambda\vec{v}_i = \lambda_i'\vec{v}_i \\ &\Leftrightarrow \quad \mathbf{A}\vec{v}_i = (\lambda_i' - \lambda)\vec{v}_i, \end{aligned}$$

that is, $\lambda_i = \lambda_i' - \lambda$ and thus $\lambda_i' = \lambda_i + \lambda$. Note also that Tikhonov regularization/ridge regression minimizes the *penalized sum of squared errors*

$$(\mathbf{X}\vec{w} - \vec{y})^\top(\mathbf{X}\vec{w} - \vec{y}) + \lambda\vec{w}^\top\vec{w}.$$

In this view λ controls a tradeoff between a least sum of squared errors and a "short" solution vector $\vec{w}$ [Cristianini and Shawe-Taylor 2000].

A.7 Matrix Inversion Lemma

The matrix inversion lemma, also known as the **Sherman-Morrison formula**, states how the inverses of two matrices relate to each other if one of the matrices results from the other by adding an outer product of vectors. That is, for matrices $\mathbf{A}$ and $\mathbf{B} = \mathbf{A} + \vec{u}\,\vec{v}^\top$, it states how $\mathbf{B}^{-1}$ relates to $\mathbf{A}^{-1}$. The matrix inversion lemma can be derived as follows [Press *et al.* 1992]:

$$\begin{aligned} \mathbf{B}^{-1} &= (\mathbf{A} + \vec{u}\,\vec{v}^\top)^{-1} \\ &= (\mathbf{1} + \mathbf{A}^{-1}\vec{u}\,\vec{v}^\top)^{-1}\mathbf{A}^{-1} \\ &= (\mathbf{1} - \mathbf{A}^{-1}\vec{u}\,\vec{v}^\top + \mathbf{A}^{-1}\vec{u}\,\vec{v}^\top\mathbf{A}^{-1}\vec{u}\,\vec{v}^\top - \ldots)\mathbf{A}^{-1} \\ &= \mathbf{A}^{-1} - \mathbf{A}^{-1}\vec{u}\,\vec{v}^\top\mathbf{A}^{-1}(1 - \lambda + \lambda^2 - \ldots) \\ &= \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\vec{u}\,\vec{v}^\top\mathbf{A}^{-1}}{1+\lambda}, \end{aligned}$$

where $\lambda = \vec{v}^\top\mathbf{A}^{-1}\vec{u}$. The second step in the above derivation factors out the matrix $\mathbf{A}^{-1}$, the third step is a formal power expansion of the first factor, and in the fourth step the associativity of matrix (and vector) multiplications is used to factor out the scalar λ. Finally, in the fifth step, the well-known formula for geometric series yields the result.

A.8 Lagrange Theory

In this thesis a recurring problem, encountered particularly often in Section 5.2, is to optimize some functional $F(\vec{\theta})$, which depends on a set $\vec{\theta} = \{\theta_1, \ldots, \theta_r\}$ of parameters, subject to a set of constraints $C_j(\vec{\theta}) = 0$, $1 \leq j \leq s$. In such a situation the standard optimization approach to set the gradient of the functional $F(\vec{\theta})$ w.r.t. $\vec{\theta}$ equal to the null vector (or equivalently: to set all partial derivatives w.r.t. the different θ_i, $1 \leq i \leq r$, equal to zero, which is a necessary condition for an optimum—see Section A.6) does not work: the result cannot be guaranteed to satisfy the constraints. Or stated the other way round: at a (local) optimum *in the constrained subspace* the gradient of $F(\vec{\theta})$ need not vanish.

One way to handle this problem is to exploit the dependence between the parameters, which is brought about by the constraints $C_j(\vec{\theta}) = 0$, to express some parameters in terms of the others and thus to reduce $\vec{\theta}$ to a set $\vec{\theta}'$ of independent parameters. The resulting independent parameters are well known in theoretical physics as **generalized coordinates** [Greiner 1989]. However, such an approach is often a bit clumsy and cumbersome, if possible at all, because the form of the constraints may not allow for expressing some parameters as proper functions of the others.

A much more elegant approach is based on the following nice insights: Let $\vec{\theta}^*$ be a (local) optimum of $F(\vec{\theta})$ *in the constrained subspace*. Then:

(1) The gradient $\nabla_{\vec{\theta}} F(\vec{\theta}^*)$, if it does not vanish, must be perpendicular to the constrained subspace. (If $\nabla_{\vec{\theta}} F(\vec{\theta}^*)$ had a component in the constrained subspace, $\vec{\theta}^*$ would not be a (local) optimum in this subspace.)

(2) The gradients $\nabla_{\vec{\theta}} C_j(\vec{\theta}^*)$, $1 \leq j \leq s$, must all be perpendicular to the constrained subspace. (They cannot have any component in the constrained subspace, because they are constant, namely 0, in this subspace.) Together these gradients span the subspace of the parameter space that is perpendicular to the constrained subspace.

(3) Therefore it must be possible to find values λ_j, $1 \leq j \leq s$, such that

$$\nabla_{\vec{\theta}} F(\vec{\theta}^*) + \sum_{j=1}^{s} \lambda_j \nabla_{\vec{\theta}} C_j(\vec{\theta}^*) = 0.$$

If the constraints (and thus their gradients) are linearly independent, the values λ_j are uniquely determined. This equation can be used to compensate the gradient of $F(\vec{\theta}^*)$, so that it vanishes at $\vec{\theta}^*$.

As a result of these insights, we obtain the method of so-called **Lagrange multipliers**. The basic idea underlying this method is to extend the functional $F(\vec{\theta})$ by terms $\lambda_j C_j(\vec{\theta})$, $1 \leq j \leq s$, to obtain a so-called **Lagrange function** $\mathcal{L}(\vec{\theta}, \vec{\lambda})$ with $\vec{\lambda} = \{\lambda_1, \ldots, \lambda_s\}$, where the λ_j are new unknowns, which have to be determined in the optimization process. That is, we get

$$\mathcal{L}(\vec{\theta}, \vec{\lambda}) = F(\vec{\theta}) + \sum_{k=1}^{s} \lambda_k C_k(\vec{\theta}),$$

which is to be optimized (usually to be minimized). Due to the representation of the gradient of $F(\vec{\theta}^*)$ found above, the gradient of the Lagrange function w.r.t. $\vec{\theta}$ must vanish at the local optimum $\vec{\theta}^*$, so that the standard optimization approach of setting the gradient equal to the null vector (necessary condition for an optimum) works again.

Note that with this we actually optimize the functional $F(\vec{\theta})$, because as long as we are inside the constrained subspace, the constraints are zero and thus do not have any influence on the value of the Lagrange function. Note also that computing the partial derivatives of $\mathcal{L}(\vec{\theta}, \vec{\lambda})$ w.r.t. the Lagrange multipliers λ_j reproduces the constraints, i.e.,

$$\forall j; 1 \leq j \leq s: \qquad \frac{\partial}{\partial \lambda_j} \mathcal{L}(\vec{\theta}, \vec{\lambda}) = C_j(\vec{\theta}),$$

as the λ_j appear neither in the functional $F(\vec{\theta})$ nor in the constraints $C_j(\vec{\theta})$ and are independent (provided the constraints are independent). In this way the constraints explicitely and naturally enter the optimization process.

The solution is then obtained in the usual way by solving the system of $r+s$ equations resulting from the necessary conditions for an optimum, i.e.,

$$\forall i; 1 \leq i \leq r: \ \frac{\partial}{\partial \theta_i} \mathcal{L}(\vec{\theta}, \vec{\lambda}) = 0 \quad \text{and} \quad \forall j; 1 \leq j \leq s: \ \frac{\partial}{\partial \lambda_j} \mathcal{L}(\vec{\theta}, \vec{\lambda}) = 0.$$

It should be noted that the method of Lagrange multipliers is more general than described here. For instance, it can also be applied if the constraints are not simple equations $C_j(\vec{\theta}) = 0$, $1 \leq j \leq s$, but are given in differential form. Details can be found, for instance, in [Greiner 1989].

An extension of Lagrange theory, the so-called **Kuhn–Tucker theory** [Kuhn and Tucker 1951], can handle even inequality constraints $C_j(\vec{\theta}) \leq 0$ of a certain type. Some explanations can be found, for example, in [Cristianini and Shawe-Taylor 2000]. However, since I use only equality constraints in this thesis, I do not discuss this extension here.

A.9 Heron's Algorithm

Heron's algorithm is a simple and efficient method for computing the square root $y = \sqrt{x}$ of a given real number x. It is based on the idea to rewrite the defining equation $y^2 = x$ as

$$y^2 = x \quad \Leftrightarrow \quad 2y^2 = y^2 + x \quad \Leftrightarrow \quad y = \frac{1}{2y}(y^2 + x) \quad \Leftrightarrow \quad y = \frac{1}{2}\left(y + \frac{x}{y}\right).$$

The resulting equation is then used as an iteration formula. That is, one computes the sequence

$$y_{k+1} = \frac{1}{2}\left(y_k + \frac{x}{y_k}\right) \qquad \text{for k} = 1, 2, \ldots \qquad \text{with} \qquad y_0 = 1.$$

(The initial value y_0 may be chosen differently, but this has only a minor effect on the number of steps needed until convergence.) It can be shown that $0 \leq y_k - \sqrt{x} \leq y_{k-1} - y_n$ for $k \geq 2$. Therefore this iteration formula provides increasingly better approximations of the square root of x and thus is a safe and simple way to compute it. For example, for $x = 2$ we get

$$y_0 = 1, \quad y_1 = 1.5, \quad y_2 \approx 1.41667, \quad y_3 \approx 1.414216, \quad y_4 \approx 1.414213 \quad \text{etc.}$$

As can already be seen from this sequence, Heron's algorithm converges very quickly and is often used in pocket calculators and microprocessors to implement the square root. In this thesis Heron's algorithm is used as an analogy to convey a core principle of the *expectation maximization algorithm* (cf. Section 5.2.4).

A.10 Types of Averages

The main types of averages of a set $\{x_1, \ldots, x_n\}$ of n numbers are the **arithmetic mean**

$$\bar{x}_{\text{arith}} = \frac{1}{n}\sum_{i=1}^{n} x_i,$$

the **quadratic mean**

$$\bar{x}_{\text{quad}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2},$$

the **geometric mean**

$$\bar{x}_{\text{geom}} = \sqrt[n]{\prod_{i=1}^{n} x_i},$$

and the **harmonic mean**

$$\bar{x}_{\text{harm}} = n \left(\sum_{i=1}^{n} x_i^{-1} \right)^{-1}.$$

The idea of the harmonic mean can be seen by writing its definition as

$$\frac{1}{n} \sum_{i=1}^{n} \frac{\bar{x}_{\text{harm}}}{x_i} = 1.$$

That is, the average ratio between the harmonic mean and the data values is 1. This average is used in Section 7.1 to compute the F_1 measure as the average of precision and recall.

Note that for positive x_i it is always

$$\min{}_{i=1}^{n} x_i \leq \bar{x}_{\text{harm}} \leq \bar{x}_{\text{geom}} \leq \bar{x}_{\text{arith}} \leq \bar{x}_{\text{quad}} \leq \max{}_{i=1}^{n} x_i.$$

A.11 The χ^2 Measure

The χ^2 measure is a test statistic that is used to assess the (in)dependence of two nominal variables forming a contingency table [Everitt 1998]. Under the hypothesis that the variables are independent it has, approximately, a χ^2 distribution. Formally, it is defined as follows:

Let A and B be two attributes with domains $\text{dom}(A) = \{a_1, \ldots a_r\}$ and $\text{dom}(B) = \{b_1, \ldots b_s\}$, respectively, and let $\mathbf{X}$ be a dataset over A and B. Let n_{ij}, $1 \leq i \leq r$, $1 \leq j \leq s$, be the number of sample cases in $\mathbf{X}$, which contain both the attribute values a_i and b_j. Furthermore, let

$$n_{i.} = \sum_{j=1}^{s} n_{ij}, \qquad n_{.j} = \sum_{i=1}^{r} n_{ij}, \qquad \text{and} \qquad n_{..} = \sum_{i=1}^{r} \sum_{j=1}^{s} n_{ij} = |\mathbf{X}|.$$

Finally, let

$$p_{i.} = \frac{n_{i.}}{n_{..}}, \qquad p_{.j} = \frac{n_{.j}}{n_{..}}, \qquad \text{and} \qquad p_{ij} = \frac{n_{ij}}{n_{..}}$$

	$B=0$	$B=1$	Σ
$A=0$	n_{00}	n_{01}	$n_{0.}$
$A=1$	n_{10}	n_{11}	$n_{1.}$
Σ	$n_{.0}$	$n_{.1}$	$n_{..}$

	$B=0$	$B=1$	Σ
$A=0$	p_{00}	p_{01}	$p_{0.}$
$A=1$	p_{10}	p_{11}	$p_{1.}$
Σ	$p_{.0}$	$p_{.1}$	1

Table A.1: A 2×2 contingency table for two binary attributes A and B and the joint probability distribution estimated from it.

be the probabilities of the attribute values and their combinations, as they can be estimated from these numbers. Then the well-known χ^2 measure is usually defined as [Everitt 1998]

$$\begin{aligned}
\chi^2(A,B) &= \sum_{i=1}^{r}\sum_{j=1}^{s}\frac{(E_{ij}-n_{ij})^2}{E_{ij}} \qquad \text{where} \quad E_{ij} = \tfrac{n_{i.}n_{.j}}{n_{..}} \\
&= \sum_{i=1}^{r}\sum_{j=1}^{s}\frac{\left(\frac{n_{i.}n_{.j}}{n_{..}} - n_{ij}\right)^2}{\frac{n_{i.}n_{.j}}{n_{..}}} = \sum_{i=1}^{r}\sum_{j=1}^{s}\frac{n_{..}^2\left(\frac{n_{i.}}{n_{..}}\frac{n_{.j}}{n_{..}} - \frac{n_{ij}}{n_{..}}\right)^2}{n_{..}\frac{n_{i.}}{n_{..}}\frac{n_{.j}}{n_{..}}} \\
&= n_{..}\sum_{i=1}^{r}\sum_{j=1}^{s}\frac{(p_{i.}\,p_{.j}-p_{ij})^2}{p_{i.}\,p_{.j}} = n_{..}\sum_{i=1}^{r}\sum_{j=1}^{s}\frac{(n_{i.}\,n_{.j}-n_{..}n_{ij})^2}{n_{i.}\,n_{.j}}.
\end{aligned}$$

This measure is often normalized by dividing it by the size $n_{..} = |\mathbf{X}|$ of the dataset to remove the dependence on the number of sample cases.

