Mixture models for clustering and dimension reduction

Jakob Verbeek

Mixture Models for Clustering and Dimension Reduction

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor

aan de Universiteit van Amsterdam

op gezag van de Rector Magnificus

prof.mr. P.F. van der Heijden

ten overstaan van een door het college voor promoties ingestelde

commissie, in het openbaar te verdedigen in de Aula der Universiteit

op woensdag 8 december 2004, te 11:00 uur

door

Jakob Jozef Verbeek

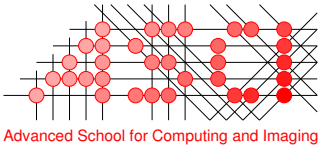
geboren te Hoevelaken

Promotiecommissie:

Promotor:	Prof. dr. ir. F.C.A. Groen
Co-promotores:	Dr. ir. B.J.A. Kröse Dr. N. Vlassis
Overige leden:	Dr. ir. R.P.W. Duin Prof. dr. C.C.A.M. Gielen Prof. E. Oja Prof. dr. M. de Rijke Prof. dr. ir. A.W.M. Smeulders

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

This work was supported by the Technology Foundation STW (project nr. AIF4997) applied science division of NWO and the technology program of the Dutch Ministry of Economic Affairs.



This work was carried out in the ASCI graduate school ASCI.
ASCI dissertation series number 107.

ISBN 90 5776 125 4

CONTENTS

ACKNOWLEDGEMENTS

While working the last four years on the subject of this thesis I enjoyed the support of my colleagues and other people. Of course, the most direct support came from my supervisors. Ben Kröse played a crucial role: he brought the PhD position to my attention which I filled during the last years. In the first two years especially Nikos invested a lot of time in me, he pointed me to many of the relevant research papers in the field that was quite new to me at the time. I much enjoyed and appreciated this, thanks! In the summer of 2003 I visited the machine learning group at the University of Toronto, I would like to thank Sam Roweis for his hospitality, the time we worked together and making the visit possible in the first place.

I would also like to thank Gabi Peters of the Graphical Systems group at the University Dortmund for sharing the image database used in Section 5.3. I would like to thank Jan Nunnink for his efforts during his MSc project on the material presented in Section 3.4.1. I would also like to thank Aristidis Likas and Josep Porta for collaborating. Furthermore, I would like to thank in particular several colleagues who read draft versions of parts of this thesis and helped to spot many typos: Andrew Bagdanov (who read the complete thesis), Bram Bakker, Stephan ten Hagen, Jan Nunnink, Dick de Ridder (thanks for the layout template), Matthijs Spaan and Wojciech Zajdel. I would also like to thank the people of Emergentia for the Sunday afternoons filled with a pleasant mix of discussing a nice paper and having dinner together. Marten Rozenbeek designed the cover, for which I'm very grateful. Finally, I would like to thank Kristel for supporting me throughout and helping me to get rid of the occasional doom-thought when working on this thesis. Thanks to all!

1

INTRODUCTION

In many current information processing systems the measurements are very high dimensional: each measurement comprises many variables. Examples include:

- Systems which use a camera as a sensor, delivering images consisting of several thousands or millions of pixels.
- Systems which process text documents, frequently represented as a vector indicating for each of several thousands of words in a dictionary how often it occurs in the document.
- Systems which analyze sound by measuring the energy in several hundreds of frequency bands every few milliseconds.

By measuring many different variables a very rich representation is obtained, that can be used for a wide variety of tasks. For example, images can be used to recognize the different digits of a postal code written on an envelope, or to recognize a person from an image of his face.

Information processing tasks where high dimensional data sets are common include classification, regression and data visualization. In classification, the goal is to predict the class of a new measurement on the basis of *supervised* examples. For example, we may want to predict which digit is displayed in an image: does the image belong to class $0, 1, \ldots, 8$ or 9? The supervised examples then consist of images of digits together with a label indicating which digit is displayed. The supervised examples are used to estimate a function that can be used to map new images from an unknown class to a prediction of the class label. Regression is similar to classification, but the goal is to predict some continuous quantity from the measurement rather than a discrete class label. Visualization techniques are used to produce an insightful graphical display of the data to assist a user in interpreting the data. For example, the results of image database queries can be presented using a two dimensional visualization that displays similar images near to each other. In this manner a user quickly gets an idea of what kind of images were found.

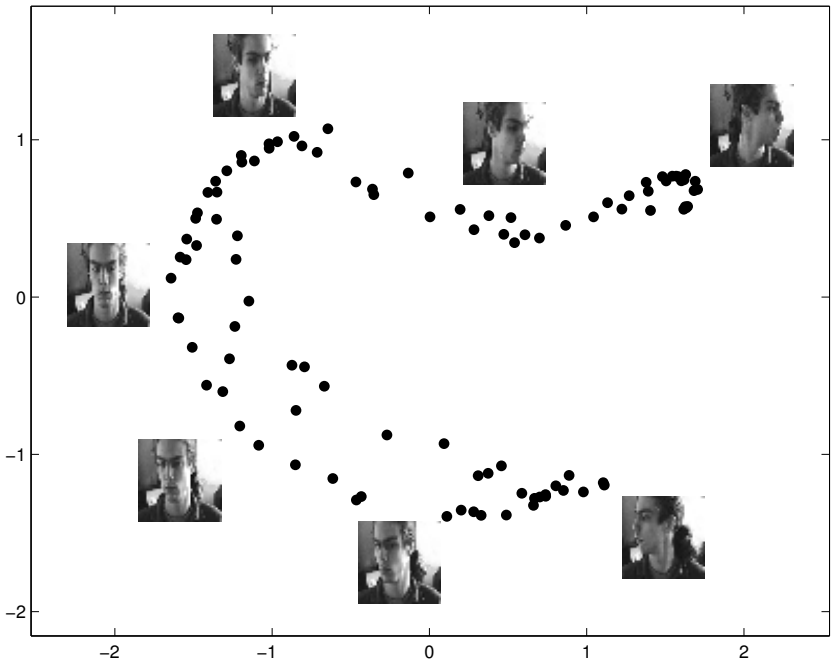


Figure 1.1: The points are given by a linear projection of the 1600 pixel values of images of a face. The images trace-out a curve along which the pose of the face gradually changes. For some points the corresponding image is plotted next to it.

Degrees of freedom and clusters. The high dimensional data can be thought of as a set of points in a high dimensional *data space*; the value of a measured variable gives the coordinate on a corresponding dimension of this space. In many applications where such high dimensional data is processed, the data is not distributed evenly over the data space but exhibits some structure.

Often the dimensionality of the data space is much larger than the number of *degrees of freedom* of the process from which the measurements are obtained. The degrees of freedom are the manners in which the process that is monitored can vary. Since the measurements are in a sense a *function* of the degrees of freedom, the number of degrees of freedom also limits the number of ways in which the measurements can vary. For example, consider $40 \times 40 = 1600$ pixel gray-scale images of a face that looks in different directions, as depicted in Fig. 1.1. We may consider each image as a 1600 dimensional measurement, one dimension for the gray value of each pixel. However, there is only one degree of freedom since the face changes only by turning from left to right. If the direction in which the face looks changes, the values of many of the 1600 pixels will change. However, these changes will be highly correlated. Apart from measuring many correlated variables, irrelevant variables may be measured as well: variables that are not dependent on the degrees of freedom. If the sensor is noisy, the value of an irrelevant variable does vary over the different measurements.

If small changes in the degrees of freedom also lead to small changes in the measured variables, then the value of the measured variables is a *smooth* function of the degrees of freedom. An illustration is given in Fig. 1.1, where we plotted images of a face as points in a two dimensional space, each coordinate is given by a linear combination of the original pixel values. We see that the points are roughly distributed around a smooth curve. By plotting for several points the corresponding image next to it, we see that indeed the direction in which the face looks changes gradually along the curve.

If the number of degrees of freedom is larger than one, then the data lies on or near surfaces —or in general *manifolds*— rather than curves. The degrees of freedom are referred to as *latent variables* underlying the data, and the number of degrees of freedom is referred to as the *intrinsic dimensionality* of the data.

In addition, the data may exhibit a clustered structure: the data forms several disconnected clouds of points in the data space. For example, consider a collection of text documents which contains articles about soccer and articles about religion. Suppose that the documents are represented as suggested above, and that very common words like 'a', 'the', 'have', 'is', etc., have been removed from the dictionary. It may be expected that articles about the same topic have a lot of words in common. On the other hand, two articles about different topics are expected to use rather different words. Thus, the documents of the two topics form two separated *clusters* in the data space. There are also situations where the data exhibits both a clustered structure and the data within each cluster is distributed near a low dimensional manifold.

The curse of dimensionality. It is well known that for classification and regression the amount of supervised examples required to reliably estimate a function with a given accuracy grows exponentially with the dimensionality of the data. Therefore, already for several tens of dimensions the number of required supervised examples becomes impractically large. This effect was termed 'the curse of dimensionality' by Bellman (Bellman, 1961). To illustrate this, consider a rather simplistic approach to classification and regression where we partition each input dimension into m cells. For D input dimensions, the total number of cells is then m^D, and to estimate the function output of each cell we would need at least m^D supervised examples.