For binary attributes, that is, $\mathrm{dom}(A) = \mathrm{dom}(B) = \{0,1\}$, we obtain the 2×2 contingency table and the corresponding estimate of the joint probability distribution shown in Table A.1. Computing the (normalized) χ^2 measure from this table yields

$$\begin{aligned}
\frac{\chi^2(A,B)}{n_{..}} &= \sum_{i=0}^{1}\sum_{j=0}^{1}\frac{(p_{i.}\,p_{.j}-p_{ij})^2}{p_{i.}\,p_{.j}}. \\
&= \frac{(p_{0.}\,p_{.0}-p_{00})^2}{p_{0.}\,p_{.0}} + \frac{(p_{0.}\,p_{.1}-p_{01})^2}{p_{0.}\,p_{.1}} \\
&+ \frac{(p_{1.}\,p_{.0}-p_{10})^2}{p_{1.}\,p_{.0}} + \frac{(p_{1.}\,p_{.1}-p_{11})^2}{p_{1.}\,p_{.1}}
\end{aligned}$$

Now we can exploit the obvious relations

$$p_{00}+p_{01}=p_{0.}, \quad p_{10}+p_{10}=p_{1.}, \quad p_{00}+p_{10}=p_{.0}, \quad p_{01}+p_{11}=p_{.1},$$
$$p_{0.}+p_{1.}=1, \qquad p_{.0}+p_{.1}=1,$$

which leads to

$$\begin{aligned} p_{0\cdot}\, p_{\cdot 0} - p_{00} &= (1-p_{1\cdot})(1-p_{\cdot 1}) - (1-p_{1\cdot}-p_{\cdot 1}+p_{11}) &&= p_{1\cdot}\, p_{\cdot 1} - p_{11}, \\ p_{0\cdot}\, p_{\cdot 1} - p_{01} &= (1-p_{1\cdot})p_{\cdot 1} - (p_{\cdot 1} - p_{11}) &&= p_{11} - p_{1\cdot}\, p_{\cdot 1}, \\ p_{1\cdot}\, p_{\cdot 0} - p_{10} &= p_{1\cdot}(1-p_{\cdot 1}) - (p_{1\cdot} - p_{11}) &&= p_{11} - p_{1\cdot}\, p_{\cdot 1}. \end{aligned}$$

Therefore it is

$$\begin{aligned} \frac{\chi^2(A,B)}{n_{\cdot\cdot}} &= \frac{(p_{1\cdot}\, p_{\cdot 1} - p_{11})^2}{(1-p_{1\cdot})(1-p_{\cdot 1})} + \frac{(p_{1\cdot}\, p_{\cdot 1} - p_{11})^2}{(1-p_{1\cdot})\, p_{\cdot 1}} \\ &\quad + \frac{(p_{1\cdot}\, p_{\cdot 1} - p_{11})^2}{p_{1\cdot}(1-p_{\cdot 1})} + \frac{(p_{1\cdot}\, p_{\cdot 1} - p_{11})^2}{p_{1\cdot}\, p_{\cdot 1}} \\ &= \frac{(p_{1\cdot}\, p_{\cdot 1} - p_{11})^2}{p_{1\cdot}(1-p_{1\cdot})p_{\cdot 1}(1-p_{\cdot 1})} = \frac{(p_{1\cdot}\, p_{\cdot 1} - p_{11})^2}{p_{1\cdot}p_{0\cdot}p_{\cdot 1}p_{\cdot 0}}. \end{aligned}$$

In an analogous way the (normalized) χ^2 measure can also be computed from the absolute frequencies n_{ij}, $n_{i\cdot}$, $n_{\cdot j}$ and $n_{\cdot\cdot}$, namely as

$$\frac{\chi^2(A,B)}{n_{\cdot\cdot}} = \frac{(n_{1\cdot}n_{\cdot 1} - n_{\cdot\cdot}n_{11})^2}{n_{1\cdot}(n_{\cdot\cdot} - n_{1\cdot})n_{\cdot 1}(n_{\cdot\cdot} - n_{\cdot 1})} = \frac{(n_{1\cdot}n_{\cdot 1} - n_{\cdot\cdot}n_{11})^2}{n_{1\cdot}n_{0\cdot}n_{\cdot 1}n_{\cdot 0}}.$$

Appendix B

List of Symbols

This appendix contains, as a quick reference, a list of the symbols and notational conventions I use in this thesis. In order to ease the lookup, the symbols are in alphabetical order, with the Greek letters placed in such a way that they are following phonetically similar Latin letters.

∇ A differential operator (read "nabla") that combines partial derivatives into a vector-like mathematical object, $\nabla_{\vec{x}} = \left(\frac{\partial}{\partial x_1}, \dots \frac{\partial}{\partial x_m}\right)$, and analogously for matrices. For scalar quantities equivalent to a partial derivative: $\nabla_x = \frac{\partial}{\partial x}$.

$\lfloor x \rfloor$ Largest integer number less than or equal to x.

$\lceil x \rceil$ Smallest integer number greater than or equal to x.

$\vec{0}$ A null vector, usually m-dimensional, $\vec{0} = (0, \dots, 0)$.

$\mathbf{0}$ A null matrix, usually of size $m \times m$.

$\mathbf{1}$ A unit matrix, usually of size $m \times m$.

$\mathbf{A}$ A matrix (in general descriptions), $\mathbf{A} = (a_{ij})$.

$\mathbf{A}^{-1}$ The inverse of the matrix $\mathbf{A}$.

$\mathbf{A}^{\top}$ The transpose of the matrix $\mathbf{A}$.

$\mathbf{A}^{*}$ The cofactor matrix of a matrix $\mathbf{A}$, $\mathbf{A}^{*} = (\mathbf{a}^{*}_{ij})$.

$|\mathbf{A}|$ The determinant of the matrix $\mathbf{A}$.

A An attribute used to describe an object or case.

a A parameter of a radial function: the exponent of the distance. With indices used for the elements of a matrix $\mathbf{A}$.

$\mathbf{a}$ The minors of a matrix $\mathbf{A}$, $\mathbf{a}_{ij} = |\mathbf{A}_{ij}|$, where $\mathbf{A}_{ij}$ results from the matrix $\mathbf{A}$ by removing the i-th row and the j-th column.

$\mathbf{a}^*$ The cofactors of a matrix $\mathbf{A}$, $\mathbf{a}^*_{ij} = (-1)^{i+j}\mathbf{a}_{ij}$, where $\mathbf{a}_{ij}$ is a minor of the matrix $\mathbf{A}$.

α A parameter of the membership transformation: the exponent of the (weighted) membership degree.

B An attribute, in particular, a binary attribute.

b A parameter of a radial function: an offset to be added to the distance (after the distance has been raised to the power a).

β A parameter of the membership transformation: the fraction of the maximal membership degree, above which membership degrees to other clusters are not annulled.

$\mathbf{C}$ The set of cluster parameters, $\mathbf{C} = \{\mathbf{c}_i \mid 1 \leq i \leq c\}$.

$\mathcal{C}$ The set of all possible sets of cluster parameters, usually restricted to a certain number c of cluster.

C A(n equality) constraint in Lagrange theory.

$\mathbf{c}$ The parameters of a cluster, $\mathbf{c} = (\vec{\mu}, \mathbf{\Sigma}, \varrho)$, consisting of a location (or center) $\vec{\mu}$, a covariance matrix $\mathbf{\Sigma}$, and a weight ϱ.

c The number of clusters, $c = |\mathbf{C}|$.

$\check{c}$ The number of clusters to which a data point has a positive degree of membership.

D The length of the diagonal of the data space, with an index a document in a document collection.

d A distance measure, most of the time either the Euclidean distance $d(\vec{x}, \vec{y}; \mathbf{1})$ or the Mahalanobis distance $d(\vec{x}, \vec{y}; \mathbf{\Sigma})$.

diag An operator that constructs a diagonal matrix, i.e., a matrix that has non-vanishing elements only in the diagonal. The values of these elements are given as parameters.

∂ A symbol denoting a partial derivative, $\frac{\partial}{\partial x}$ means the partial derivative w.r.t. x.

Δ	Generally a difference, in particular the (one-dimensional) distance between two cluster centers.
δ	The so-called Kronecker symbol. It is $\delta_{x,y} = 1$ if $x = y$ and $\delta_{x,y} = 0$ otherwise.
E	The expected value of a random variable.
e	An objective function based on the sum of (squared) errors, also the base of the exponential function, $e \approx 2.718282$.
e_{sqr}	The sum of squared errors.
e_{abs}	The sum of absolute errors.
η	A learning rate (or step width parameter) in a gradient method.
F	A functional that is to be optimized.
f	A function, most of the time a radial function that describes the degree of membership to a cluster or a probability density function (in the latter case indices state the random variables referred to).
f_{Cauchy}	The (generalized) Cauchy function, $f_{\text{Cauchy}}(r; a, b) = \frac{1}{r^a + b}$.
f_{Gauss}	The (generalized) Gaussian function, $f_{\text{Gauss}}(r; a, b) = e^{-\frac{1}{2} r^a}$.
ϕ	A rotation angle (in eigenvalue decomposition).
g	A (usually linear) function that yields the degree of membership to a class (not to a cluster!).
$\vec{g}$	A vector combining the values of the (linear) classification functions of the different classes, $\vec{g}(\vec{x}) = (g_1(\vec{x}), \ldots, g_s(\vec{x})$.
Γ	The Gamma function, also known as the generalized factorial.
γ	The normalization factor for a radial function that is based on a Mahalanobis distance, $\gamma(a, b, m, \mathbf{\Sigma}_i)$, which leads to the radial function having integral 1 over the data space.
γ^*	The normalization factor for a radial function that is based on a Euclidean distance, $\gamma^*(a, b, m) = \gamma(a, b, m, \mathbf{1})$.
h	Transformation function for the membership degrees in fuzzy clustering. Most of the time it is $h(u) = u^w$, $w \in (1, \infty)$. Also a regularization parameter for shape regularization.
I	A set of indices, $I \subseteq \mathbb{N}$.

i	An index variable; used mainly for clusters.
j	An index variable; used mainly for data points.
J	An objective function based on the sum of (squared) distances.
K	A kernel function for kernel density estimation.
k	An index variable; used mainly for classes and dimensions.
κ	A parameter, used for different purposes. In particular, κ is the exponent of the isotropic cluster radius that specifies how the size of a cluster is measured.
$\mathbf{L}$	A lower (or left) triangular matrix, $\mathbf{L} = (l_{ij})$, as it appears in the Cholesky decomposition of a covariance matrix $\mathbf{\Sigma}$.
L	A likelihood function.
$\mathcal{L}$	A Lagrange function.
l	An index variable, used for different purposes. With indices an element of a lower (or left) triangular matrix.
λ	An eigenvalue of a covariance matrix $\mathbf{\Sigma}$.
m	The number of dimensions of the data space, which is $\mathbb{R}^m$.
$\vec{\mu}$	The location parameter of a cluster, $\vec{\mu} = (\mu_1, \ldots, \mu_m)$.
N	A number of data point pairs with a certain property (which is specified by indices).
n	The number of data points in the given data set, $n = \lvert\mathbf{X}\rvert$. With indices the number of data points with a certain property (for example, the number of true positives).
ν	A cluster-specific parameter of possibilistic fuzzy clustering, also used for other parameters in different contexts.
o	The class predicted by a classifier (i.e., its output), also a regularization parameter for size and weight regularization.
$\vec{o}$	An offset vector (for a translation).
P	A probability measure.
p	A probability distribution (indices state random variables).
Π	The set of all permutations of a set of objects.

π	The circle measure, $\pi \approx 3.14159265$. Also used to denote the precision of a classifier.
$\mathbf{\Psi}$	A coincidence matrix for comparing two partitions, $\mathbf{\Psi} = (\psi_{ij})$.
ψ	An element of a coincidence matrix $\mathbf{\Psi}$.
Q	A quality measure for assessing clustering results (like the Dunn index or the Xie-Beni index) or for evaluating keywords w.r.t. their usefulness to classify a set of documents (like information gain or the χ^2 measure).
$\mathbf{R}$	A rotation matrix (for example, in eigenvalue decomposition).
r	A radius, i.e., the distance from a cluster center. Often appearing as the argument of a radial function.
ϱ	The weight of a cluster.
ρ	The recall of a classifier w.r.t. a class.
S	A measure for the size of or the scatter within a cluster in cluster assessment indices.
$\mathbf{S}$	A symmetric and positive definite matrix with determinant 1 that captures the shape parameters of a covariance matrix $\mathbf{\Sigma}$.
s	The number of classes. Also used to denote a scaling factor.
$\mathbf{\Sigma}$	A covariance matrix, the shape and size parameter of a cluster, $\mathbf{\Sigma} = (\sigma_{ij})_{1 \leq i,j \leq m}$.
$\mathbf{\Sigma}^{-1}$	The inverse of a covariance matrix $\mathbf{\Sigma}$.
σ^2	A variance, either as an element of a covariance matrix or as an equivalent isotropic variance.
σ	An isotropic standard deviation or (with a double index for row and column) an element of a covariance matrix $\mathbf{\Sigma}$.
ς	An index mapping function used to sort membership degrees. Also used to denote an arbitrary permutation of a set of indices.
$\mathbf{T}$	A transformation matrix, usually the result of an eigenvalue decomposition of a covariance matrix $\mathbf{\Sigma}$.
T	The total number of iterations in an iterative update scheme.
t	An argument of a function or an independent variable, also a time step or series element index; used for different purposes.

θ A parameter of a model (in general descriptions).

$\vec{\theta}$ A vector of parameters of a model (in general descriptions).

$\top$ A symbol indicating that a matrix or a vector is to be transposed.

$\mathbf{U}$ A (fuzzy) partition matrix, $\mathbf{U} = (u_{ij})_{1 \leq i \leq c, 1 \leq j \leq n}$, which comprises the assignments of the data points $\vec{x}_j \in \mathbf{X}$ to the clusters $\mathbf{c}_i \in \mathbf{C}$. Also the left matrix in singular value decomposition.

U A sum of membership degrees, needed as a normalization factor.

u° The raw membership function.

u^* The weighted membership function.

$u^\bullet$ The weighted and transformed membership function.

u The normalized membership function; with indices an element of a (fuzzy) partition matrix.

$\mathbf{V}$ The right matrix in singular value decomposition.

$\vec{v}$ An eigenvector of a covariance matrix $\mathbf{\Sigma}$, $\vec{v} = (v_1, \ldots, v_m)$, sometimes also used for other vectors.

W A word in a document.

$\mathbf{W}$ A weight matrix combining the weight vectors of the (linear) classification functions for all classes, $\mathbf{W} = (w_{ki})_{1 \leq k \leq s, 0 \leq i \leq c}$. Also the diagonal matrix in singular value decomposition.

w Without indices the fuzzifier or weighting exponent in fuzzy clustering or the window width in kernel estimation; with a single index a singular value; with two indices an element of a weight matrix.

$\vec{w}$ A weight vector in a (linear) classification function, as it is used, for instance, in a radial basis function network for the weights of the connections from the hidden layer to the output layer. Also the parameters in a multivariate (linear) regression.

ω A data point weight in fuzzy clustering with outliers.

$\mathbf{X}$ The given data set, which may be seen either as a set of vectors $\mathbf{X} = \{\vec{x}_j \mid 1 \leq j \leq n\}$ or as a data matrix $\mathbf{X} = (x_{jk})_{1 \leq j \leq n, 1 \leq k \leq m}$. It is always $\mathbf{X} \subset \mathbb{R}^m$ (with $\mathbf{X}$ interpreted as a set of vectors).

$\mathcal{X}$ A random variable that has data sets as possible values.

$\vec{X}$ A random vector that has the data space, i.e. $\mathbb{R}^m$, as its domain.

X	An element of a random vector $\vec{X}$, also used for a binary random variable used to describe the occurrence of a keyword in a document.
$\vec{x}$	A data point, $\vec{x} = (x_1, \ldots, x_m)^\top \in \mathbb{R}^m$.
$\vec{x}^\top$	The transpose of the vector $\vec{x}$.
ξ	A parameter, used for different purposes.
Y	A random variable that has cluster indices as possible values, $\mathrm{dom}(Y) = \{1, \ldots, c\}$.
y	A cluster index associated with a data point $\vec{x}$, $y \in \{1, \ldots, c\}$.
$\vec{y}$	Usually a vector $\vec{y} = (y_1, \ldots, y_n)$ that states the clusters associated with the data points in the given data set $\mathbf{X}$. Also used as a point in the data space in Section 2.1 on distance measures.
Z	A random variable that has indices of classes as possible values, $\mathrm{dom}(Z) = \{1, \ldots, s\}$.
z	A class index associated with a data point $\vec{x}$, $z \in \{1, \ldots, s\}$.
$\vec{z}$	Usually a vector $\vec{z} = (z_1, \ldots, z_n)$ that states the classes associated with the data points in the given data set $\mathbf{X}$. Also used as a point in the data space in Section 2.1 on distance measures.
ζ	A parameter, used for different purposes. With an index the class associated with a cluster or a reference vector.

When writing functions, in particular distance measure or probability density functions, I distinguish between the **arguments** of a function and its **parameters**. The arguments specify the point at which the function is to be evaluated, the parameters select a particular function from a parameterized family of functions of the same type. In my notation I distinguish arguments and parameters by separating them with a semicolon: the arguments of the function come first and then, after a semicolon, its parameters are listed. For example, in the distance $d(\vec{x}, \vec{y}; \mathbf{\Sigma})$ the vectors $\vec{x}$ and $\vec{y}$ are the arguments, that is, the points the distance of which is to be determined. The distance measure is parameterized by a covariance matrix $\mathbf{\Sigma}$, which specifies how the distance computation is to be carried out. Another example is the probability density function $f_{\vec{X}}(\vec{x}; \Theta)$. Here $\vec{x}$, the value of the random vector $\vec{X}$, is the argument and Θ is the (set of) parameters. This example also shows why it is inconvenient to write the parameters as indices: the place of the index is already taken by the random variable.

Bibliography

[Abraham *et al.* 2002] A. Abraham, J. Ruiz-del-Solar, and M. Köppen, eds. *Soft Computing Systems: Design, Management and Applications*. IOS Press, Amsterdam, The Netherlands 2002

[Acciani *et al.* 1999] G. Acciani, E. Chiarantoni, G. Fornarelli, and S. Vergura. A Feature Extraction Unsupervised Neural Network for an Environmental Data Set. *Neural Networks* 16(3–4):427–436. Elsevier Science, Amsterdam, Netherlands 1999

[Aha 1992] D.W. Aha. Tolerating Noisy, Irrelevant and Novel Attributes in Instance-based Learning Algorithms. *Int. Journal of Man-Machine Studies* 36(2):267–287. Academic Press, San Diego, CA, USA 1992

[Akaike 1974] H. Akaike. A New Look at the Statistical Model Identification. *IEEE Trans. on Automatic Control* 19:716–723. IEEE Press, Piscataway, NJ, USA 1974

[Albert 1972] A. Albert. *Regression and the Moore-Penrose Pseudoinverse*. Academic Press, New York, NY, USA 1972

[Anderson 1935] E. Anderson. The Irises of the Gaspe Peninsula. *Bulletin of the American Iris Society* 59:2–5. American Iris Society, Philadelphia, PA, USA 1935

[Anderson 1995] J.A. Anderson. *An Introduction to Neural Networks*. MIT Press, Cambridge, MA, USA 1995

[Aurenhammer 1991] F. Aurenhammer. Voronoi Diagrams — A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys* 23(3):345–405. ACM Press, New York, NY, USA 1991

[Bachelor and Wilkins 1969] B.G. Bachelor and B.R. Wilkins. Method for Location of Clusters of Patterns to Initiate a Learning Machine. *Elec-*

tronics Letters 5:481–483. Institution of Electrical Engineers, London, United Kingdom 1969

[Babu and Murty 1994] G.P. Babu and M.N. Murty. Clustering with Evolutionary Strategies. *Pattern Recognition* 27:321–329. Pergamon Press, Oxford, United Kingdom 1994

[Bäck *et al.* 1991] T. Bäck, F. Hoffmeister, and H. Schwefel. A Survey of Evolution Strategies. *Proc. 4th Int. Conf. on Genetic Algorithms (ICGA'91, San Diego, CA)*, 2–9. Morgan Kaufmann, San Mateo, CA 1991

[Backer and Jain 1981] E. Backer and A.K. Jain. A Clustering Performance Measure based on Fuzzy Set Decomposition. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)* 3(1):66–74. IEEE Press, Piscataway, NJ, USA 1981

[Baeza-Yates and Ribeiro-Neto 1999] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, Reading, MA, USA 1999

[Ball and Hall 1966] G.H. Ball and D.J. Hall. ISODATA — An Iterative Method of Multivariate Data Analysis and Pattern Classification. *IEEE Int. Comm. Conf. (Philadelphia, PA)*. IEEE Press, Piscataway, NJ, USA 1966

[Ball and Hall 1967] G.H. Ball and D.J. Hall. A Clustering Technique for Summarizing Multivariate Data. *Behavioral Science* 12(2):153–155. 1967

[Barni *et al.* 1996] M. Barni, V. Cappellini, and A. Mecocci. Comments on "A Possibilistic Approach to Clustering". *IEEE Trans. on Fuzzy Systems* 4(3):393–396. IEEE Press, Piscataway, NJ, USA 1996

[Bauer *et al.* 1997] E. Bauer, D. Koller, and Y. Singer. Update Rules for Parameter Estimation in Bayesian Networks. *Proc. 13th Conf. on Uncertainty in Artificial Intelligence (UAI'97, Providence, RI, USA)*, 3–13. Morgan Kaufmann, San Mateo, CA, USA 1997

[Berry 2003] M.W. Berry, ed. *Survey of Text Mining*. Springer-Verlag, New York, NY, USA 2003

[Berry and Linoff 2004] M.J.A. Berry and G. Linoff. *Data Mining Techniques (2nd edition)*. J. Wiley & Sons, Chichester, England 2004 (1st edition 1997)

[Berthold and Diamond 1995] M.R. Berthold and J. Diamond. Boosting the Performance of RBF Networks with Dynamic Decay Adjustment. In: [Tesauro *et al.* 1995], 512–528.

[Berthold and Diamond 1998] M.R. Berthold and J. Diamond. Constructive Training of Probabilistic Neural Networks. *Neurocomputing* 19:167–183. Elsevier Science, Amsterdam, Netherlands 1998

[Berthold and Huber 1999] M.R. Berthold and K.-P. Klaus-Peter. Constructing Fuzzy Graphs from Examples. *Intelligent Data Analysis* 3(1):37–54. IOS Press, Amsterdam, Netherlands 1999

[Berthold and Hand 1999] M.R. Berthold and D.J. Hand. *Intelligent Data Analysis*. Springer-Verlag, Berlin, Germany 1999

[Bezdek 1973] J.C. Bezdek. Cluster Validity with Fuzzy Sets. *Journal of Cybernetics* 3(3):58–73. Hemisphere, Washington, DC, USA 1973

[Bezdek 1975] J.C. Bezdek. Mathematical Models for Systematics and Taxanomy. *Proc. 8th Int. Conf. on Numerical Taxanomy*, 143–166. Freeman, San Francisco, CA 1975

[Bezdek 1980] J.C. Bezdek. A Convergence Theorem for the Fuzzy ISODATA Clustering Algorithm. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)* 2(1):1–8. IEEE Press, Piscataway, NJ, USA 1980. Reprinted in [Bezdek and Pal 1992], 130–137

[Bezdek 1981] J.C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, NY, USA 1981

[Bezdek *et al.* 1987] J.C. Bezdek, R.J. Hathaway, M.J. Sabin, and W.T. Tucker. Convergence Theory for Fuzzy c-Means: Counterexamples and Repairs. *IEEE Trans. on Systems, Man, and Cybernetics (Part B: Cybernetics)* 17(5):873–877. IEEE Press, Piscataway, NJ, USA 1987. Reprinted in [Bezdek and Pal 1992], 138–142

[Bezdek and Pal 1992] J.C. Bezdek and N. Pal. *Fuzzy Models for Pattern Recognition*. IEEE Press, New York, NY, USA 1992

[Bezdek *et al.* 1994] J.C. Bezdek, S. Boggavarapu, L.O. Hall, and A. Bensaid. Genetic Algorithm Guided Clustering. *Proc. 1st IEEE Conf. on Evolutionary Computation (ICEC'94, Orlando, FL)*, 34–39. IEEE Press, Piscataway, NJ, USA 1994

[Bezdek and Pal 1995] J.C. Bezdek and N. Pal. Two Soft Relatives of Learning Vector Quantization. *Neural Networks* 8(5):729–764. Prentice Hall, Upper Saddle River, NJ, USA 1995

[Bezdek *et al.* 1997] J.C. Bezdek, W.Q. Li, Y. Attikiouzel, and M. Windham. A Geometric Approach to Cluster Validity for Normal Mixtures. *Soft Computing* 1(4):166–179. Springer-Verlag, Heidelberg, Germany 1997

[Bezdek and Pal 1998] J.C. Bezdek and N. Pal. Some New Indices for Cluster Validity. *IEEE Trans. on Systems, Man, and Cybernetics (Part B: Cybernetics)* 28(2):301–315. IEEE Press, Piscataway, NJ, USA 1998

[Bezdek *et al.* 1999] J.C. Bezdek, J. Keller, R. Krishnapuram, and N. Pal. *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Kluwer, Dordrecht, Netherlands 1999

[Bilmes 1997] J. Bilmes. A Gentle Tutorial on the EM Algorithm and Its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. *Tech. Report ICSI-TR-97-021.* University of Berkeley, CA, USA 1997

[Bishop 1995] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, United Kingdom 1995

[Blake and Merz 1998] C.L. Blake and C.J. Merz. *UCI Repository of Machine Learning Databases*. University of California, Irvine, CA, USA 1998
`http://www.ics.uci.edu/~mlearn/MLRepository.html`

[Bobrowski and Bezdek 1991] L. Bobrowski and J.C. Bezdek. c-Means Clustering with the L_1 and L_∞ Norms. *IEEE Trans. on Systems, Man and Cybernetics* 21(3):545–554. IEEE Press, Piscataway, NJ, USA 1991

[Bock 1974] H.H. Bock. *Automatische Klassifikation (Cluster-Analyse).* Vandenhoek & Ruprecht, Göttingen, Germany 1974

[Böhme 1994] G. Böhme. *Fuzzy-Logik*. Springer-Verlag, Berlin, Germany 1993

[Borgelt and Kruse 2002] C. Borgelt and R. Kruse. *Graphical Models — Methods for Data Analysis and Mining*. J. Wiley & Sons, Chichester, United Kingdom 2002

[Borgelt and Kruse 2003] C. Borgelt and R. Kruse. Speeding Up Fuzzy Clustering with Neural Network Techniques. *Proc. 12th IEEE Int. Conference on Fuzzy Systems (FUZZ-IEEE'03, St. Louis, MO, USA)*, on CDROM. IEEE Press, Piscataway, NJ, USA 2003

[Borgelt and Kruse 2004] C. Borgelt and R. Kruse. Shape and Size Regularization in Expectation Maximization and Fuzzy Clustering. *Proc.*

8th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD 2004, Pisa, Italy), 52–62. Springer-Verlag, Berlin, Germany 2004

[Borgelt and Kruse 2005] C. Borgelt and R. Kruse. Fuzzy and Probabilistic Clustering with Shape and Size Constraints. *Proc. 15th Int. Fuzzy Systems Association World Congress (IFSA'05, Beijing, China)*, to appear. 2005

[Borgelt and Nürnberger 2004a] C. Borgelt and A. Nürnberger. Fast Fuzzy Clustering of Web Page Collections. *Proc. PKDD Workshop on Statistical Approaches for Web Mining (SAWM 2004)*, 75–86. University of Pisa, Pisa, Italy 2004

[Borgelt and Nürnberger 2004b] C. Borgelt and A. Nürnberger. Experiments in Document Clustering using Cluster Specific Term Weights. *Proc. Workshop Machine Learning and Interaction for Text-based Information Retrieval (TIR 2004, Ulm, Germany)*, 55–68. University of Ulm, Ulm, Germany, 2004

[Borgelt and Nürnberger 2004c] C. Borgelt and A. Nürnberger. Experiments in Term Weighting and Keyword Extraction in Document Clustering. *Lernen, Wissensentdeckung und Adaptivität, Workshop GI Fachgruppe Maschinelles Lernen (LWA 2004/FGML 2004, Berlin, Germany)*, 123–130. Humboldt University, Berlin, Germany 2004

[Borgelt *et al.* 2004] C. Borgelt, D. Girimonte, and G. Acciani. Learning Vector Quantization: Cluster Size and Cluster Number. *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS 2004, Vancouver, Canada)*. IEEE Press, Piscataway, NJ, USA, 2004

[Borgelt *et al.* 2005] C. Borgelt, A. Nürnberger, and R. Kruse. Fuzzy Learning Vector Quantization with Size and Shape Parameters. *Proc. 14th IEEE Int. Conference on Fuzzy Systems (FUZZ-IEEE'05, Reno, NV, USA)*, to appear. IEEE Press, Piscataway, NJ, USA 2005

[Borgelt *et al.* 2001] C. Borgelt, H. Timm, and R. Kruse. Probabilistic Networks and Fuzzy Clustering as Generalizations of Naive Bayes Classifiers. In: [Reusch and Temme 2001], 121–138.