The set of classification or regression functions that is considered for a specific task plays a crucial role here. If the set of functions is very small, then, regardless of the data dimensionality, chances are high that already with relatively few supervised examples we can determine which function gives the best predictions for future data. As the set of functions becomes larger, more supervised examples are needed to identify the best function among the many candidates. When processing high dimensional data, the absence of prior knowledge to reduce the set of potentially useful functions often leads to a very large set of functions that is considered.

High dimensional data is also problematic for data visualization, since the number of

variables that can be displayed is inherently limited to two or three. With colors and temporal effects, i.e. showing a movie rather than a still plot, it is possible to somewhat increase the number of variables that can be displayed. However, the total number of variables that can be displayed remains limited to about six.

Finding the relevant information. To avoid the curse of dimensionality and to visualize high dimensional data, there is a need for methods which are able to detect clusters and to identify the manifolds on which the data resides. The data can then be represented in terms of cluster membership and/or low dimensional coordinates on these manifolds. In a sense, such methods find the relevant information in an overwhelming amount of variables. In Chapter 2 we give a review of different types of clustering and dimension reduction methods that have been proposed.

In most classification and regression applications it is time and/or money consuming to generate supervised examples: for a set of measurements the class label or regression variable has to be determined manually. Therefore the number of supervised examples is often limited. Unsupervised examples —lacking the class label or regression variable— on the other hand, are often much easier to acquire. For example, when the goal is to classify email messages as 'spam' or 'not-spam', it is easy to acquire thousands or more email messages but quite cumbersome to determine the class label for all these messages. Therefore, it is attractive to apply clustering or dimension reduction techniques on a relatively large set of unsupervised data to recover a compact representation. A classification or regression function can then be estimated from a low dimensional description of a smaller set of supervised data and / or from the supervised data in each cluster separately. In this thesis we mainly focus on such unsupervised methods for clustering and dimension reduction.

Among the best known clustering approaches are the k-means algorithm and the EM algorithm to estimate Gaussian mixture densities. Both algorithms iterate two steps, and both algorithms are guaranteed to converge and to increase some performance measure after each iteration. The main drawback of these algorithms is that the resulting clustering depends strongly on the initialization of the algorithm and is not guaranteed to maximize the performance criterion. In Section 3.1 we introduce mixture densities and the EM algorithm. Then in Section 3.2 and Section 3.3 we consider how to overcome the drawbacks of the k-means algorithm and the EM algorithm for Gaussian mixtures. In Section 3.4 we consider how the EM algorithm can be accelerated when applied to data sets containing many points.

Dimension reduction methods can be divided into methods which are limited to finding *linear* manifolds in the data space (lines, planes, etc.), and methods which also find *non-linear* manifolds. The self-organizing map, introduced by Kohonen in the early 1980's, is a non-linear dimension reduction method that has been used in many practical applications. Nevertheless, there are several problematic issues with self-organizing

maps. First, Kohonen's parameter estimation algorithm is not guaranteed to converge. Second, the self-organizing map was originally designed for dimension reduction of real valued data. Extensions for data that are not specified as real numbers have been proposed, but were rather ad-hoc and not derived from a general principle. The same holds when there are data points with missing values; i.e. data points for which the value of some variables is unknown. In Chapter 4 we present a dimension reduction technique based on mixture densities, similar to Kohonen's self-organizing map, that resolves the mentioned problems.

In some applications it is not only desirable to have a mapping from the high dimensional data space to a low dimensional representation, but also a mapping from the low dimensional representation to the original high dimensional data space. Such a mapping allows us to *reconstruct* the high dimensional data from the low dimensional representation. In the example of facial images given above, such a mapping would allow us to generate the different facial images by specifying a single number: their location along the curve. Linear dimension reduction methods, such as principal component analysis, are widely used for data compression and reconstruction. Although linear methods can be implemented very efficiently and provide a smooth two-way mapping, they are limited to map to and from linear manifolds. Most non-linear dimension reduction methods do not provide a smooth two-way mapping or lack an efficient algorithm for parameter estimation.

Several methods exist that combine clustering and linear dimension reduction. Such methods have been applied to data which lies on or near a non-linear manifold in the data space, i.e. like the images in Fig. 1.1 which lie along a non-linear curve. The data is clustered in such a manner that the data within each cluster lies close to a linear manifold; the curve is approximated by several linear segments. These methods find a separate low dimensional representation for the data in each cluster by projecting the data onto the linear subspace associated with the cluster. Thus, there is no single —global— low dimensional representation for the complete data set. In Chapter 5 we explore a probabilistic approach to combine the several linear low dimensional representations into a global non-linear one. This approach provides a smooth two-way mapping and parameters can be estimated with an EM-like algorithm.

In Chapter 6, we summarize our conclusions from the preceding chapters and discuss directions for further research.

2

A REVIEW OF CLUSTERING AND DIMENSION REDUCTION TECHNIQUES

In this chapter we introduce clustering and dimension reduction, which will be the main topics of the rest of this thesis. We also give an overview of different types of techniques that have been developed for these tasks. In the first section we consider clustering techniques, after which we proceed to dimension reduction methods in Section 2.2.

2.1 Clustering

Clustering problems arise in many fields of (computer) science, in particular in computer vision, pattern recognition, data mining and machine learning. The clustering problem (Jain et al., 1999) is the problem of dividing a given set $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ of N data points into several non-overlapping homogenous groups. Each such group or cluster should contain similar data items and data items from different groups should not be similar. We refer to a clustering in k groups as a k-clustering.

Clustering techniques can be useful in explorative data analysis, e.g. a sales-company might identify different types of customers based on a clustering of data about the purchases that customers made. Clustering is also used as a preprocessing step for other tasks. For example, in data visualization the data of widely separated clusters may be visualized in a separate displays (Bishop and Tipping, 1998).

Many different approaches to the clustering problem have been developed. Some operate on data represented by their coordinates in a feature space and others operate on a matrix of pairwise similarities between data points. To give a, necessarily limited, overview of the different types of methods, we categorize them in three groups:

1. **Hierarchical clustering methods.** These produce a hierarchy of clusters for the data. The first level of the hierarchy contains all data and at each subsequent level

of the hierarchy, one of the clusters of the previous level is split in two. The last level contains all data in individual clusters. The hierarchy is based on pairwise similarities between data points and is constructed either top-down or bottom-up.

2. **Partitional clustering methods.** These produce a single clustering with a fixed and (often) specified number of clusters. Most partitional clustering algorithms do not operate on the basis of pairwise similarities, but with data represented in some feature space. Typically, these methods start with an initial k-clustering and apply an iterative algorithm to improve upon the initial clustering according to some criterion. Most partitional clustering methods make, sometimes implicitly, assumptions on the distribution of data within each cluster.

3. **Spectral clustering methods.** These operate on a matrix with pairwise similarities between the data points. The optimal clustering is defined as the clustering that minimizes the 'normalized cut' criterion, that depends on the size of the clusters and the total sum of the similarities between points that are assigned to different clusters. Unfortunately, finding the clustering that minimizes the normalized cut is an NP-complete problem. However, a relaxation of this optimization problem can be efficiently solved, and the solution is given by an eigenvector of the normalized similarity matrix. The solution of the relaxed problem is then further processed to find an approximate solution for the original problem. The term 'spectral clustering' refers to the eigenspectrum of the normalized similarity matrix which can be used to assess the number of clusters in the data. Spectral methods are used both to find hierarchical clusterings and k-clusterings for a given k. We treat them separately since their working is quite different from the other approaches.

In following three sections we describe the three different types of clustering methods in more detail, and we compare them in Section 2.1.4.

2.1.1 Hierarchical clustering

A hierarchy of clusters can be represented as a tree; the root node contains all data and the two children of each node contain disjoint subsets of the data contained in the parent. The leaves of the tree contain the individual data points. A hierarchy of clusters, rather than a 'flat' clustering in k clusters, is desired in some applications. For example, consider hierarchical clustering of newspaper articles: in the top-levels general topics are found, such as politics, financial news and sports. At lower levels the sports cluster might be further subdivided into articles on individual sports. Such a hierarchy of clusters enables a user to quickly find articles of interest. This is because at each level of the tree the user can discard large clusters of uninteresting articles and explore further only the more promising clusters. See e.g. (Zhao and Karypis, 2002; Blei et al., 2004) for work on hierarchical clustering of documents.

Hierarchical methods (Johnson, 1967) are either agglomerative or divisive. Agglomer-

ative clustering is a 'bottom-up' approach; it starts by considering each data point as a single cluster and proceeds by iteratively merging the two most similar clusters. Divisive clustering algorithms are 'top-down' and start with one cluster containing all the data and proceed by iteratively splitting the clusters into two disjoint subsets.

The agglomerative and divisive methods differ greatly in their computational demands. Suppose we have N data points, then the number of possible mergers an agglomerative algorithm has to consider in the first step is $N(N-1)/2$ (any two data points can be merged to form a cluster of size two). In total $N-1$ mergers have to be made to construct the complete cluster hierarchy, and in total $O(N^3)$ possible mergers have to be considered. The number of possible splits a divisive algorithm has to consider in the first step alone is already $O(2^N)$. Therefore, divisive algorithms that consider every possible split to find the optimal split will be intractable already for moderately sized data sets. Agglomerative algorithms scale much better: 'only' cubic in the number of data points. Because of the intractability of divisive methods we will from now on focus only on agglomerative methods.

Agglomerative methods differ in the (dis)similarity measure that is used. Most dissimilarity measures d_{AB} between clusters A and B are defined in terms of a dissimilarity measure d_{ij} between elements of the clusters. Well-known measures are:

$$d_{AB} = \min_{i \in A, j \in B} d_{ij} \quad \text{(single link)}, \tag{2.1}$$

$$d_{AB} = \max_{i \in A, j \in B} d_{ij} \quad \text{(complete link)}. \tag{2.2}$$

The single-link measure is based on the smallest distance between pairs of members of A and B; clusters are similar if there is at least a 'single link' of similar data items between them. The complete-link measure uses the largest distance between pairs of members, thus clusters are similar if all pairs $(i \in A, j \in B)$ are similar. The difference between the single link and complete link method are illustrated in Fig. 2.1. There the result of a hierarchical agglomerative clustering is given, using both single and complete link. Only the 2-clustering of the hierarchy is indicated in the plot for clarity. In this example the dissimilarity between data points is the squared Euclidean distance between the data points: $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^2$. Arguably, the 2-clustering found in this data using single-link is better: it identifies the two horizontal bands.

Another distance measure between clusters for an agglomerative clustering is used in Ward's sum-of-squares method (Ward, 1963). Here, the aim is to minimize the total sum of squared distances between data points and the mean of their associated cluster. For a k-clustering into clusters $A_1, \ldots, A_k$, we denote by n_i the number of data points in cluster A_i and write the mean of the data in each cluster as:

$$\boldsymbol{\mu}_i = \frac{1}{n_i} \sum_{n \in A_i} \mathbf{x}_n, \tag{2.3}$$

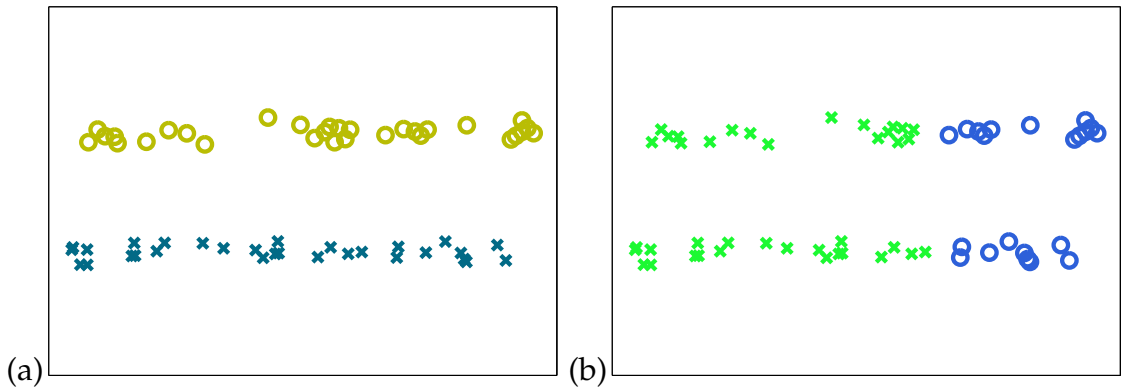


Figure 2.1: Agglomerative clustering of 60 points in a 2-dimensional feature space, using (a) single link and (b) complete link distance. Recovered clusters are indicated by plotting their members with 'x' and 'o' respectively.

The sum-of-squares error is then given by:

$$E_{SoS} = \sum_{i=1}^{k} \sum_{n \in A_i} \|\mathbf{x}_n - \boldsymbol{\mu}_i\|^2. \tag{2.4}$$

An agglomerative clustering algorithm based on the sum-of-squares error E_{SoS} should use as the dissimilarity between clusters i and j the increase in the error incurred if we merge the i-th and the j-th cluster. This increase only depends on the data in these clusters and can be written as:

$$d_{A_i A_j} = \sum_{n \in (A_i \cup A_j)} \|\mathbf{x}_n - \boldsymbol{\mu}_{(A_i \cup A_j)}\|^2 - \sum_{n \in A_i} \|\mathbf{x}_n - \boldsymbol{\mu}_i\|^2 - \sum_{n \in A_j} \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2 \tag{2.5}$$

$$= n_i \|\boldsymbol{\mu}_{A_i}\|^2 + n_j \|\boldsymbol{\mu}_{A_j}\|^2 - n_{(A_i \cup A_j)} \|\boldsymbol{\mu}_{(A_i \cup A_j)}\|^2. \tag{2.6}$$

However, the optimal $(k-1)$-clustering is not necessarily obtained by merging two clusters of the optimal k-clustering. Therefore the agglomerative method can yield sub-optimal k-clusterings for the sum-of-squares criterion (Webb, 2002).

For many agglomerative clustering algorithms the cluster dissimilarities that are obtained after a merge can be expressed as a function of the dissimilarities before the merge. Thus after merging clusters i and j into a cluster $(i \cup j)$ the dissimilarities $d_{i \cup j,k}$ between the new cluster and another cluster k, can be expressed in terms of d_{ij}, d_{ik} and d_{jk} and dissimilarities between clusters not involved in the merge remain unchanged. This is for example the case for the single-link, complete-link and sum-of-squares methods, see e.g. (Webb, 2002) for the update rules of the dissimilarity measures for these and a number of other agglomerative clustering algorithms. If the dissimilarities can be updated in this way the complexity of the algorithm drops to $O(N^2 \log N)$, using a sorted list to store the distances between the clusters. If after every merge we have to

recompute all dissimilarities between the clusters the complexity is at least $O(N^3)$, since in each step i $(i = 0, \ldots, N-1)$ we need to compute $O((N-i)^2)$ distances.

For other agglomerative clustering methods, for example methods based on seeking the mode of a density estimate (Cheng, 1995; Leung et al., 2000), distances between clusters cannot be specified and the hierarchical clustering is obtained as the fixed point of an algorithmic process applied to the data.

2.1.2 Partitional clustering

Partitional methods cluster the data in a specified number of groups. Their main attraction over the hierarchical methods is that partitional algorithms are generally much more efficient. The main drawback is that assumptions on the shape of the clusters have to be made in advance. Also a desired number of clusters has to be specified, which may be known in some applications but not in others. Some work has been done on estimating the number of clusters from the data, see e.g. (Pelleg and Moore, 2000; Rasmussen, 2000; Fred and Jain, 2002). In general however this issue remains unresolved.

The k-means algorithm (Gersho and Gray, 1992) is one of the most frequently applied partitional clustering algorithms. It is also known under a variety of other names: Generalized Lloyd algorithm, Lloyd Max algorithm, Forgy algorithm, or Linde-Buzo-Gray algorithm. We consider this algorithm in detail in Section 3.3. Here we will briefly explain the algorithm, mention some of its properties, and consider how the principle of the algorithm can be used for different error-functions.

Given an initial k-clustering the k-means algorithm adapts this clustering to reduce the sum-of-squares criterion. The k-means algorithm alternates between two steps to improve a given k-clustering. The idea is to treat E_{SoS} in (2.4) as a function of both the cluster centers $\boldsymbol{\mu}_i$ and the assignment of data points to the clusters. In the first step we fix the assignment of data to clusters and minimize the error with respect to the cluster means $\boldsymbol{\mu}_i$. The optimal cluster centers are given by the mean of the data assigned to each cluster. After this step the error (as function of both the assignments and cluster means) equals E_{SoS} as defined in (2.3) and (2.4). In the second step we fix the cluster means $\boldsymbol{\mu}_i$ and minimize the error with respect to the assignment of the data to the clusters. This amounts to assigning every point $\mathbf{x}_n$ to the cluster $i = \arg\min_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2$. After these two steps we obtain a new k-clustering for which E_{SoS} cannot be larger than for the previous k-clustering.

The algorithm is guaranteed to terminate after a finite number of steps since (i) there are a finite number of k-clusterings and (ii) each step can not increase the sum-of-squares error. However, the k-means algorithm does not necessarily terminate with the k-clustering yielding the minimum sum-of-squares. The resulting k-clustering depends strongly on the initial clustering. Therefore, in practice the k-means algorithm is started from many (random) initial k-clusterings, and the best final clustering is retained.

The k-means algorithm is based on an implicit assumption that the data exhibits compact spherical clusters. For the example data set used in Fig. 2.1, the assumption of compact clusters is clearly violated since the 'true' clusters (the two horizontal bands) are actually quite elongated. For this example data set the k-means algorithm will find a 2-clustering similar to the clustering depicted in panel (b) found using the agglomerative complete-link algorithm.