[Breckenridge 1989] J. Breckenridge. Replicating Cluster Analysis: Method, Consistency and Validity. *Multivariate Behavioral Research* 24:147–161. Lawrence Erlbaum Associates, Mahwah, NJ, USA 1989

[Breiman *et al.* 1984] L. Breiman, J.H. Friedman, R.A. Olsen, and C.J. Stone. *Classification and Regression Trees.* Wadsworth International Group, Belmont, CA, USA 1984

[Bronstein *et al.* 1995] I.N. Bronstein, K.A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik (2. edition).* Verlag Harri Deutsch, Thun/Frankfurt am Main, Germany 1995

[Bronstein *et al.* 1996] I.N. Bronstein, K.A. Semendjajew, G. Grosche, V. Ziegler, and D. Ziegler. *Teubner Taschenbuch der Mathematik*, Part 1. Teubner, Leipzig, Germany 1996

[Clark and Niblett 1989] P. Clark and T. Niblett. The CN2 Induction Algorithm. *Machine Learning* 3(4):261-283. Kluwer, Dordrecht, Netherlands 1989

[Clark and Boswell 1991] P. Clark and R.A. Boswell. Rule Induction with CN2: Some Recent Improvements. *Proc. 5th European Working Session on Learning (EWSL-91, Porto, Portugal)*, 151–163. Springer-Verlag, Berlin, Germany 1991

[Cheeseman and Stutz 1996] P. Cheeseman and J. Stutz. Bayesian Classification (AutoClass): Theory and Results. In: [Fayyad *et al.* 1996a], 153–180.

[Chen *et al.* 2004] S. Chen, X. Hong, C.J. Harris, and P.M. Sharkey. Sparse Modelling Using Orthogonal Forward Regression with PRESS Statistic and Regularization. *IEEE Trans. on Systems, Man and Cybernetics (Part B)* 34(4):898-911. IEEE Press, Piscataway, NJ, USA 2004

[Chiu 1994] S. Chiu. Fuzzy Model Identification Based on Cluster Estimation. *Journal of Intelligent & Fuzzy Systems* 2(3):267–278. IEEE Press, Piscataway, NJ, USA 1994

[Cristianini and Shawe-Taylor 2000] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, Cambridge, United Kingdom 2000

[Chu *et al.* 2001] S.C. Chu, J.F. Roddick, and J.S. Pan. A Comparative Study and Extensions to k-Medoids Algorithms. *Proc. 5th Int. Conf. on Optimization: Techniques and Applications (ICOTA'01, Hong Kong, China)*, 1708–1717. World Scientific, Hackensack, NJ, USA 2001

[Cohen 1995] W.W. Cohen. Fast Effective Rule Induction. *Proc. 12th Int. Conf. on Machine Learning (ICML 95, Lake Tahoe, CA)*, 115–123. Morgan Kaufmann, San Mateo, CA, USA 1995

[Darwin 1859] C. Darwin. *The Origin of Species*. Penguin, London, United Kongdom 1980 (first published in 1859)

[Dasarathy 1990] B.V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classifcation Techniques*. IEEE Computer Science Press, Los Alamitos, CA, USA 1990

[Davé 1991] R.N. Davé. Characterization and Detection of Noise in Clustering. *Pattern Recognition Letters* 12:657–664. Elsevier Science, Amsterdam, Netherlands 1991

[Davé and Krishnapuram 1997] R.N. Davé and R. Krishnapuram. Robust Clustering Methods: A Unified View. *IEEE Trans. on Fuzzy Systems* 5:270–293. IEEE Press, Piscataway, NJ, USA 1997

[Davé and Sen 1997] R.N. Dav and S. Sen. On Generalizing the Noise Clustering Algorithms. *Proc. 7th Int. Fuzzy Systems Association World Congress (IFSA'97)*, 3:205–210. Academia, Prague, Czech Republic 1997

[Davies and Bouldin 1979] D.L. Davies and D.W. Bouldin. A Cluster Separation Measure. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)* 1(4):224–227. IEEE Press, Piscataway, NJ, USA 1979

[Dawkins 1976] R. Dawkins. *The Selfish Gene*. Oxford University Press, Oxford, United Kingdom, 1976

[Dawkins 1987] R. Dawkins. *The Blind Watchmaker*. W.W. Norton & Co., London, United Kingdom, 1987

[Deerwester *et al.* 1990] S. Deerwester, S.T. Dumais, G.W. Furnas, and T.K. Landauer. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Sciences* 41:391–407. J. Wiley & Sons, New York, NY, USA 1990

[Dempster *et al.* 1977] A.P. Dempster, N. Laird, and D. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society (Series B)* 39:1–38. Blackwell, Oxford, United Kingdom 1977

[DeSieno 1988] D. DeSieno. Adding a Conscience to Competitive Learning. *IEEE Int. Conf. on Neural Networks*, Vol. I, 117–124. IEEE Press, Piscataway, NJ, USA 1988

[Döring *et al.* 2004] C. Döring, C. Borgelt, and R. Kruse. Fuzzy Clustering of Quantitative and Qualitative Data. *Proc. 23rd Conf. North American Fuzzy Information Processing Society (NAFIPS 2004, Banff, Alberta, Canada)*, 84–89. IEEE Press, Piscataway, NJ, USA 2004

[Döring *et al.* 2005] C. Döring, C. Borgelt, and R. Kruse. Effects of Irrelevant Attributes in Fuzzy Clustering. *Proc. 14th IEEE Int. Conference on Fuzzy Systems (FUZZ-IEEE'05, Reno, NV, USA)*, on CDROM. IEEE Press, Piscataway, NJ, USA 2005

[Domingos 1996] P. Domingos. Efficient Specific-to-general Rule Induction. *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD 96, Portland, Oregon)*, 319–322. AAAI Press, Menlo Park, CA, USA 1996

[Drineas *et al.* 2004] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering Large Graphs via the Singular Value Decomposition. *Machine Learning* 56:9–33. Kluwer, Dordrecht, Netherlands 2004

[Dubois and Prade 1980] D. Dubois and H. Prade. *Fuzzy Sets and Systems — Theory and Applications*. Academic Press, San Diego, CA, USA 1980

[Dubois and Prade 1988] D. Dubois and H. Prade. *Possibility Theory*. Plenum Press, New York, NY, USA 1988

[Duda and Hart 1973] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. J. Wiley & Sons, New York, NY, USA 1973

[Dunn 1973] J.C. Dunn. A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. *Journal of Cybernetics* 3(3):32–57. American Society for Cybernetics, Washington, DC, USA 1973 Reprinted in [Bezdek and Pal 1992], 82–101

[Efron and Tibshirani 1993] B. Efron and R.J. Tibshirani. An Introduction to the Boostrap. Chapman & Hall, London, United Kingdom 2003

[Engl *et al.* 1996] H. Engl, M. Hanke, and A. Neubauer. *Regularization of Inverse Problems*. Kluwer, Dordrecht, Netherlands 1996

[Everitt 1981] B.S. Everitt. *Cluster Analysis*. Heinemann, London, United Kingdom 1981

[Everitt 1998] B.S. Everitt. *The Cambridge Dictionary of Statistics*. Cambridge University Press, Cambridge, United Kingdom 1998

[Everitt and Hand 1981] B.S. Everitt and D.J. Hand. *Finite Mixture Distributions*. Chapman & Hall, London, United Kingdom 1981

[Fahlman 1988] S.E. Fahlman. An Empirical Study of Learning Speed in Backpropagation Networks. In: [Touretzky *et al.* 1988].

[Fayyad *et al.* 1996a] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, eds. *Advances in Knowledge Discovery and Data Min-*

ing. AAAI Press and MIT Press, Menlo Park and Cambridge, MA, USA 1996

[Fayyad *et al.* 1998] U.M. Fayyad, C.A. Reina, and P.S. Bradley. Initialization of Iterative Refinement Clustering Algorithms. *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining (KDD'98, New York, NY)*, 194–198. AAAI Press, Menlo Park, CA, USA 1998

[Fisher 1925] R.A. Fisher. Theory of Statistical Estimation. *Proc. Cambridge Philosophical Society* 22:700–725. Cambridge University Press, Cambridge, United Kingdom 1925

[Fisher 1936] R.A. Fisher. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics* 7(2):179–188. Cambridge University Press, Cambridge, United Kingdom 1936

[Foreman 2003] G. Forman. An Extensive Empirical Study of Feature Selection Metrics for Text Classification. *Journal of Machine Learning Research* 3:1289-1305, 2003

[Frakes and Baeza-Yates 1992] W.B. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures & Algorithms.* Prentice Hall, Upper Saddle River, NJ, USA 1992

[Frigui and Krishnapuram 1997] H. Frigui and R. Krishnapuram. Clustering by Competitive Agglomeration. *Pattern Recognition* 30(7):1109–1119. Pergamon Press, Oxford, United Kingdom 1997

[Frigui and Nasraoui 2003] H. Frigui and O. Nasraoui. Simultaneous Clustering and Dynamic Keyword Weighting for Text Documents. In: [Berry 2003], 45–72.

[Fukunaga 1990] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, San Diego, CA, USA 1990

[Fukuyama and Sugeno 1989] Y. Fukuyama and M. Sugeno. A New Method of Choosing the Number of Clusters for the Fuzzy *c*-Means Method. *Proc. 5th Fuzzy Systems Symposium (in Japanese)*, 247–256. Japan Society for Fuzzy Sets and Systems, Kobe, Japan 1989

[Gabriel and Berthold 2003a] T.R. Gabriel and M.R. Berthold. Formation of Hierarchical Fuzzy Rule Systems. *Proc. Conf. North American Fuzzy Information Processing Society (NAFIPS 2003, Chicago, IL).* IEEE Press, Piscataway, NJ, USA 2003

[Gabriel and Berthold 2003b] T.R. Gabriel and M.R. Berthold. Constructing Hierarchical Rule Systems. *Proc. 5th Int. Symposium on Intelligent*

Data Analysis (IDA 2003, Berlin, Germany), 76–87. Springer-Verlag, Berlin, Germany 2003

[Gath and Geva 1989] I. Gath and A.B. Geva. Unsupervised Optimal Fuzzy Clustering. *IEEE on Trans. Pattern Analysis and Machine Intelligence (PAMI)* 11:773–781. IEEE Press, Piscataway, NJ, USA 1989. Reprinted in [Bezdek and Pal 1992], 211–218

[Gerdes *et al.* 2004] I. Gerdes, F. Klawonn, and R. Kruse. *Genetische Algorithmen*. Vieweg, Braunschweig/Wiesbaden, Germany 2004

[Gersho 1982] A. Gersho. On the Structure of Vector Quantizers. *IEEE Trans. on Information Theory* 28(2):157–166. IEEE Press, Piscataway, NJ, USA 1982

[Goldberg 1989] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading, MA, USA 1989

[Golub and Van Loan 1989] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, USA 1989

[Good 1965] I.J. Good. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press, Cambridge, MA, USA 1965

[Good 1999] P. Good. *Resampling Methods*. Springer-Verlag, New York, NY, USA 1999

[Graepel and Obermayer 1998] T. Graepel and K. Obermayer. Fuzzy Topographic Kernel Clustering. *Proc. 5th GI Workshop Fuzzy-Neuro Systems*, 90–97. Gesellschaft für Informatik, Munich, Germany 1998

[Gray 1984] R.M. Gray. Vector Quantization. *IEEE Acoustic, Speech and Signal Processing Magazine* 1(2):4–29. IEEE Press, Piscataway, NJ, USA 1984

[Greiff 1998] W.R. Greiff. A Theory of Term Weighting Based on Exploratory Data Analysis. *Proc. 21st Ann. Int. Conf. on Research and Development in Information Retrieval (Sydney, Australia)*, 17–19. ACM Press, New York, NY, USA 1998

[Greiner 1989] W. Greiner. *Mechanik, Teil 2 (Series: Theoretische Physik)*. Verlag Harri Deutsch, Thun/Frankfurt am Main, Germany 1989

[Greiner *et al.* 1987] W. Greiner, L. Neise, and H. Stöcker. *Thermodynamik und Statistische Mechanik (Series: Theoretische Physik)*. Verlag Harri Deutsch, Thun/Frankfurt am Main, Germany 1987

[Grossman and Frieder 2004] D.A. Grossman and O. Frieder. *Information Retrieval*. Kluwer, Dordrecht, Netherlands 2004

[Gustafson and Kessel 1979] E.E. Gustafson and W.C. Kessel. Fuzzy Clustering with a Fuzzy Covariance Matrix. *Proc. of the IEEE Conf. on Decision and Control (CDC 1979, San Diego, CA)*, 761–766. IEEE Press, Piscataway, NJ, USA 1979. Reprinted in [Bezdek and Pal 1992], 117–122

[Halkidi *et al.* 2002a] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering Validity Checking Methods: Part I. *ACM SIGMOD Record* 31(2):40–45. ACM Press, New York, NY, USA 2002

[Halkidi *et al.* 2002b] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering Validity Checking Methods: Part II. *ACM SIGMOD Record* 31(3):19–27. ACM Press, New York, NY, USA 2002

[Hall *et al.* 1994] L.O. Hall, J.C. Bezdek, S. Boggavarapu, and A. Bensaid. Genetic Fuzzy Clustering. *Proc. 13th Int. Conf. North American Fuzzy Information Processing Society (NAFIPS'94, San Antonio, TX)*, 411–415. IEEE Press, Piscataway, NJ, USA 1994

[Hall *et al.* 1999] L.O. Hall, I.B. Özyurt, and J.C. Bezdek. Clustering with a Genetically Optimized Search. *IEEE Trans. on Evolutionary Computation* 3:102–112. IEEE Press, Piscataway, NJ, USA 1999

[Hand *et al.* 2001] D.J. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. Bradford Books/MIT Press, Cambridge, MA, USA 2001

[Hanley and McNeil 1982] J.A. Henley and B.J. McNeil. The Meaning and Use of the Area under a Receiver Operating Characteristic (ROC) Curve. *Radiology* 143:29-36. Radiological Society of North America, Oak Brook, IL, USA 1982

[Hartigan und Wong 1979] J.A. Hartigan and M.A. Wong. A k-means Clustering Algorithm. *Applied Statistics* 28:100–108. Blackwell, Oxford, United Kingdom 1979

[Hathaway and Bezdek 1995] R.J. Hathaway and J.C. Bezdek. Optimization of Clustering Criteria by Reformulation. *IEEE Trans. on Fuzzy Systems* 3:241–145. IEEE Press, Piscataway, NJ, USA 1995

[Haykin 1994] S. Haykin. *Neural Networks — A Comprehensive Foundation*. Prentice Hall, Upper Saddle River, NJ, USA 1994

[Heckerman 1998] D. Heckerman. *A Tutorial on Learning with Bayesian Networks*. In: [Jordan 1998], 301–354.

[Hershfinkel and Dinstein 1996] D. Hershfinkel and I. Dinstein. *Proc. SPIE Applications of Fuzzy Logic Technology III* 2761:41–52. Int. Society for Optical Engineering, Bellingham, WA, USA 1996

[Hong *et al.* 2003] X. Hong, P.M. Sharkey, and K. Warwick. Automatic Nonlinear Predictive Model Construction Algorithm using Forward Regression and the PRESS Statistic. *IEE Proc. Control Theory and Applications* 150(3):245–254. IEE Computer Society Press, San Diego, CA, USA 2003

[Höppner *et al.* 1999] F. Höppner, F. Klawonn, R. Kruse, and T. Runkler. *Fuzzy Cluster Analysis.* J. Wiley & Sons, Chichester, United Kingdom 1999

[Huffman 1952] D.A. Huffman. A Method for the Construction of Minimum Redundancy Codes. *Proc. Institute of Radio Engineers* 40(9):1098–1101. Institute of Radio Engineers, Menasha, WI, USA 1952

[Isbell and Viola 1998] C.L. Isbell and P. Viola. Restructuring Sparse High Dimensional Data for Effective Retrieval. *Proc. Conf. on Neural Information Processing (NIPS'98, Denver, CO)*, 480–486. NIPS (online proceedings) 1998

[Ismael and Selim 1986] M.A. Ismael and S.Z. Selim. Fuzzy c-Means: Optimality of Solutions and Effective Termination of the Algorithm. *Pattern Recognition* 19(6):481–485. Pergamon Press, Oxford, United Kingdom 1986

[Jain and Dubes 1988] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, NJ, USA 1988

[Jain and Moreau 1986] A.K. Jain and J. Moreau. Bootstrap Technique in Cluster Analysis. *Pattern Recognition* 20:547–569. Pergamon Press, Oxford, United Kingdom 1986

[Jakobs 1988] R.A. Jakobs. Increased Rates of Convergence Through Learning Rate Adaption. *Neural Networks* 1:295–307. Pergamon Press, Oxford, United Kingdom 1988

[Jamshidian and Jennrich 1993] M. Jamshidian and R.I. Jennrich. Conjugate Gradient Acceleration of the EM Algorithm. *Journal of the American Statistical Society* 88(412):221–228. American Statistical Society, Providence, RI, USA 1993

[Johnson 1967] S.C. Johnson. Hierarchical Clustering Schemes. *Psychometrika* 32:241–254. Psychometric Society, USA 1967

[Jones 1997] K.S. Jones, ed. *Readings in Information Retrieval.* Morgan Kaufmann, San Mateo, CA, USA 1997

[Jordan 1998] M.I. Jordan, ed. *Learning in Graphical Models.* MIT Press, Cambridge, MA, USA 1998

[Karayiannis and Bezdek 1997] N.B. Karayiannis and J.C. Bezdek. An Integrated Approach to Fuzzy Learning Vector Quantization and Fuzzy c-Means Clustering. *IEEE Trans. on Fuzzy Systems* 5(4):622–628. IEEE Press, Piscataway, NJ, USA 1997

[Karayiannis and Pai 1996] N.B. Karayiannis and P.-I. Pai. Fuzzy Algorithms for Learning Vector Quantization. *IEEE Trans. on Neural Networks* 7:1196–1211. IEEE Press, Piscataway, NJ, USA 1996

[Kaski 1998] S. Kaski. Dimensionality Reduction by Random Mapping: Fast Similarity Computation for Clustering. *Proc. Int. Joint Conf. on Artificial Neural Networks (IJCNN'98, Anchorage, Alaska)*, 1:413–418. IEEE Press, Piscataway, NJ, USA 1998

[Kaufman and Rousseeuw 1990] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. J. Wiley & Sons, New York, NY, USA 1990

[Kearns *et al.* 1997] M. Kearns, Y. Mansour, A.Y. Ng, and D. Ron. An Experimental and Theoretical Comparison of Model Selection Methods. *Machine Learning* 27(1):7–50. Kluwer, Dordrecht, Netherlands 1997

[Keller 2000] A. Keller. Fuzzy Clustering with Outliers. *Proc. 19th Conf. North American Fuzzy Information Processing Society (NAFIPS'00, Atlanta, Canada)*, 143–147. IEEE Press, Piscataway, NJ, USA 2000

[Keller 2002] A. Keller. *Objective Function Based Fuzzy Clustering in Air Traffic Management*. Ph.D. thesis, University of Magdeburg, Germany 2002

[Keller and Klawonn 2003] A. Keller and F. Klawonn. Adaptation of Cluster Sizes in Objective Function Based Fuzzy Clustering. In [Leondes 2003], 181–199.

[Kharin 1997] Y. Kharin. Robustness of Clustering under Outliers. In [Liu *et al.* 1997], 501–511.

[Kirkpatrick *et al.* 1983] S. Kirkpatrick, C.D. Gelatt, and M.P. Vercchi. Optimization by Simulated Annealing. *Science* 220:671–680. High Wire Press, Stanford, CA, USA 1983

[Kirsten 2002] M. Kirsten. *Multirelational Distance-Based Clustering*. Ph.D. Thesis, Otto-von-Guericke-University of Magdeburg, Magdeburg, Germany 2002

[Klawonn and Höppner 2003] F. Klawonn and F. Höppner. What is Fuzzy about Fuzzy Clustering? Understanding and Improving the Concept of

the Fuzzifier. *Proc. 5th Int. Symposium on Intelligent Data Analysis (IDA 2003, Berlin, Germany)*, 254–264. Springer-Verlag, Berlin, Germany 2003

[Klawonn and Keller 1998] F. Klawonn and A. Keller. Fuzzy Clustering with Evolutionary Algorithms. *Int. Journal of Intelligent Systems* 13:975–991. J. Wiley & Sons, Chichester, United Kingdom 1998

[Klawonn and Keller 1999] F. Klawonn and A. Keller. Fuzzy Clustering based on Modified Distance Measures. *Proc. 3rd Int. Symposium on Intelligent Data Analysis (IDA'99, Amsterdam, Netherlands)*, 291–301. Springer-Verlag, Berlin, Germany 1999

[Klawonn and Kruse 1997] F. Klawonn and R. Kruse. Constructing a Fuzzy Controller from Data. *Fuzzy Sets and Systems* 85:177–193. North-Holland, Amsterdam, Netherlands 1997

[Klir and Yuan 1997] G.J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications.* Prentice Hall, Upper Saddle River, NJ, USA 1997

[Klose *et al.* 2000] A. Klose, A. Nürnberger, R. Kruse, G.K. Hartmann, and M. Richards. Interactive Text Retrieval Based on Document Similarities. *Physics and Chemistry of the Earth, (Part A: Solid Earth and Geodesy)* 25:649–654. Elsevier Science, Amsterdam, Netherlands 2000

[Klose 2004] A. Klose. *Partially Supervised Learning of Fuzzy Classification Rules.* Ph.D. thesis, University of Magdeburg, Germany 2004

[Kohonen 1986] T. Kohonen. *Learning Vector Quantization for Pattern Recognition.* Technical Report TKK-F-A601. Helsinki University of Technology, Finland 1986

[Kohonen 1990] T. Kohonen. Improved Versions of Learning Vector Quantization. *Proc. Int. Joint Conference on Neural Networks* 1:545–550. IEE Computer Society Press, San Diego, CA, USA 1990

[Kohonen 1995] T. Kohonen. *Self-Organizing Maps.* Springer-Verlag, Berlin, Germany 1995 (3rd ext. edition 2001)

[Kolodner 1993] J. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann, San Mateo, CA, USA 1993

[Koza 1992] J.R. Koza. *Genetic Programming 1 & 2.* MIT Press, Cambridge, CA, USA 1992/1994

[Krishnapuram and Freg 1992] R. Krishnapuram and C.-P. Freg. Fitting an Unknown Number of Lines and Planes to Image Data through Com-

patible Cluster Merging. Pergamon Press, Oxford, United Kingdom 1992

[Krishnapuram and Keller 1993] R. Krishnapuram and J.M. Keller. A Possibilistic Approach to Clustering. *IEEE Trans. on Fuzzy Systems* 1(2):98–110. IEEE Press, Piscataway, NJ, USA 1993

[Krishnapuram and Keller 1996] R. Krishnapuram and J.M. Keller. The Possibilistic c-Means Algorithm: Insights and Recommendations. *IEEE Trans. on Fuzzy Systems* 4(3):385–393. IEEE Press, Piscataway, NJ, USA 1996

[Krishnapuram *et al.* 1999] R. Krishnapuram, A. Joshi, and L. Yi. A Fuzzy Relative of the k-Medoids Algorithm with Applications to Web Document and Snippet Clustering. *Proc. 8th IEEE Int. Conference on Fuzzy Systems (FUZZ-IEEE'99, Seoul, Korea)*, 1281–1286. IEEE Press, Piscataway, NJ, USA 1999

[Kruse *et al.* 1994] R. Kruse, J. Gebhardt, and F. Klawonn. *Foundations of Fuzzy Systems*, J. Wiley & Sons, Chichester, United Kingdom 1994. Translation of the book: *Fuzzy Systeme (Series: Leitfäden und Monographien der Informatik)*. Teubner, Stuttgart, Germany 1993 (second edition 1995)

[Kuhn and Tucker 1951] H. Kuhn and A. Tucker. Nonlinear Programming. *Proc. 2nd Berkeley Symp. on Mathematical Statistics and Probabilistics*, 481–492. University of California Press, Berkeley, CA, USA 1951

[Kullback and Leibler 1951] S. Kullback and R.A. Leibler. On Information and Sufficiency. *Annals of Mathematical Statistics* 22:79–86. Institute of Mathematical Statistics, Hayward, CA, USA 1951

[Langley *et al.* 1992] P. Langley, W. Iba, and K. Thompson. An Analysis of Bayesian Classifiers. *Proc. 10th Nat. Conf. on Artificial Intelligence (AAAI'92, San Jose, CA, USA)*, 223–228. AAAI Press / MIT Press, Menlo Park / Cambridge, CA, USA 1992

[Law and Jain 2003] M.H.C. Law and A.K. Jain. *Cluster Validity by Bootstrapping Partitions*. Technical Report MSU-CSE-03-5, Dept. of Computer Science and Engineering, Michigan State University, Michigan, , USA 2003

[Law *et al.* 2004] M.H.C. Law, M.A.T. Figueiredo, and A.K. Jain . Simultaneous Feature Selection and Clustering Using Mixture Models. *IEEE Trans. ON Pattern Analysis and Machine Intelligence (PAMI)* 26(9):1154–1166. IEEE Press, Piscataway, NJ, USA 2004

[Leondes 2003] C.T. Leondes, ed. *Database and Learning Systems IV*. CRC Press, Boca Raton, FL, USA 2003

[Lesot *et al.* 2003] M.-J. Lesot, F. d'Alché Buc, and G. Siolas. Evaluation of Topographic Clustering and Its Kernelization. *Proc. European Conference on Machine Learning (ECML'03, Cavtat-Dubrovnik, Croatia)*, 265–276. Springer-Verlag, Heidelberg, Germany 2003

[Levenberg 1944] K. Levenberg. A Method for the Solution of Certain Problems in Least Squares. *Quarterly Journal of Applied Mathematics* 2:164–168. American Mathematical Society, Providence, RI, USA 1944

[Levine and Domany 2001] E. Levine and E. Domany. Resampling Method for Unsupervised Estimation of Cluster Validity. *Neural Computation* 13:2573–2593. MIT Press, Cambridge, MA, USA 2001

[Liu *et al.* 1997] X. Liu, P. Cohen, and M. Berthold, eds. *Advances in Intelligent Data Analysis 2 (Proc. 2nd Symposium on Intelligent Data Analysis, IDA'97, London, UK)*. Springer-Verlag, Berlin, Germany 1997

[Lloyd 1982] S. Lloyd. Least Squares Quantization in PCM. *IEEE Trans. on Information Theory* 28:129–137. IEEE Press, Piscataway, NJ, USA 1982

[Lochbaum and Streeter 1989] K.E. Lochbaum and L.A. Streeter. Combining and Comparing the Effectiveness of Latent Semantic Indexing and the Ordinary Vector Space Model for Information Retrieval. *Information Processing and Management* 25:665–676. Elsevier Science, Amsterdam, Netherlands 1989

[MacQueen 1967] J. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability* I:281–297. University of California, Berkeley, CA, USA 1967

[Mahalanobis 1936] P.C. Mahalanobis. On the Generalized Distance in Statistics. *Proc. Nat. Inst. Sci. India* 2:49–55. India 1936

[Mamdani and Assilian 1975] E.H. Mamdani and S. Assilian. An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller. *Int. Journal of Man Machine Studies* 7:1-13. Academic Press, New York, NY, USA 1975