The idea of the k-means algorithm can be used for other error functions than sum-of-squares. For example, (Dhillon et al., 2002) proposed a similar algorithm to cluster probability distributions based on an error function that sums Kullback-Leibler divergences rather than squared Euclidean distances. In (Dhillon et al., 2002) the authors consider clustering of documents based on the occurrence frequency of words in them. Each document n is represented by a distribution $p_n(w)$ over the words w in the lexicon. Just as above, each cluster is also represented in the same space as the data items, thus in this case cluster i is represented by a distribution $q_i(w)$ over words. In order to cluster the documents, the authors minimize the sum over all documents of the Kullback-Leibler (KL) divergence $\mathcal{D}(p_n \| q_i)$ between the distribution p_n over words of document n and the distribution over words of the cluster i associated with document n. The KL divergence $\mathcal{D}(p\|q)$ measures how well distribution q matches distribution p and is defined as:

$$\mathcal{D}(p\|q) = \sum_w p(w) \log \frac{p(w)}{q(w)}. \tag{2.7}$$

The resulting clustering algorithm is completely analogous to the standard k-means algorithm. The cluster distributions q_i are set to the average distribution of assigned documents, i.e. $q_i(w) = \frac{1}{n_i} \sum_{n \in A_i} p_n(w)$. The only difference is that we now assign a document to the cluster with minimum Kullback-Leibler divergence rather than the one with minimum Euclidean distance. See (Saul and Pereira, 1997) for a similar clustering approach to estimate bigram language models.

Probabilistic mixture distributions are another important class of partitional clustering methods. A mixture distribution is a distribution that is a weighted sum (weights are positive and sum to one) of several component distributions. The components are restricted to some parametric family. The fit of the model to the data is defined as the likelihood of the data according to the model. Often the expectation-maximization (EM) algorithm is used to adjust the parameters such that the likelihood is increased (Dempster et al., 1977). The EM algorithm is reminiscent of the k-means algorithm: it also iterates between a step assigning data to clusters and a step that optimizes parameters that characterize the clusters. However, whereas the k-means algorithm assigns each data point to a *single* cluster, the EM algorithm uses a 'soft' assignment: that is, each data point is assigned to each cluster, but in a weighted manner.

The EM algorithm shares two important properties with the k-means algorithm: (i) monotone decrease of the error: each iteration of EM algorithm is guaranteed not to decrease the data likelihood, and (ii) local optima: if the EM algorithm terminates (the

soft assignment does not change between two iterations), it is *not* guaranteed to have found the parameters yielding the maximum possible likelihood. Such fixed points that do not yield maximum likelihood can be shown to be local maxima of the likelihood as function of the parameters of the mixture model (Neal and Hinton, 1998). In practice, the standard approach to reduce the risk of finding only a poor locally optimal solution is to start the EM algorithm with a number of different initial parameters. Finally, the local optimum yielding the highest likelihood is retained.

Mixture models and the EM algorithm are tools we use in the rest of this thesis, both will be introduced in detail in Chapter 3. In the same chapter we consider alternative techniques to avoid poor local optima when clustering using Gaussian mixture distributions (Section 3.2) and the k-means algorithm (Section 3.3). Both techniques use a greedy strategy that 'builds' a k-clustering step by step. After initialization, which assigns all data to a single cluster, two steps are iterated: (i) adding a new cluster to the existing $(k-1)$-clustering to obtain a k-clustering and (ii) improving the current clustering with the k-means or EM algorithm. We compare this greedy approach to the standard method that starts the algorithms from many different initial parameters. Experimentally we show the greedy approach to yield considerably improved clusterings.

2.1.3 Spectral clustering

A relatively recent approach based on pairwise similarities is spectral clustering (Scott and Longuet-Higgins, 1990; Weiss, 1999), which draws on results of spectral graph theory (Chung, 1997). Spectral methods are attractive because they (i) make less severe assumptions on the shape of the clusters than partitional algorithms and (ii) can be very fast, depending on the sparsity of the similarity matrix. Furthermore, implementation of most spectral clustering algorithms is quite easy since the main component is a procedure to find a few eigenvectors of a matrix: this is a well studied problem for which highly optimized implementations are available.

Below we follow (Shi and Malik, 2000) in the presentation of the 2-clustering problem. Extensions to general k-clustering exist, but we will not treat them here, see e.g. (Meilă and Shi, 2001; Ng et al., 2002). Spectral clustering is based on viewing the data as a fully connected graph $G = (V, E)$ with a node $v_i \in V$ for each data point $\mathbf{x}_i$. With each edge $e_{ij} \in E$ between node v_i and v_j we associate a weight $w_{ij} \geq 0$ representing the similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$, with a large value indicating great similarity. The weights w_{ij} are collected in the symmetric weight matrix $\mathbf{W}$ with an associated diagonal 'degree matrix' $\mathbf{D}$ with $d_i = \sum_j w_{ij}$ in the i-th diagonal element and zero in all off-diagonal elements.

With a partitioning of V in disjoint subsets A and B we associate a 'cut value' $cut(A, B)$:

$$cut(A, B) = \sum_{i \in A, j \in B} w_{ij}, \tag{2.8}$$

which is the sum of all edge weights across the clustering, i.e. edges from A to B. If the edge weights are binary, one for similar points and zero for dissimilar points, the cut value counts the pairs of similar points that are separated by the clustering. At first it might seem a good idea to define an optimal clustering as one that minimizes the cut value. However, the cut value has the undesired property that it gives very low values for when the partition separates a single point or just a few points from the rest, i.e. it favors unbalanced clusterings.

The cut value fails to take into account the size of the clusters, which seems to be an important notion to define the quality of a clustering. To formalize the notion of size, we associate with a subset of the vertices $A \subseteq V$ a 'volume' $vol(A)$, which is the sum of all edge weights connected to points in A:

$$vol(A) = \sum_{i \in A, j \in V} w_{ij} = \sum_{i \in A} d_i. \tag{2.9}$$

The normalized cut (Ncut) measure to evaluate a clustering into sets A and B does take into account the size of the clusters and is defined as:

$$Ncut(A, B) = \frac{cut(A, B)}{vol(A)} + \frac{cut(A, B)}{vol(B)}. \tag{2.10}$$

Thus, we add for each cluster the ratio of (i) the total weight of edges that cross the cluster boundary and (ii) the total weight of edges emanating from points in the cluster. Due to normalization, Ncut favors clustering into clusters of similar size with a small cut value. We can now define an optimal clustering as one that minimizes Ncut.

To analyze the minimization of Ncut, we assign to each data point $\mathbf{x}_i$ a value y_i:

$$y_i = \begin{cases} 2 & \text{if} \quad \mathbf{x}_i \in A \\ -2vol(A)/vol(B) & \text{if} \quad \mathbf{x}_i \in B \end{cases} \tag{2.11}$$

We can now write[1] Ncut in terms of the y_i, which we collect in a vector $\mathbf{y}$ of length N:

$$Ncut(A, B) = \frac{\mathbf{y}^\top (\mathbf{D} - \mathbf{W})\mathbf{y}}{\mathbf{y}^\top \mathbf{D}\mathbf{y}} = \frac{\sum_{i,j} w_{ij}(y_i - y_j)^2}{\sum_i d_i y_i^2}. \tag{2.12}$$

The minimization of Ncut is an NP-complete problem. However, if we relax the constraint that the y_i can take only two values and allow the y_i to take any real value, then stationary points of (2.12) can be found as generalized eigenvectors:

$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}, \tag{2.13}$$

where λ is the value of the relaxed Ncut criterion corresponding to $\mathbf{y}$. The vector $\mathbf{y} = \mathbf{1}$ is easily verified to be an eigenvector with eigenvalue zero, corresponding to a degenerate

[1] See (Shi and Malik, 2000) for the derivation which we omit here.

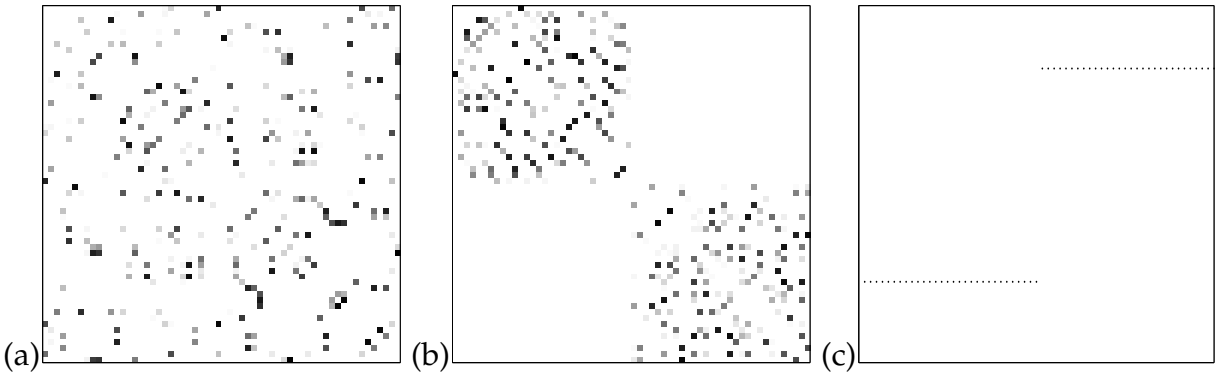


Figure 2.2: (a) Original weight matrix, larger values are plotted darker. (b) Weight permuted to reveal clusters. (c) Entries in the second eigenvector, ordered as in panel (b).

cut which collects all points in a single cluster. Thus, neglecting this degenerate solution, the minimizer of the relaxed Ncut problem is given by the eigenvector corresponding to the second smallest eigenvalue.