[Marquardt 1963] D.W. Marquardt. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics (SIAM)* 11:431–441. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA 1963

[Mendes and Sacks 2001] M.E.S. Mendes and L. Sacks. Dynamic Knowledge Representation for e-Learning Applications. *Proc. BISC Int. Workshop on Fuzzy Logic and the Internet (FLINT 2001, UC Berkeley, CA)*, 176–181. Memorandum No. UCB/ERL M01/28, University of California, Berkeley, CA, USA 2001

[Mendes and Sacks 2003] M.E.S. Mendes and L. Sacks. Evaluating Fuzzy Clustering for Relevance-based Information Access. *Proc. 12th IEEE Int. Conference on Fuzzy Systems (FUZZ-IEEE'03, St. Louis, MO, USA)*, on CDROM. IEEE Press, Piscataway, NJ, USA 2003

[Metropolis *et al.* 1953] N. Metropolis, N. Rosenblut, A. Teller, and E. Teller. Equation of State Calculations for Fast Computing Machines. *Journal of Chemical Physics* 21:1087–1092. American Institute of Physics, Melville, NY, USA 1953

[Michalewicz 1996] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Berlin, Germany 1996

[Michalski *et al.* 1983] R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, ed. *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann, San Mateo, CA, USA 1983

[Mitchell 1997] T.M. Mitchell. *Machine Learning.* McGraw-Hill, New York, NY, USA 1997

[Mitchell 1998] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, USA 1998

[Mladenic 2001] D. Mladenic. Using Text Learning to Help Web Browsing. *Proc. 9th Int. Conf. on Human-Computer Interaction.* New Orleans, LA, USA 2001

[Mosier 1951] C. Mosier. Problems and Design of Cross Validation. *Educational and Psychological Measurement* 11:5–11. Sage Publications, Thousand Oaks, CA, USA 1951

[Mucha 1992] H.-J. Mucha. *Clusteranalyse mit Mikrocomputern.* Akademie-Verlag, Berlin, Germany 1992

[Murthy *et al.* 1994] S.K. Murthy, S. Kasif, and S. Salzberg. A System for Induction of Oblique Decision Trees. *Journal of Artificial Intelligence Research* 2:1–32. Morgan Kaufmann, San Mateo, CA, USA 1994

[Myers 1990] R.H. Myers. *Classical and Modern Regression with Applications (2nd edition).* PWS-Kent, Boston, MA, USA 1990

[Nakhaeizadeh 1998a] G. Nakhaeizadeh, ed. *Data Mining: Theoretische Aspekte und Anwendungen*. Physica-Verlag, Heidelberg, Germany 1998

[Nakhaeizadeh 1998b] G. Nakhaeizadeh. Wissensentdeckung in Datenbanken und Data Mining: Ein Überblick. In: [Nakhaeizadeh 1998a], 1–33

[Nauck and Kruse 1997] D. Nauck and R. Kruse. A Neuro-Fuzzy Method to Learn Fuzzy Classification Rules from Data. *Fuzzy Sets and Systems* 89:277–288. North-Holland, Amsterdam, Netherlands 1997

[Nauck *et al.* 1997] D. Nauck, F. Klawonn, and R. Kruse. *Foundations of Neuro-Fuzzy Systems*. J. Wiley & Sons, Chichester, United Kingdom 1997

[Nauck *et al.* 2003] D. Nauck, C. Borgelt, F. Klawonn, and R. Kruse. *Neuro-Fuzzy-Systeme — Von den Grundlagen künstlicher neuronaler Netze zur Kopplung mit Fuzzy-Systemen*. Vieweg, Braunschweig/Wiesbaden, Germany 2003

[Nilsson 1998] N.J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, San Francisco, CA, USA 1998

[NNRC 2002] Neural Networks Research Centre. *Bibliography on the Self-organizing Map and Learning Vector Quantization*. Laboratory of Computer and Information Science, Helsinki University of Technology, Espoo, Finland 2002
`http://www.cis.hut.fi/research/som-bibl/references.bib`

[Nascimiento and Moura-Pires 1997] S. Nascimiento and F. Moura-Pires. A Genetic Approach for Fuzzy Clustering In: [Liu *et al.* 1997], 325–335.

[Nürnberger *et al.* 1999] A. Nürnberger, C. Borgelt, and A. Klose. Improving Naive Bayes Classifiers Using Neuro-Fuzzy Learning. *Proc. Int. Conf. on Neural Information Processing (ICONIP'99, Perth, Australia)*, 154–159. IEEE Press, Piscataway, NJ, USA 1999

[Ohashi 1984] Y. Ohashi. Fuzzy Clustering and Robust Estimation. *Proc. 9th Meeting SAS Users Group Int.* Hollywood Beach, FL, USA 1984

[Pal and Bezdek 1995] N.R. Pal and J.C. Bezdek. On Cluster Validity for the Fuzzy c-Means Model. *IEEE Trans. on Fuzzy Systems* 3:370–379. IEEE Press, Piscataway, NJ, USA 1995

[Papadimitriou and Steiglitz 1982] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization, Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, USA 1982

[Parzen 1962] E. Parzen. On Estimation of a Probability Density Function and Mode. *Annals of Mathematical Statistics* 33:1065–1076. Institute of Mathematical Statistics, Beachwood, OH, USA 1962

[Pedrycz 1988] W. Pedrycz. *Fuzzy Control and Fuzzy Systems.* J. Wiley & Sons, Chichester, United Kingdom 1988

[Petry *et al.* 1994] F. Petry, B. Buckles, D. Prabhu, R. George, and R. Srikanth. Fuzzy Clustering with Genetic Search. *Proc. 1st IEEE Conf. on Evolutionary Computation (ICEC)*, 46–50. IEEE Press, Piscataway, NJ, USA 1994

[Porter 1980] M. Porter. An Algorithm for Suffix Stripping. *Program: Electronic Library & Information Systems* 14(3):130–137. Emerald, Bradford, United Kingdom 1980

[Preparata and Shamos 1985] F.R. Preparata, and M.I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, NY, USA 1985

[Press *et al.* 1992] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C — The Art of Scientific Computing (2nd edition).* Cambridge University Press, Cambridge, United Kingdom 1992

[Quinlan 1986] J.R. Quinlan. Induction of Decision Trees. *Machine Learning* 1:81–106. Kluwer, Dordrecht, Netherlands 1986

[Quinlan 1993] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, USA 1993

[Reusch and Temme 2001] B. Reusch and K.-H. Temme, eds. *Computational Intelligence in Theory and Practice.* Series: Advances in Soft Computing. Physica-Verlag, Heidelberg, Germany 2001

[Riedmiller and Braun 1993] M. Riedmiller and H. Braun. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. *Int. Conf. on Neural Networks (ICNN-93, San Francisco, CA)*, 586–591. IEEE Press, Piscataway, NJ, USA 1993

[Rijsbergen 1979] C.J. van Rijsbergen. *Information Retrieval.* Butterworth, London, United Kingdom 1979

[Rijsbergen 1986] C.J. van Rijsbergen. A Non-classical Logic for Information Retrieval. *The Computer Journal* 29:481–485. Oxford University Press, Oxford, United Kingdom 1986

[Rissanen 1983] J. Rissanen. A Universal Prior for Integers and Estimation by Minimum Description Length. *Annals of Statistics* 11:416–431. Institute of Mathematical Statistics, Hayward, CA, USA 1983

[Rissanen 1987] J. Rissanen. Stochastic Complexity. *Journal of the Royal Statistical Society (Series B)*, 49:223-239. Blackwell, Oxford, United Kingdom 1987

[Robertson 1977] S.E. Robertson. The Probability Ranking Principle. *Journal of Documentation* 33:294–304. Emerald, Bradford, United Kingdom 1977

[Robbins and Monro 1951] H. Robbins and S. Monro. A Stochastic Approximation Method. *Ann. Math. Stat.* 22:400–407. Institute of Mathematical Statistics, Beachwood, OH, USA 1951

[Rojas 1993] R. Rojas. *Theorie der Neuronalen Netze: Eine systematische Einführung*. Springer-Verlag, Berlin, Germany 1993

[Rose 1998] K. Rose. Deterministic Annealing for Clustering, Compression, Classification, Regression, and Related Optimization Problems. *Proc. IEEE* 86(11):2210–2239. IEEE Press, Piscataway, NJ, USA 1998

[Roth *et al.* 2002] V. Roth, T. Lange, M. Braun, and J.M. Buhmann. A Resampling Approach to Cluster Validation. *Proc. Computational Statistics (CompStat'02, Berlin, Germany)*, 123–128. Springer, Heidelberg, Germany 2002

[Roth and Lange 2003] V. Roth and T. Lange. Feature Selection in Clustering Problems. *Proc. 17th Ann. Conf. on Neural Information Processing Systems (NIPS'03, Vancouver, Canada)*. MIT Press, Cambridge, MA, USA 2004

[Runkler and Bezdek 1999] T.A. Runkler and J.C. Bezdek. Alternating Cluster Estimation: A New Tool for Clustering and Function Approximation. *IEEE Trans. on Fuzzy Systems* 7(4):377–393. IEEE Press, Piscataway, NJ, USA 1999

[Ruspini 1969] E.H. Ruspini. A New Approach to Clustering. *Information and Control* 15(1):22–32. Academic Press, San Diego, CA, USA 1969. Reprinted in [Bezdek and Pal 1992], 63–70

[Rumelhart *et al.* 1986] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning Representations by Back-Propagating Errors. *Nature* 323:533–536. Nature Publishing Group, Basingstoke, United Kingdom 1986

[Russel *et al.* 1995] K. Russel, J. Binder, D. Koller, and K. Kanazawa. Local Learning in Probabilistic Networks with Hidden Variables. *Proc. 1st*

Int. Conf. on Knowledge Discovery and Data Mining (KDD'95, Montreal, Canada), 1146–1152. AAAI Press, Menlo Park, CA, USA 1995

[Salakhutdinov *et al.* 2003] R. Salakhutdinov, S. Roweis, and Z. Ghahramani. Optimization with EM and Expectation-Conjugate-Gradient. *Proc. 20th Int. Conf. on Machine Learning (ICML'03,Washington, DC)*, 672–679. AAAI Press, Menlo Park, CA, USA 2003

[Salton *et al.* 1975] G. Salton, A. Wong, and C.S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM* 18:613–620. ACM Press, New York, NY, USA 1975

[Salton and Buckley 1988] G. Salton and C. Buckley. Term Weighting Approaches in Automatic Text Retrieval. *Information Processing & Management* 24:513–523. Elsevier Science, Amsterdam, Netherlands 1988

[Salton *et al.* 1994] G. Salton, J. Allan, and C. Buckley. Automatic Structuring and Retrieval of Large Text Files. *Communications of the ACM* 37:97–108. ACM Press, New York, NY, USA 1994

[Scheffer and Joachims 1999] T. Scheffer and T. Joachims. Expected Error Analysis for Model Selection. *Proc. 16th Int. Conf. on Machine Learning (ICML'99, Bled, Slovenia)*, 361–370. Morgan Kaufman, San Francisco, CA USA 1999

[Schnell 1964] P. Schnell. Eine Methode zur Auffindung von Gruppen. *Biometrika* 6:47–48. Biometrika Trust, London, United Kingdom 1964

[Schölkopf and Smola 2002] B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, USA 2002

[Schwarz 1978] G. Schwarz. Estimating the Dimension of a Model. *Annals of Statistics* 6:461–464. Institute of Mathematical Statistics, Hayward, CA, USA 1978

[Scott 1992] D.W. Scott. *Multivariate Density Estimation*. J. Wiley & Sons, New York, NY, USA 1992

[Sebastiani 2002] F. Sebastiani. Machine Learning in Automated Text Categorization. *ACM Computing Surveys (CSUR)* 34(1):1–47. ACM Press, New York, NY, USA 2002

[Selim and Ismael 1984] S.Z. Selim and M.A. Ismail. k-Means-Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)* 6(1):81–87. IEEE Press, Piscataway, NJ, USA 1984

[Seo and Obermayer 2003] S. Seo and K. Obermayer. Soft Learning Vector Quantization. *Neural Computation* 15(7):1589–1604. MIT Press, Cambridge, MA, USA 2003

[Shannon 1948] C.E. Shannon. The Mathematical Theory of Communication. *The Bell System Technical Journal* 27:379–423. Bell Laboratories, Murray Hill, NJ, USA 1948

[Sinka and Corne 2002] M.P. Sinka, and D.W. Corne. A Large Benchmark Dataset for Web Document Clustering. In [Abraham *et al.* 2002], 881–890.

[Sokal and Sneath 1963] R.R. Sokal and P.H.A. Sneath. *Principles of Numerical Taxonomy.* Freeman, San Francisco, CA, USA 1963

[Spall 2003] J.C. Spall. *Introduction to Stochastic Search and Optimization.* J. Wiley & Sons, Chichester, United Kingdom 2003

[Stutz 1998] C. Stutz. Partially Supervised Fuzzy c-Means Clustering with Cluster Merging. *Proc. 6th European Congress on Intelligent Techniques and Soft Computing (EUFIT'98, Aachen, Germany)*, 1725–1729. Verlag Mainz, Aachen, Germany 1998

[Takagi and Sugeno 1985] H. Takagi and M. Sugeno. Fuzzy Identification of Systems and Its Application to Modeling and Control. *IEEE Trans. on Systems, Man, and Cybernetics* 15:116–132. IEEE Press, Piscataway, NJ, USA 1985

[Tesauro *et al.* 1995] G. Tesauro, D.S. Touretzky, and T.K. Leen, eds. *Advances in Neural Information Processing Systems*, Vol. 7. MIT Press, Cambridge, MA, USA 1995

[Tikhonov and Arsenin 1977] A.N. Tikhonov and V.Y. Arsenin. Solutions of Ill-Posed Problems. J. Wiley & Sons, New York, NY, USA 1977

[Timm *et al.* 2001] H. Timm, C. Borgelt, C. Döring, and R. Kruse. Fuzzy Cluster Analysis with Cluster Repulsion. *Proc. European Symposium on Intelligent Technologies (EUNITE, Tenerife, Spain)*, on CDROM. Verlag Mainz, Aachen, Germany 2001

[Timm 2002] H. Timm. *Fuzzy-Clusteranalyse: Methoden zur Exploration von Daten mit fehlenden Werten sowie klassifizierten Daten.* Ph.D. thesis, University of Magdeburg, Germany 2002

[Timm and Kruse 2002] H. Timm and R. Kruse. A Modification to Improve Possibilistic Cluster Analysis. *Proc. IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE 2002, Honolulu, Hawaii).* IEEE Press, Piscataway, NJ, USA 2002

[Timm *et al.* 2004] H. Timm, C. Borgelt, C. Döring, and R. Kruse. An Extension to Possibilistic Fuzzy Cluster Analysis. *Fuzzy Sets and Systems* 147:3–16. Elsevier Science, Amsterdam, Netherlands 2004

[Tollenaere 1990] T. Tollenaere. SuperSAB: Fast Adaptive Backpropagation with Good Scaling Properties. *Neural Networks* 3:561–573. Pergamon Press, Oxford, United Kingdom 1990

[Touretzky *et al.* 1988] D. Touretzky, G. Hinton, and T. Sejnowski, eds. *Proc. Connectionist Models Summer School (Carnegie Mellon University, Pittsburgh, PA).* Morgan Kaufmann, San Mateo, CA, USA 1988

[Toussaint 1974] G. Toussaint. Bibliography on Estimation of Misclassification. *IEEE Trans. on Information Theory* 20(4):472–279. IEEE Press, Piscataway, NJ, USA 1974

[Tsao *et al.* 1994] E.C.-K. Tsao, J.C. Bezdek, and N.R. Pal. Fuzzy Kohonen Clustering Networks. *Pattern Recognition* 27(5):757–764. Pergamon Press, Oxford, United Kingdom 1994

[Turtle and Croft 1990] H. Turtle and W.B. Croft. Inference Networks for Document Retrieval. *Proc. 13th Int. Conf. on Research and Development in Information Retrieval*, 1–24. ACM Press, New York, NY, USA 1990

[Van Le 1995] T. Van Le. Evolutionary Fuzzy Clustering. *Proc. 2nd IEEE Conf. on Evolutionary Computation*, 2:753–758. IEEE Press, Piscataway, NJ, USA 1995

[Vapnik 1995] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, NY, USA 1995

[Vapnik 1998] V. Vapnik. *Statistical Learning Theory.* J. Wiley & Sons, Chichester, England 1998

[Ward 1963] J.H. Ward. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Society* 58:236–244. American Statistical Society, Providence, RI, USA 1963

[Wettschereck 1994] D. Wettschereck. *A Study of Distance-Based Machine Learning Algorithms.* PhD Thesis, Oregon State University, OR, USA 1994

[Whitley 2001] D. Whitley. An Overview of Evolutionary Algorithms: Practical Issues and Common Pitfalls. *Journal of Information and Software Technology* 43:817–831. Elsevier Science, Amsterdam, Netherlands 2001

[Windham 1982] M.P. Windham. Cluster Validity for the Fuzzy *c*-Means Algorithm. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)* 4(4): 357–363. IEEE Press, Piscataway, NJ, USA 1982

[Witten and Frank 1999] I.H. Witten and E. Frank. *Data Mining*. Morgan Kaufmann, San Mateo, CA, USA 1999

[Witten *et al.* 1999] I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Mateo, CA, USA 1999

[Wu 1983] C.F.J. Wu. On the Convergence Properties of the EM Algorithm. *Annals of Statistics* 11(1):95–103. Institute of Mathematical Statistics, Hayward, CA, USA 1983

[Xie and Beni 1991] X.L. Xie and G.A. Beni. Validity Measure for Fuzzy Clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 3(8):841–846. IEEE Press, Piscataway, NJ, USA 1991. Reprinted in [Bezdek and Pal 1992], 219–226

[Yair *et al.* 1992] E. Yair, K. Zeger, and A. Gersho. Competitive Learning and Soft Competition for Vector Quantizer Design. *IEEE Trans. on Signal Processing* 40:294–309. IEEE Press, Piscataway, NJ, USA 1992

[Yager and Filev 1994] R.R. Yager and D.P. Filev. Generation of Fuzzy Rules by Mountain Clustering. *Journal of Intelligent & Fuzzy Systems* 2(3):209–219. IEEE Press, Piscataway, NJ, USA 1994

[Yang and Pedersen 1997] Y. Yang and J.O. Pedersen. A Comparative Study on Feature Selection in Text Categorization. *Proc. 14th Int. Conf. on Machine Learning (ICML'97, Nashville, TN)*, 412–420. Morgan Kaufmann, San Mateo, CA, USA 1997

[Zadeh 1965] L.A. Zadeh. Fuzzy Sets. *Information and Control* 8:338–353. Academic Press, San Diego, CA, USA 1965

[Zell 1994] A. Zell. *Simulation Neuronaler Netze*. Addison Wesley, Bonn, Germany 1994

[Zhang *et al.* 2004] D. Zhang, S. Chen, and Z.-H. Zhou. Fuzzy-kernel Learning Vector Quantization. *Proc. 1st Int. Symp. on Neural Networks (ISNN'04, Dalian, China)*, LNCS 3173, 180–185. Springer-Verlag, Berlin, Germany 2004

Index

0-1 loss, 4, 59, 202
1-in-n encoding, 3, 5
11-point average precision, 207

abalone data, 239
absolute loss, 60, 202
acceleration, 188–195, 239
    momentum term, 191
    quick backpropagation, 193
    resilient backpropagation, 192
    step expansion, 190
    super self-adaptive backpropagation, 191
accuracy, 205
    cross-classification, 226
ACE, 70, 139
activation, 159
adjoint matrix, 269
AIC, 92
Akaike information criterion, 92
alternating cluster estimation, 70, 139
alternating optimization, 120
annealing, 172
argument, 301
arithmetic mean, 46, 290
attraction rule, 153
attribute, 2
    hidden, 139
    informationless, 128
    metric, 2
    nominal, 2
    ordinal, 2
average, 290
    arithmetic, 290
    geometric, 291
    harmonic, 291
    quadratic, 290
average linkage, 84

backpropagation, 190
    momentum term, 191
    quick, 193
    resilient, 192
    super self-adaptive, 191
backward elimination, 90
backward regression, 90
backward substitution, 280
bag of words, 243
bandwidth, 82
batch training, 102
Bayes classifier, 5, 40, 77, 198
    full, 5, 79
    naïve, 5, 79
Bayes error, 198
Bayesian classification, 198
Bayesian information criterion, 92
Bayesian network model, 243
bias error, 199
bias value, 94, 285
BIC, 92
bootstrapping, 229

breakeven point, 207
breast data, 239

c-means clustering, 6, 47, 121
c-medoids clustering, 47
case-based reasoning, 8
Cauchy function, 16, 271
centroid method, 84
χ^2 measure, 228
Cholesky decomposition, 14, 278
chromosome, 173
city block distance, 12
class, 4
class membership function, 38
classification, 2–5
  accuracy, 205
  crisp, 4
  error, 198
  fuzzy, 4
  odds, 66
  problem, 2
  quality, 4, 198–209
  supervised, 2
  unsupervised, 2
classifier, 2, 4
  construction, 2
  linear function, 38
  maximum margin, 69
  maximum membership, 38
  nearest prototype, 38
closeness, 5
cluster
  center, 20
  connectivity matrix, 226
  covariance matrix, 20
  hierarchy, 85
  location, 20
  membership degree, 20
  membership function, 20
  membership matrix, 94
  number of, 20, 221, 231
  prototype, 11, 20
  shape, 20, 21
  size, 20, 21
  weight, 20
clustering, 2–5
  acceleration, 188
  ACE, 70
  alternating cluster estimation, 70, 139
  c-means, 47, 121
  c-medoids, 47
  clustering results, 87
  convergence, 71
  crisp, 3
  distance-based, 4
  FMLE, 65, 150, 235
  fuzzy, 48, 125
  fuzzy c-means, 6, 48, 125, 240
  fuzzy maximum likelihood estimation, 65, 150, 235
  Gath–Geva, 65, 150, 235
  Gustafson–Kessel, 125, 136, 234, 241
  hard, 3
  hard c-means, 6, 47
  hierarchical agglomerative, 8, 84
  ISODATA, 122
  k-means, 121
  Lloyd's algorithm, 122
  mountain, 81
  noise, 178
  possibilistic fuzzy, 53, 159
  probabilistic, 4
  probabilistic fuzzy, 53
  quality, 209–231
  regularization, 183
  subtractive, 81
cofactor, 269
cofactor matrix, 269

coincidence matrix, 226
commit step, 89
competitive learning, 152–171
  frequency sensitive, 162
  fuzzy, 159
  maximum likelihood ratio, 165–171
  size and shape, 162–165
complete linkage, 84
condition number, 284
conflict region, 89
contingency table, 224, 227
convergence, 71
core region, 89
cosine down to zero, 18
coupled prototypes, 266
covariance matrix, 13, 14, 20
  "square root", 278, 280
  determinant, 279, 283
  fuzzy, 136
  inverse, 279, 283
crisp assignment, 27, 48
crisp classification, 4
crisp clustering, 3
crisp prediction, 4
cross entropy, 255
cross validation, 200–201
  k-fold, 201
  leave-1-out, 93, 96
cross-classification accuracy, 226
crossing-over, 174
curse of dimensionality, 231

data analysis, 1
data matrix, 2
data set, 2
  abalone, 239
  breast, 239
  iris, 55, 79, 234, 239
  wine, 235, 239
Davies–Bouldin index, 213
DDA, 88
decision tree, 7
decomposition
  Cholesky, 14, 278
  eigenvalue, 14, 280
  LU, 280
degree of membership, 4
Delaunay triangulation, 122
dendrogram, 85
density, 19
desired reference radius, 163
determinant, 269
differential operator, 267
discriminant analysis, 2
distance measure, 3, 12, 20
  city block, 12
  cosine-based, 15
  Euclidean, 12
  Mahalanobis, 13, 273, 279
  Manhattan, 12
  maximum, 12
  Minkowski family, 12
distance-based clustering, 4
distribution
  Maxwell, 173
  velocity, 173
document representation
  Bayesian network model, 243
  filtering and stemming, 246
  index term selection, 247
  keyword selection, 247
  keyword weighting, 254
  logical model, 243
  preprocessing, 243
  probabilistic model, 243
  rescaling, 256
  vector space model, 243
Dunn index, 211
dynamic decay adjustment, 88

eigensystem, 15
eigenvalue, 14, 280
    decomposition, 14, 280
    degenerate, 281
eigenvector, 14, 280
EM, 6, 139
    algorithm, 62, 65, 139
encoding
    1-in-n, 3, 5
energy, 172
energy minimum, 172
entropy, 247
    Shannon, 247
epoch, 89, 155
error
    absolute, 60
    Bayes, 198
    bias, 199
    classification, 198
    scatter, 199
    squared, 60
    variance, 199
Euclidean distance, 12
evaluation measure, 202
    0-1 loss, 202
    absolute loss, 202
    classification, 202
    external, 210
    F_1 measure, 207
    internal, 210
    precision, 203
    quadratic loss, 202
    recall, 203
    relative, 210, 221
evolution, 173
evolutionary algorithms, 173–174
example case, 2
expansion by minors, 269
expectation maximization, 6, 139
    algorithm, 62, 65, 139
expectation step, 143, 145
exponential decay, 153
external evaluation measure, 210

F_1 measure, 207, 226
factorial (generalized), 19
feature selection, 90, 266
filtering stop words, 246
fitness, 173
FMLE, 65, 150
fold, 201
Folkes–Mallows index, 228
forward regression, 90
forward selection, 90
forward substitution, 279
frequency sensitive competitive learning, 162
full Bayes classifier, 5, 79
function argument, 301
function parameter, 301
fuzzifier, 51, 238
fuzzy c-means clustering, 6, 48, 125, 240
fuzzy classification, 4
fuzzy clustering, 48
    c-means, 48
    constraints, 46
    possibilistic, 36, 53
    probabilistic, 53
fuzzy conjunction, 222
fuzzy covariance matrix, 136
fuzzy disjunction, 223
fuzzy learning vector quantization, 65, 159–162
fuzzy LVQ, 65
fuzzy maximum likelihood estimation, 65, 150
fuzzy negation, 222
fuzzy partition matrix, 50
fuzzy rule-based system, 7, 42

fuzzy set theory, 222
fuzzy system
    Mamdani–Assilian, 43
    neuro-fuzzy, 7, 42
    relational equations, 43

Γ-function, 19
Gath–Geva clustering, 65, 150, 235
Gaussian function, 18, 272
Gaussian mixture model, 62, 146
gene, 173
generalized coordinates, 288
generalized factorial, 19
genetic algorithms, 173–174
genotype, 173
geometric mean, 291
Gödel implication, 43
gradient, 54, 100
    ascent, 101
    color coding, 54
    descent, 101
    likelihood, 113
    likelihood ratio, 116
    problems, 117
    stochastic descent, 102
    sum of squared distances, 103
    sum of squared errors, 108
guided random search, 171
Gustafson–Kessel clustering, 125, 135, 136, 234, 241

hard assignment, 27
    restricted, 37
hard c-means clustering, 6
hard clustering, 3
harmonic mean, 291
Heron's algorithm, 142, 290
hidden variable, 139
hierarchical agglomerative clustering, 8, 84
Householder transformation, 282
Hubert index, 228
(hyper-)ellipsoid, 21
(hyper-)sphere
    surface area, 270
    volume, 283
hyperspherical fuzzy clustering, 245
(hyper-)volume, 21

ICA, 243
ideal gas, 173
ill-conditioned, 284
independent component analysis, 243
index
    Davies–Bouldin, 213
    Dunn, 211
    Folkes–Mallows, 228
    Fukuyama–Sugeno, 215
    fuzzy hyper-volume, 218
    Hubert, 228
    Jaccard, 228
    partition coefficient, 216
    partition density, 219
    partition entropy, 217
    Rand, 227
    separation, 211
    Xie–Beni, 214
index term selection, 247
information criterion, 91, 208
    Akaike, 92
    Bayesian, 92
information divergence, 169
information gain, 255
information retrieval, 242
informationless attribute, 128
initialization, 73, 259
    data independent, 74–76
    diagonal of the data space, 76
    Latin hypercube sampling, 75