A solution that minimizes the relaxed Ncut criterion assigns to each data point $\mathbf{x}_i$ a real value $y_i \in \mathbb{R}$ and not just either one of two possible values. Therefore such a solution does not directly provide a clustering. However, in some cases it can be expected that all the y_i are close to either one of just two values. Suppose we order the data points such that all points of the second cluster have a larger index than points of the first cluster. Then, if the weight matrix is near block diagonal, the second smallest eigenvector is expected to be nearly piecewise constant (see (Ng et al., 2002) for an analysis), and it will be easy to transform the eigenvector into a crisp clustering. In general, some non-trivial post-processing has to be performed on the extracted eigenvector to obtain a crisp clustering. Different post-processing methods have been proposed, see e.g. (Ng et al., 2002; Verma and Meilă, 2003) for overviews. However, often this post-processing employs clustering algorithms susceptible to local optima, e.g. k-means.[2] As such, spectral clustering methods can also be considered as a pre-processing step for other clustering techniques, i.e. a pre-processing that finds a representation for the data such that it becomes easy to cluster.

In Fig. 2.2 we illustrate spectral clustering on the same data used to illustrate agglomerative hierarchical clustering. In panel (a) the weight matrix is shown, where we used $w_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/\sigma^2)$ if $i \neq j$ and $w_{ii} = 0$. Larger values are depicted darker. The weight matrix has many entries that are close to zero and can be made sparse by setting all values, say, smaller than 10^{-6} to zero. In panel (b) we permuted rows and columns such that all points in the upper band come before those in the lower band. The matrix is near block diagonal, indicating that all weights across the bands are near zero and that the eigenvector with second smallest eigenvalue will be nearly piecewise

[2] An exception is Brand and Huang's alternating projection algorithm (Brand and Huang, 2003) that does not need any post-processing.

constant. Panel (c) shows on the vertical axis the values y_i corresponding to the data points (ordered as in panel (b)) as found in the second eigenvector, which clearly reveal the two clusters. Thus here spectral clustering yields the same clusters as agglomerative clustering using the single link measure (depicted in panel (a) of Fig. 2.1).

2.1.4 Comparison of clustering methods

In this section we have reviewed the three main categories of clustering approaches and illustrated them with popular algorithms in those categories. All three approaches have their niche of applications where a particular approach is preferred. Below we compare these approaches in terms of their scalability and the assumptions underlying them.

Scalability. The hierarchical methods are in general computationally quite demanding, for N data points the agglomerative approach takes a time that is either $O(N^2 \log N)$ or $O(N^3)$, depending on whether or not after each merge all distances between all clusters have to be computed.

Most of the partitional clustering algorithms take computation time of $O(Nk)$ for k clusters. This is the case for both the k-means algorithm and the EM algorithm for probabilistic mixture models: in the assignment step each combination of data point and cluster has to be considered to find the optimal assignments.

Spectral clustering approaches need the similarity for each of the N^2 pairs of points and thus take in principle at least N^2 time to compute the similarity matrix. Depending on the similarity measure, speed-ups might be possible. For example, one could use a matrix in which an entry (i, j) is non-zero only if $\mathbf{x}_i$ is among the q-nearest neighbors of $\mathbf{x}_j$ or vice-versa. Efficient techniques exist to find nearest neighbors among N points in time $O(N \log N)$ for fixed q (Bentley, 1980; Karger and Ruhl, 2002). Often the similarity matrix used in spectral clustering is sparse, i.e. each point has non-zero similarity to only a few others. For such sparse matrices the eigenvectors can be efficiently computed using the power-method (Horn and Johnson, 1985). The iterations of the power method take an amount of time proportional to the number of non-zero entries in the similarity matrix rather than N^2 for a dense matrix.

Assumptions on clusters. For all clustering methods the distance or similarity measure that is used plays a crucial role. The measure impacts the clusters that will be found and also determines the amount of computation needed for each distance calculation. The number of distance calculations that have to be performed depends on the clustering algorithm that is used.

Probabilistic mixture models have the advantage that assumptions on the cluster shape are made explicit by assuming the distribution of data within a cluster to be in a para-

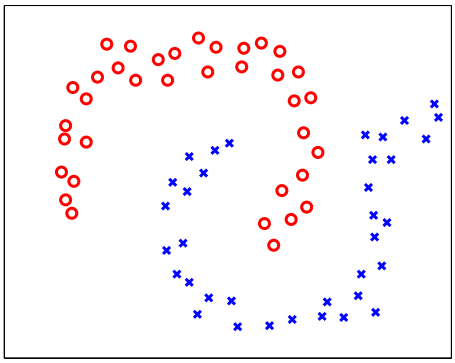


Figure 2.3: Data that is intuitively easy to cluster but the data distribution in the clusters does not correspond to some parametric class.

metric class of distributions. The assumptions in other clustering methods are often less explicit. In the rest of this thesis we will mainly focus on probabilistic mixture models and their applications in dimension reduction techniques.

Although the assumptions in mixture models are clear, they are often incorrect. Fig. 2.3 gives an example of data that exhibits two clusters which are hard to discover using mixture models since the data distribution in each cluster does not belong to any standard parametric class of densities. However, the clusters are readily recovered with spectral or single-link agglomerative clustering. Of course, for the latter methods one needs to determine a suitable similarity measure; some interesting work has been done (Bach and Jordan, 2004) on *learning* the similarity measure on the basis of several example clusterings. Interestingly, the set of densities that can be implemented using standard component classes can be increased by mapping the data to a space of much larger dimensionality where the new dimensions are (non-linear) functions of the original variables (Wang et al., 2003). Efficient parameter estimation of mixtures in the higher dimensional space is possible using the kernel trick which is discussed in the next section.

Further reading. The review provided here is limited in scope and merely gives a flavor of the different types of clustering algorithms. More detailed and extensive overviews can be found in (Jain and Dubes, 1988; Jain et al., 1999; Verma and Meilă, 2003) and in textbooks like (Bishop, 1995; Ripley, 1996; Hastie et al., 2001; Webb, 2002).

2.2 Dimension reduction

Clustering techniques provide a compact representation of the data by mapping each data point to a *discrete cluster index*. In contrast, dimension reduction techniques (DRTs) are used to find compact representation of data by mapping each point to a lower

dimensional *continuous vector*. A representation of the data in fewer dimensions can be advantageous for further processing of the data, as we discuss below. However, the reduction of the number of dimensions should not result in loss of information relevant to the task at hand. Thus, there is a trade-off between the advantages of the reduced dimensionality and the loss of information incurred by the dimension reduction. DRTs are either 'supervised' or 'unsupervised'. Supervised techniques perform dimension reduction in a manner that allows for optimal prediction of a variable of interest, e.g. class membership, or some other response variable. Unsupervised techniques perform dimension reduction to optimally predict the original data from the representation in few dimensions or to optimally preserve some properties of the original data.

The most important applications of DRTs can be divided into three groups:

- **Visualization of high-dimensional data for explorative data analysis.** Reducing the dimension of the data to two or three dimensions enables a data analyst to plot the data on a computer screen.
- **Data compression.** Using fewer dimensions to express the data allows for more compact storage of data and transmission of data using less bandwidth.
- **Increasing efficiency and performance for subsequent data analysis.** Dimension reduction can increase the efficiency of the operation and training of automatic classifiers and regression functions, which typically have an execution time and training time at least linear in the number of dimensions of the data. Reducing the dimensionality before training a classification or regression function may also increase performance. This effect, which may be counterintuitive (why would it help to discard information?), is also known as the 'curse of dimensionality'. It occurs since in a lower dimensional space fewer parameters have to be estimated for the classification or regression function, thus with the same amount of data in a lower dimensional space the parameters can be estimated more reliably.

DRTs can be divided into techniques for 'feature extraction' and 'feature selection'. Methods differ in how the new features are derived from the original features (what is the class of functions mapping from original features to extracted features) and the criterion that is used to identify appropriate features.

Feature *selection* is concerned with finding an appropriate subset of the original features to represent the data. Different, application dependent, criteria exist to select appropriate features and an appropriate *number* of features. An example of feature selection is to find a subset of variables to be used by a classifier. Using fewer features can be advantageous in situations where only a limited amount of data is present but represented with a huge number of features. For example, in diagnostic use of gene expression data, a classifier is used to determine the disease of a patient based on the expression of thousands of genes. The classifier often has to be trained from data of only several tens of patients (Tibshirani et al., 2002). In such cases it is crucial to determine a small set of relevant variables so that reliable parameter estimates can be made. An-

other advantage of identifying a small set of relevant features for a classification system is that when the system is employed in practice only few features of the data have to be computed. For example in (Viola and Jones, 2004) an algorithm is described to detect faces in images on any possible location and over a variety of scales. The algorithm can be applied to real-time video streams of 15 frames per second at a resolution of 384×288 pixels, which demands the classification of several tens of thousands of image windows as being 'face' or 'non-face'. It is possible to perform this huge number of classifications because only a small relevant set of the possible features is selected using the AdaBoost algorithm (Freund and Schapire, 1997).

The term feature *extraction* is used for techniques that find new features rather than a subset of the original ones. In order to retain information present in the original features, the new 'extracted' features are often smooth functions of the original features. Feature extraction methods can be divided into linear and non-linear techniques. Linear techniques are restricted to yield features y that are given by a linear combination of the original features $\mathbf{x}$, i.e. $y = \mathbf{x}^\top \mathbf{w}$. Thus the features are completely specified by the weighting coefficients of the original features, collected in the vector $\mathbf{w}$. Linear feature extraction finds a weight vector $\mathbf{w}$ that optimizes a criterion that indicates the quality of the resulting feature.

Not all DRTs work by explicitly finding a mapping from original features to extracted features. A number of techniques merely produce a low dimensional representation of the given data. Of course, given the original data and the low dimensional representation one can find functions mapping between the two spaces. The advantage of doing without an explicit mapping is that no (potentially incorrect) assumptions are made about the data by choosing a specific class of functions. Depending on the application, a function that maps the original data to the extracted features may be desired or not. If only a low dimensional representation of a *specific* data set is needed (which does not have to be generalized to new data later), an explicit function is superfluous. An example of an application of feature extraction where such a function is not needed is found in some image database searching systems. The images that best match a user query are presented in a display where similar images are plotted nearby. This enables the user to quickly select a subset of images that is of interest (Rubner et al., 1998). In cases where dimension reduction has to be performed very fast in an on-line manner, as new data arrives, it is often useful to have a fixed function perform the dimension reduction. An appropriate function is then found in advance on the basis of a 'training' set of typical data. Such applications can be found in some robotic systems that navigate on the basis of images obtained from mounted cameras. The images typically contain thousands of pixels (dimensions) and need to be mapped to a low dimensional vectors that are sufficiently informative for navigation task but allow efficient further processing (Vlassis et al., 2002; Porta et al., 2004).

Below, we review several feature extraction techniques. We limit ourselves here (and throughout this thesis) to 'unsupervised' methods, i.e. methods that perform feature

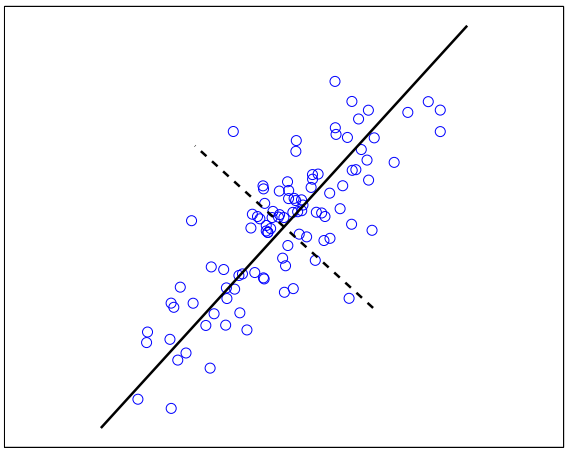


Figure 2.4: Example of PCA; data are plotted as circles, the principal component and the direction with least variance are plotted respectively with a solid and dashed line.

extraction without specification of a task that has to be performed using the data in the produced low dimensional representation. The techniques we describe can be divided into three groups:

1. **Principal components.** Principal component analysis (PCA) is a linear technique that minimizes the reconstruction error of the original data from the low dimensional representation as measured by the squared Euclidean distance. We also consider two approaches that extend PCA to extract non-linear features: kernel PCA and principal curves.

2. **Methods based on neural networks.** Auto-encoder networks and self-organizing maps are non-linear techniques from the neural-networks research field. We also consider generative topographic mapping, which is a technique similar to self-organizing maps but based on mixture density estimation.

3. **Methods based on pairwise similarity.** Most of the similarity based methods produce only a low-dimensional representation of the data without a function mapping the original features to the low dimensional space and optimization is performed directly over the low dimensional coordinates for the data.

After we describe the different techniques, we compare them in Section 2.2.4.

2.2.1 Principal components and generalizations

Below, we first describe in detail principal component analysis and kernel principal component analysis, a direct non-linear extension. Then, we consider principal curves; another non-linear extension of principal component analysis.

Principal component analysis. Principal component analysis (PCA) (Pearson, 1901; Jolliffe, 1986) is a linear feature extraction technique that can be motivated from different perspectives. Above, we mentioned that PCA minimizes the total squared distance between the original data and its reconstruction from the extracted features. Here we describe PCA as a method to find the linear feature with maximum variance of zero-mean multivariate data. By projecting the data on the principal component, we discard the dimensions of the data with less variance. If dimensions with little variance are unimportant for the task at hand (i.e. in a supervised setting), one can preprocess the data with a PCA projection to reduce the dimensionality. Fig. 2.4 illustrates PCA on a data set with points in $\mathbb{R}^2$.

Formally, let $\mathbf{X}$ be a $N \times D$ matrix containing N measurements $\mathbf{x}_n$ $(n = 1, \ldots, N)$, each being a vector in $\mathbb{R}^D$ stacked as rows in $\mathbf{X}$. Suppose that the data has zero mean; a non-zero mean can be accommodated by first subtracting the mean from the data. Clearly, any linear combination of the original features will also have zero mean. Let $\mathbf{w} \in \mathbb{R}^D$ be the vector that contains the coefficients of the linear combination that gives the new feature. Let $\mathbf{y} = \mathbf{X}\mathbf{w}$ be the vector of length N with the values of the new feature for our data. Since $\mathbf{y}$ will also have zero mean, we can write its variance as:

$$\frac{1}{N}\sum_{n=1}^{N} y_n^2 = \mathbf{y}^\top \mathbf{y}/N = \mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w}/N = \mathbf{w}^\top \mathbf{C}\mathbf{w}. \tag{2.14}$$

Thus the variance in $\mathbf{y}$, which we seek to maximize, can be expressed in terms of $\mathbf{w}$ and $\mathbf{C} = \mathbf{X}^\top \mathbf{X}/N$ – the covariance matrix of the zero mean data $\mathbf{X}$. Note that by multiplying $\mathbf{w}$ with a factor larger than one, we can enlarge the variance by any factor. To remove this degeneracy, we impose the constraint that $\mathbf{w}$ should have norm one, i.e. $\mathbf{w}^\top \mathbf{w} - 1 = 0$. Now, we can use the theory of Lagrange multipliers, see e.g. (Bishop, 1995), to solve this constrained maximization problem. Using the Lagrangian:

$$L(\mathbf{w}, \lambda) = \mathbf{w}^\top \mathbf{C}\mathbf{w} - \lambda(\mathbf{w}^\top \mathbf{w} - 1). \tag{2.15}$$

we find the critical points of the variance satisfying the constraint as vectors $\mathbf{w}$ for which both $\partial L/\partial \mathbf{w} = 0$ and $\partial L/\partial \lambda = (\mathbf{w}^\top \mathbf{w} - 1) = 0$. The first condition reads:

$$\partial L/\partial \mathbf{w} = \mathbf{C}\mathbf{w} - \lambda \mathbf{w} = 0 \tag{2.16}$$

$$\Leftrightarrow$$

$$\mathbf{C}\mathbf{w} = \lambda \mathbf{w}, \tag{2.17}$$

thus $\mathbf{w}$ is an eigenvector of $\mathbf{C}$. The second condition implies that the variance corresponding to some eigenvector $\mathbf{w}$ is given by the corresponding eigenvalue λ. This can be seen by multiplying both sides of (2.17) on the left with $\mathbf{w}^\top$. Thus the linear feature maximizing the variance of the projected data, known as the first principal component, is given by the eigenvector with largest eigenvalue of the covariance matrix.

The second principal component can now be defined as the vector $\mathbf{w}$ that maximizes the variance, under the constraint of unit norm and being orthogonal to the first principal component. The second principal component is given by the eigenvector of $\mathbf{C}$ with second largest eigenvalue. The other principal components are defined analogously. The eigenspectrum (the set of eigenvalues, ordered from large to small) can be used to select a reasonable number of principal components on which one projects the data. The sum of the eigenvalues equals the data variance, and one might select a number of principal components such that their cumulative variance is, say, 90% of the total variance. Alternatively, one can plot the eigenspectrum and see if there is a point where there is a considerable drop in the eigenvalues, i.e. a point before which the eigenvalues are of reasonable size and after which the eigenvalues get very small. More sophisticated techniques exist, see e.g. (Minka, 2001) and the discussion in Section 2.2.4.