Maximindist, 76
mountain clustering, 81
normal distribution, 75
refinement approach, 87
sample from a probabilistic model, 77
samples from the data set, 76
simple data dependent, 76–80
sophisticated, 81–93
subtractive clustering, 81
uniform distribution, 74
weight, 94–97
inner product, 268
intercept term, 94, 285
internal evaluation measure, 210
interval scale, 2
inverse document frequency, 243
inverse matrix, 269
iris data, 55, 79, 234, 239
ISODATA, 122
isotropic standard deviation, 22
isotropic variance, 22

Jaccard coefficient, 228
Jacobi formula
surface area, 270
volume, 283
Jacobi transformation, 282

k-means clustering, 121
k-nearest neighbor, 8
kernel based methods, 41
kernel estimation, 82
kernel function, 8, 81
keyword
extraction, 259
selection, 247
weighting, 254
Kronecker symbol, 60, 94, 297
Kullback–Leibler information divergence, 169

Lagrange
function, 289
multiplier, 288
theory, 288
latent semantic indexing, 243
latent variable, 139
Latin hypercube sampling, 75
lattice energy, 172
learning rate, 101, 117
time dependent, 155
learning vector quantization, 6, 152
classical, 152–158
crisp, 167, 170
fuzzy, 65, 159–162
hard, 167, 170
maximum likelihood ratio, 165–171
size and shape, 162–165
soft, 167, 170
leave-1-out cross validation, 93, 96
left triangular matrix, 278
Levenberg-Marquardt method, 195
likelihood, 4, 29, 62, 113, 140
log-, 64
ratio, 66
likelihood ratio, 116, 202
linear function classifier, 38
linearly separable, 69
linkage
average, 84
centroid, 84
complete, 84
single, 84
Ward's method, 85
Lloyd's algorithm, 122
local optimum, 119
log-likelihood, 64, 113, 140

logical model, 243
loss function, 4
    0-1, 4, 59, 202
    absolute, 60, 202
    quadratic, 60, 202
lower triangular matrix, 278
LSI, 243
LU decomposition, 280
LVQ
    fuzzy, 65

machine learning, 2
macro-averaging, 205, 225
Mahalanobis distance, 13, 273, 279
Mamdani–Assilian system, 43
Manhattan distance, 12
margin, 69
    soft, 69
    target, 69
mating, 174
matrix
    adjoint, 269
    cluster connectivity, 226
    cluster membership, 94
    cofactor, 269
    coincidence, 226
    covariance, 13, 14
    diagonal, 13
    fuzzy covariance, 136
    fuzzy partition, 50
    ill-conditioned, 284
    inverse, 269
    inversion lemma, 96, 287
    left triangular, 278
    lower triangular, 278
    orthogonal, 13, 281
    partition, 46, 224
    positive definite, 20, 278
    product, 268
    right triangular, 278
    symmetric, 20, 268
    transpose, 268
    upper triangular, 278
Maximindist, 76
maximization step, 143, 146
maximum distance, 12
maximum likelihood, 62
    estimation, 62
    ratio, 65
maximum margin classifier, 69
maximum membership classifier, 38
Maxwell distribution, 173
MDL, 92, 208
mean
    arithmetic, 46, 290
    geometric, 291
    harmonic, 291
    quadratic, 290
measure
    distance, 12
    evaluation, 202, 210
    penalized, 91, 208
    similarity, 16
median, 46, 47
medoid, 47
    clustering, 3, 41
membership degree, 20
    normalization, 23, 27
    transformation, 23
    weighting, 23
membership function, 20
membership transformation, 49
metal, 172
method of least squares, 63
metric, 12
metric scale, 2
micro-averaging, 206
migration, 174
minimum description length, 92, 208
Minkowski family, 12

minor, 269
  expansion by, 269
misclassification costs, 61
missing value, 139
mixture model, 28, 39, 140
  Gaussian, 62, 146
momentum term, 191
monocrystalline structure, 172
Monte Carlo method, 228
mountain clustering, 81
multilayer perceptron, 7
multilinear regression, 94, 285
mutation, 174
mutual information, 255

naïve Bayes classifier, 5, 79
nearest prototype classifier, 38
neural network
  multilayer perceptron, 7
  radial basis function, 6, 38, 100, 108
neuro-fuzzy system, 7, 42
noise cluster, 178
noise clustering, 178
nominal scale, 2
normal distribution initialization, 75
normal equations, 95, 286
normalization, 23
  crisp, 27
  factor, 19, 20, 270
  hard, 27
  maximum 1, 27
  mode, 22
  none, 27
  sum 1, 27
number of clusters, 220, 221, 231

object, 2
objective function, 45
odds of correct classification, 66
one-point crossover, 174
online training, 102
ordinal scale, 2
orthogonal linear transformation, 13
orthogonal matrix, 13, 281
outer product, 268
outlier, 178
overfitting, 199

parameter, 301
partition matrix, 46, 224
  fuzzy, 50
Parzen window, 82
penalized measure, 91
phenotype, 173
point prototype, 43
polycrystalline structure, 172
population, 173
positive definite, 20, 278
possibilistic fuzzy clustering, 53, 159
posterior probability, 29, 66, 77
power, 208
precision, 204, 225
predicted residual sum of squared errors, 93, 96
prediction, 4, 44
  crisp, 4
  function, 38, 60
PRESS, 93, 96
prior probability, 78
probabilistic clustering, 4
probabilistic fuzzy clustering, 53
probabilistic model, 243
probability density, 19
product
  inner, 268
  matrix, 268
  outer, 268
  scalar, 268
product-moment correlation, 228

prototype
coupled, 266
merging, 265
splitting, 265
pseudo inverse, 286

QR algorithm, 282
quadratic form, 15
quadratic loss, 60, 202
quadratic mean, 290
quality measure, 202
0-1 loss, 202
absolute loss, 202
classification, 202
external, 210
internal, 210
quadratic loss, 202
relative, 210, 221
quick backpropagation, 193

radial basis function network, 6, 38, 100, 108
radial function, 16, 20
Cauchy, 16
cosine down to zero, 18
Gauss, 18
reference point, 18
spline, 18
trapezoidal, 18
triangular, 18
radius, 16
desired reference, 163
reference, 18
Rand statistic, 227
random projection, 243
ratio scale, 2
RBF network, 100, 108
recall, 204, 225
receiver operating characteristic curve, 208
refinement approach, 87
reflection, 13
region
conflict, 89
core, 89
regression, 63
backward, 90
forward, 90
multilinear, 94, 285
ridge, 286
tree, 7
regressor variable, 285
regularization, 183–188, 234
shape, 183, 235
size, 186, 235
Tikhonov, 286
weight, 187, 238
relational equation, 43
relative evaluation measure, 210, 221
repulsion rule, 157
resampling, 228
resilient backpropagation, 192
response variable, 285
restricted hard assignment, 37
ridge regression, 286
right triangular matrix, 278
robustness, 178
ROC curve, 208
rotation, 13, 281
rule
attraction, 153
repulsion, 157
window, 158
rule-based system, 7
fuzzy, 7, 42

sampling
Latin hypercube, 75
stratified, 201
scalar product, 268

scale
  interval, 2
  invariant, 13
  metric, 2
  nominal, 2
  ordinal, 2
  ratio, 2
  types, 2
scaling, 281
scatter, 199
schema theorem, 174
selection, 173
self-organizing map, 266
separation index, 211
sequence of words, 243
sexual reproduction, 174
Shannon entropy, 247
shape, 20
  regularization, 183, 235
Sherman-Morrison formula, 287
shifting the eigenvalues, 184
shrink step, 89
similarity, 3, 5
similarity measure, 16
simulated annealing, 171–173
single linkage, 84
singular value, 284
singular value decomposition, 284
size, 20
  regularization, 186, 235
slack variable, 70
soft learning vector quantization, 167, 170
soft margin approach, 69
software, 10
spline, 18
  order, 18
standard deviation
  isotropic, 22
statistics, 2
stemming, 246
step expansion, 190
stochastic gradient descent, 102
stop word filtering, 246
stratification, 201
stratified sampling, 201
subsampling, 229
subtractive clustering, 81
sum of absolute errors, 60, 202
sum of squared distances, 46, 103
sum of squared errors, 5, 59, 60, 95, 108, 202, 285
  penalized, 287
  predicted residual, 93, 96
super self-adaptive backpropagation, 191
supervised classification, 2
support vector, 8, 69
support vector machine, 8, 41, 69
survival of the fittest, 174
SVM, 69
symmetric, 20, 268

t-conorm, 223
t-norm, 43, 222
target margin, 69
temperature, 173
term frequency, 243
term weight, 243
thermal activity, 172
Tikhonov regularization, 286
time-dependent learning rate, 155
topographic clustering, 266
topology preserving mapping, 266
tournament selection, 173
training
  batch, 102
  data set, 200
  Manhattan, 112
  online, 102

transformation
    Householder, 282
    Jacobi, 282
    membership, 49
    membership degree, 23
    orthogonal linear, 13
    reflection, 13
    rotation, 13
    translation, 13
translation, 13
transpose, 268
trapezoidal function, 18
triangular function, 18
triangulation, 122
    Delaunay, 122
trivial solution, 46

ultra metric, 84
uniform distribution initialization, 74
unsupervised classification, 2
upper triangular matrix, 278

validation data set, 200
Vapnik–Chervonenkis dimension, 208
variance
    isotropic, 22
variance error, 199
VC dimension, 208
vector field, 54, 100
vector space model, 243
velocity distribution, 173
volume, 21
Voronoi cell, 122
Voronoi diagram, 122

Ward's method, 85
weight, 20
weight regularization, 187, 238
weighting exponent, 51, 238
window rule, 158
window width, 82
wine data, 235, 239

Curriculum Vitae

Name:	Christian Borgelt
Anschrift:	Ankerstraße 3 D-39124 Magdeburg
Geburtstag:	06. Mai 1967
Geburtsort:	Bünde (Westfalen)
Familienstand:	ledig
Eltern:	Friedrich-Wilhelm Borgelt Ursula Borgelt, geborene Kaemper Im Obrock 77, D-32278 Kirchlengern
Geschwister:	keine
Schulbildung:	1973 – 1977 Besuch der Grundschule Südlengern 1977 – 1986 Besuch des Gymnasiums am Markt in Bünde (Westfalen)
Schulabschluß:	Allgemeine Hochschulreife (Durchschnittsnote 1.9)
Wehrdienst:	01. Juni 1986 – 30. September 1987 in Munster
Studium:	1987 – 1995 an der Technischen Universität Carolo-Wilhelmina zu Braunschweig
Oktober 1987:	Immatrikulation für den Studiengang Informatik
September 1989:	Vordiplom im Studiengang Informatik
Oktober 1989:	Immatrikulation für den Studiengang Physik
September 1992:	Vordiplom im Studiengang Physik
April 1995:	Diplom im Studiengang Informatik (Gesamtnote: mit Auszeichnung)
Oktober 1995:	Auszeichnung für hervorragende Leistungen im Rahmen der Diplomprüfung im Studiengang Informatik
Schwerpunkte:	Künstliche Intelligenz/Wissensverarbeitung Datenbanksysteme Technische Informatik (integrierte Schaltungen) Theoretische Informatik
Anwendungsfach:	Physik

Weitere Gebiete: Mathematik (Wahrscheinlichkeitstheorie, Numerik, Differentialgleichungen)
Psychologie (Kognitions- und Wahrnehmungspsychologie, Problemlösen)
Philosophie (Wissenschafts- und Erkenntnistheorie, Philosophie der Mathematik, philosophische Probleme der modernen Physik etc.)

Sonstiges: Mitarbeit an der Übersetzung des Buches:
Kruse, Gebhardt, Klawonn: "Fuzzy Systeme",
Teubner, Stuttgart 1993, ins Englische:
"Foundations of Fuzzy Systems",
J. Wiley & Sons, Chichester, United Kingdom 1994.

Promotion zum Doktoringenieur (Dr.-Ing.) mit der Dissertation
"Data Mining with Graphical Models"
an der Otto-von-Guericke-Universität Magdeburg.

Eingereicht: 01. März 2000
Verteidigt: 04. Juli 2000

Gutachter: Prof. Dr. Rudolf Kruse
Prof. Dr. Stefan Wrobel
Prof. Dr. Hans-Joachim Lenz

Note: Summa cum laude

Diese Arbeit wurde mit dem Dissertationspreis 2000 der Fakultät für Informatik der Otto-von-Guericke-Universität Magdeburg ausgezeichnet.

Berufstätigkeit: In den Jahren 1991 bis 1993 neben dem Studium Mitarbeit an Softwareprojekten der Firma Lineas Informationstechnik GmbH, Braunschweig.

01. Juli 1995 – 30. September 1995
angestellt als System-Analytiker bei der Lineas Informationstechnik GmbH, Braunschweig.

01. Oktober 1995 – 30. September 1996
tätig als wissenschaftlicher Angestellter der Technischen Universität Braunschweig im Forschungszentrum der Daimler-Benz AG in Ulm (Arbeitsgruppe maschinelles Lernen und Data Mining, Prof. Dr. Gholamreza Nakhaeizadeh).

01. Oktober 1996 – 30. September 2000
wissenschaftlicher Mitarbeiter am Institut für Wissens- und Sprachverarbeitung der Otto-von-Guericke-Universität Magdeburg (am Lehrstuhl für Neuronale Netze und Fuzzy-Systeme, Prof. Dr. Rudolf Kruse).

seit dem 01. Oktober 2000
wissenschaftlicher Assistent (C1) am Institut für Wissens- und Sprachverarbeitung der Otto-von-Guericke-Universität Magdeburg (am Lehrstuhl für Neuronale Netze und Fuzzy-Systeme, Prof. Dr. Rudolf Kruse).

Bücher:

C. Borgelt and R. Kruse.
Graphical Models —
Methods for Data Analysis and Mining.
J. Wiley & Sons, Chichester, United Kingdom 2002

D. Nauck, C. Borgelt, F. Klawonn, and R. Kruse.
Neuro-Fuzzy-Systeme —
Von den Grundlagen künstlicher neuronaler Netze zur Kopplung mit Fuzzy-Systemen.
Vieweg, Braunschweig/Wiesbaden, Germany 2003

M.R. Berthold, H.-J. Lenz, E. Bradley, R. Kruse, and C. Borgelt (eds.).
Advances in Intelligent Data Analysis V — Proc. 5th Int. Symposium on Intelligent Data Analysis (IDA2003, Berlin, Germany).
Springer-Verlag, Heidelberg, Germany 2003

Magdeburg, den 2. November 2005