It is not hard to show that the eigenvectors of $\mathbf{C}$ can also be obtained from the eigenvectors of the data inner product matrix $\mathbf{T} = \mathbf{X}\mathbf{X}^\top$. If $\mathbf{v}$ is an eigenvector of $\mathbf{T}$ with eigenvalue λ, then $\mathbf{w} = \mathbf{X}^\top\mathbf{v}$ is an eigenvector of $\mathbf{C}$ with eigenvalue λ/N. Note that $\mathbf{w}$ is expressed as a linear combination of the data points, where $\mathbf{x}_n$ is weighted by the n-th element of $\mathbf{v}$. Depending on whether $N < D$ (or vice versa) it is computationally more efficient to perform PCA based on $\mathbf{T}$ (or $\mathbf{C}$). Methods based on $\mathbf{T}$ (or $\mathbf{C}$) have runtime at least $O(DN^2)$ (or $O(ND^2)$), needed to compute this matrix. If only few principal components are needed, efficient iterative methods can be used that do not compute $\mathbf{C}$ nor $\mathbf{T}$. The run time per iteration and per principal component of the algorithms is $O(DN)$. Some methods are based on the expectation-maximization algorithm (Roweis, 1998; Tipping and Bishop, 1999), others on ideas from neural networks (Oja, 1982). Oja's work is closely related to a modified version of the power method (Golub and Van Loan, 1996). The FastMap algorithm (Faloutsos and Lin, 1995) finds approximate PCA projections: the projection on the first principal component is approximated by finding two distant data points and projecting the data on the line passing through these data points. Then, the distance metric is adjusted to neglect distances in the first projection direction and the process is repeated to find other approximate principal components.

The linearity of PCA suggests using a linear map to reconstruct the original data from the PCA projection. Let $\mathbf{Y} = \mathbf{X}\mathbf{W}$ be the $N \times d$ matrix having in row n the projection of $\mathbf{x}_n$ on the first d principal components, with $\mathbf{W}$ the $D \times d$ matrix with the corresponding d eigenvectors of $\mathbf{C}$ as columns. The optimal reconstruction $\hat{\mathbf{X}}$ (in the sense of minimum squared distance between original and reconstructed points) of $\mathbf{X}$ is given by $\hat{\mathbf{X}} = \mathbf{Y}\mathbf{W}^\top = \mathbf{X}\mathbf{W}\mathbf{W}^\top$.

Kernel PCA. Kernel PCA (Schölkopf et al., 1998) is an approach to extend PCA such that it can find non-linear subspaces with high variance. The basic idea is to extend or replace the original features with a (large) number of non-linear features and then to apply linear PCA in the new feature space.

If we increase the number of features, it quickly becomes unfeasible to perform PCA via the data covariance matrix since its size grows quadratically with the number of features. If we use the data inner product matrix $\mathbf{T}$ then the size of the matrix is constant, but the time needed to compute the inner products will increase linearly with the number of features. However, note that if the new features are (non-linear) functions of the original features, then the inner product in the extended feature space is still a function of the original features. It turns out that for some choices of new features, we can express the resulting inner product as a function of the original features that can be evaluated in a number of operations much smaller than the number of new features.

The function computing the inner product is called the 'kernel function' corresponding to these features. Conversely, Mercer's theorem (Schölkopf et al., 1998) provides the conditions under which a function $k(\mathbf{x}_i, \mathbf{x}_j)$ computes the inner product in some associated feature space. Using a kernel allows the representation of data in extremely high dimensional spaces without explicitly mapping the data to this feature space and thus avoiding the computational burden of using such rich representations. Some kernels even have an associated feature space with an infinite number of dimensions, e.g. the Gaussian kernel: $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(- \parallel \mathbf{x}_i - \mathbf{x}_j \parallel^2 /(2\sigma^2))$, with $\sigma^2 > 0$. The idea of using kernels to compute inner products in high dimensional feature spaces has been popularized in the machine learning field by Vapnik's work on support vector machines (Vapnik, 1995; Schölkopf and Smola, 2002). Kernels have also been applied to define non-linear counterparts for several other linear data analysis techniques, among which are Fisher discriminant analysis (Mika et al., 2001), partial least squares regression (Rosipal and Trejo, 2001), and independent component analysis (Bach and Jordan, 2002).

We now briefly discuss how we can use kernels to efficiently perform PCA in a high dimensional feature space. Let $\boldsymbol{\phi}(\mathbf{x})$ be the high dimensional feature vector corresponding to $\mathbf{x}$ and $k(\cdot, \cdot)$ the corresponding kernel function, i.e. $k(\mathbf{x}_i, \mathbf{x}_j) := \boldsymbol{\phi}(\mathbf{x}_i)^\top \boldsymbol{\phi}(\mathbf{x}_j)$. PCA computes the eigenvectors of the inner product matrix $\mathbf{T}$ of zero mean, or 'centered', data. In general we do not know whether our data is centered in the feature space associated with a particular kernel. However, we can compute the inner products of the centered data from the inner products of the non-centered data, collected in the $N \times N$ matrix $\mathbf{K}$. Let $\boldsymbol{\mu} = \frac{1}{N} \sum_i \boldsymbol{\phi}(\mathbf{x}_i)$ denote the mean of the data in the feature space, then the centered inner products t_{ij} are given by:

$$\tilde{k}(\mathbf{x}_i, \mathbf{x}_j) = (\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\mu})^\top (\boldsymbol{\phi}(\mathbf{x}_j) - \boldsymbol{\mu}) \tag{2.18}$$

$$= k(\mathbf{x}_i, \mathbf{x}_j) - \frac{1}{N} \sum_l (k(\mathbf{x}_i, \mathbf{x}_l) + k(\mathbf{x}_j, \mathbf{x}_l)) + \frac{1}{N^2} \sum_{l,m} k(\mathbf{x}_l, \mathbf{x}_m). \tag{2.19}$$

Suppose $\mathbf{v}$ is an eigenvector of the centered inner product matrix, corresponding to a principal component on which we want to project. We write v_i for the i-th element of $\mathbf{v}$ and $\bar{v}$ for the average of its elements. Then, as mentioned in the previous section, the principal component is given by $\mathbf{w} = \sum_{i=1}^{N} v_i (\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\mu})$.

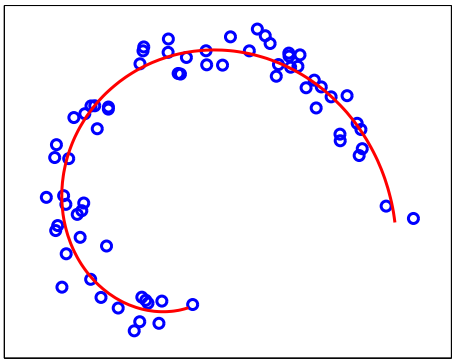


Figure 2.5: The curve is the principal curve for the data plotted as circles.

To map a new data point $\mathbf{x}$ on the principal component in the feature space, we first subtract the estimated mean in the feature space, $\boldsymbol{\mu}$, and then project it on the principal component. Thus the mapping of $\mathbf{x}$ on the principal component is given by:

$$(\boldsymbol{\phi}(\mathbf{x}) - \boldsymbol{\mu})^\top \mathbf{w} = (\boldsymbol{\phi}(\mathbf{x}) - \boldsymbol{\mu})^\top \sum_{i=1}^{N} v_i (\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\mu}) \tag{2.20}$$

$$= \sum_i k(\mathbf{x}, \mathbf{x}_i)(v_i - \bar{v}) - \frac{1}{N} \sum_{i,j} k(\mathbf{x}_i, \mathbf{x}_j)(v_i - \bar{v}). \tag{2.21}$$

Thus, the projection on the principal components can be computed directly from the kernel without explicitly mapping to the feature space.

Summarizing, PCA can be extended to extract non-linear features by replacing the original features with a new set of (non-linear) functions of the original features and then performing linear PCA. By using a kernel function we can use many non-linear features without the need to explicitly map the data to this feature space. The choice of kernel function is crucial, since it determines the type of non-linearities that are considered. Some authors (Seeger, 2000) considered the problem of automatically selecting the type of kernel or its parameters, such as σ^2 in the Gaussian kernel.

Principal curves. Principal curves are another approach to define a non-linear analogue of the linear principal component. Intuitively, a principal curve 'passes through the middle of the (curved) data cloud', as illustrated in Fig. 2.5. Several definitions of principal curves have been proposed in the literature. The earliest definition (Hastie and Stuetzle, 1989), is based on 'self-consistency' property of a curve with respect to a density p on $\mathbb{R}^D$. Let $\mathbf{f}(\lambda)$ be a smooth curve of finite length in $\mathbb{R}^D$. The 'projection index' $\lambda_{\mathbf{x}} \in \mathbb{R}$ of a point $\mathbf{x} \in \mathbb{R}^D$ is the value of λ for which $\mathbf{f}(\lambda)$ is closest to $\mathbf{x}$ (or the largest such λ if multiple exist). The curve $\mathbf{f}$ is called a principal curve with respect to p if the expectation under p of points with projection index λ equals $\mathbf{f}(\lambda)$, i.e. if $\mathrm{E}_p[\mathbf{x} : \lambda_{\mathbf{x}} = \lambda] = \mathbf{f}(\lambda)$. Note that for finite data the principal curve is not well defined

since for all but finitely many values of λ there will be no data with that projection index and thus $\mathrm{E}[\mathbf{x} : \lambda_{\mathbf{x}} = \lambda]$ is undefined. In general, conditions on the distribution that guarantee existence of principal curves defined is this way are not well understood.

In (Hastie and Stuetzle, 1989) an iterative algorithm is proposed to find principal curves for finite data sets. The authors use local averaging to overcome the above mentioned problem with finite data sets. The algorithm is summarized as follows:

1. Initialize the curve, typically to a segment of the first principal component of the data such that there is no data point that projects to either end point of the curve. The initial curve is $\mathbf{f}_0$ and set $i \leftarrow 0$.
2. Find for each $\mathbf{x}_n$ in the data set its projection index on the curve $\lambda_{\mathbf{x}_n}$.
3. Set $\mathbf{f}_{i+1}(\lambda)$ to be the average of data that has a projection index within some distance ϵ to λ. Re-parameterize the curve to unit speed.
4. If for all points $\mathbf{x}_n$ in the data set $\mathbf{f}_{i+1}(\lambda_{\mathbf{x}_n}) = \mathbf{f}_i(\lambda_{\mathbf{x}_n})$ then the curve is consistent and the algorithm is terminated. Otherwise, set $i \leftarrow i + 1$ and return to step 2.

To make the algorithm practical, the curve is defined by joining a finite number of 'support' points to form a polygonal line. In step 3 only the support points are updated and again joined to form a polygonal line that is then re-parameterized to be unit speed. The number of support points and the value of ϵ influence the resulting curve. No proof is known that shows the convergence of this algorithm.

Another approach (Kégl et al., 2000) is to define principal curves with respect to a length constraint. A curve of length l is called a principal curve if it achieves the minimum expected squared distance between points in $\mathbb{R}^D$ and the closest point on the curve among all curves of length l. For this definition the existence of at least principal curves is guaranteed if the distribution has finite second moments. Other approaches to define and find principal curves can be found in (Tibshirani, 1992; Delicado and Huerta, 2003).

The principal curve algorithms mentioned above rely on iterative adaptation of an initial curve. Experimentally we observed that these algorithms can perform poorly if the data lies along complex curves, e.g. a spiral which wraps around itself. In (Verbeek et al., 2001; Verbeek et al., 2002a) we proposed an algorithm to find principal curves that does not rely on iterative adaptation of a curve.[3] Instead, the algorithm fits a set of k separate line segments to the data, minimizing squared distance from the data points to the segments. Then, in a second phase, the line segments are connected by $k - 1$ new segments to form a polygonal line. A search is performed to find the polygonal line that minimizes an error function that takes into account the total length of the curve as well as the angles between subsequent segments. This approach is similar to the approach considered in Chapter 5, but is based on optimizing a different objective function.

[3] Because of limited space this work is not included in this thesis.

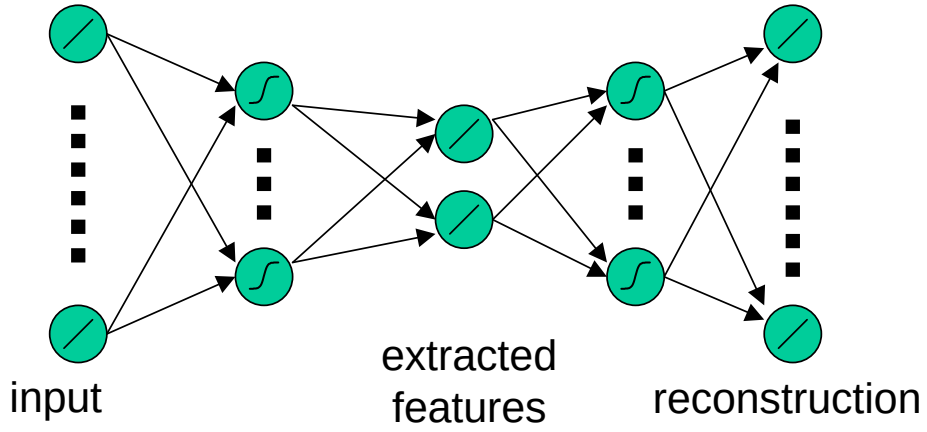


Figure 2.6: The architecture of a auto-encoder neural network with a bottleneck of two neurons.

2.2.2 Neural network based methods

In this section we discuss techniques that stem from the neural networks research community. Usually neural networks are used to model the relation between a set of input vectors $\mathbf{x} \in \mathcal{X}$ and output vectors $\mathbf{y} \in \mathcal{Y}$. A function $\mathbf{f} : \mathcal{X} \rightarrow \mathcal{Y}$ is constructed by combining several, possibly non-linear, basis functions g_i. Each basis function, sometimes called an 'activation function', models the input-output behavior of a neuron; the basis function maps the input of a neuron to the output of a neuron. Each neuron acts as a small computational unit and by connecting neurons (the output of one neuron acting as the input for a second neuron) complex functions can be constructed.

Different network architectures have been studied, among which layered feed-forward neural networks are the most well-known. In such networks the input of neurons in one layer is a linear combination of the outputs of the neurons in the previous layer. The output of the first layer of neurons in the network is set to the values of the input variables, thus if $\mathbf{x}$ is a vector with D components then we have D neurons in the first layer. The output of the neurons in the last layer give the final network output $\mathbf{f}(\mathbf{x})$. The layers of neurons in between the input and output layers are often referred to as 'hidden' layers. Common choices for the basis functions are: linear $g_i(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}_i + b_i$ and sigmoidal $g_i(x) = [1 + \exp(\mathbf{x}^\top \mathbf{w}_i + b_i)]^{-1}$. Given a set of input vectors $\mathbf{x}_i$ and corresponding desired output vectors $\mathbf{y}_i$, the parameters of the neural network are adjusted to minimize the average of the error between $f(\mathbf{x}_i)$ and $\mathbf{y}_i$. The error of the network as a function of its parameters can then be minimized by methods based on the gradient of the error w.r.t. the parameters, such as gradient descent, conjugate gradient, and the Levenberg-Marquardt algorithm (Bishop, 1995).

Auto-encoder networks. Auto-encoder neural networks are feed-forward networks used to find non-linear features of the data that allow good reconstruction of the data. The layer in the network with the fewest neurons, say d, is called the 'bottleneck'. The

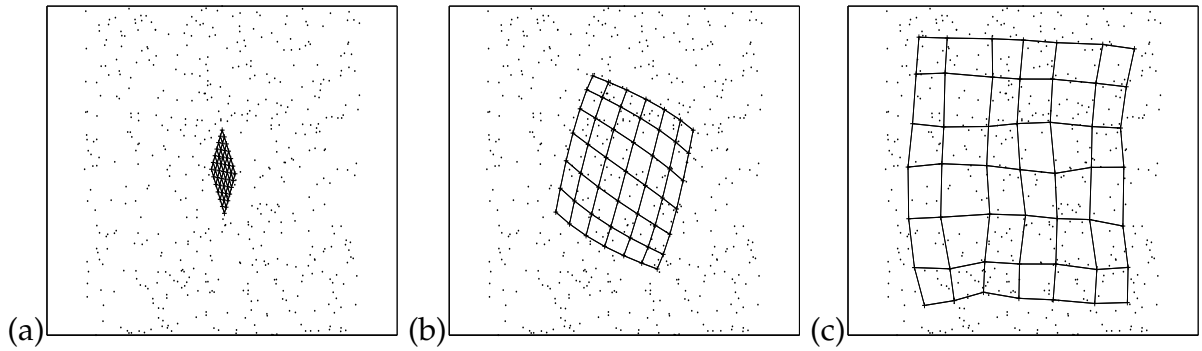


Figure 2.7: Panels (a)-(c): configurations of the SOM in the data space as training progresses on data depicted as dots.

network maps an input vector in $\mathbb{R}^D$ to a vector in $\mathbb{R}^d$, represented by the activations of the bottleneck layer and the layers after the bottleneck map the low dimensional representation in the bottleneck back to a vector in $\mathbb{R}^D$. To find a network that allows for good reconstructions we define the error of the network as the sum of squared distances between the input vectors $\mathbf{x}_i$ and their corresponding network outputs $\mathbf{f}(\mathbf{x}_i)$.

If all activation functions in the network are linear, then the function implemented by the network is linear. As might be expected, it has been shown that the minimum error is achieved if the network weights are such that the mapping of the input to bottleneck layer is a projection on the space spanned by the first d principal components (Bourlard and Kamp, 1988; Baldi and Hornik, 1989). Note that such linear networks with a bottleneck are equivalent to linear networks with just three layers: input, bottleneck and output. If we use a three layer network with non-linear activation functions in the hidden layer and linear activations in the output layer, then still the error is minimized if the network weights are such that the output of the hidden layer is a projection of the data on the PCA subspace. To find potentially better non-linear representations, the auto-encoder network should have at least five layers, as illustrated in Fig. 2.6, where the second and fourth layer have non-linear activation functions (Kramer, 1991; Oja, 1991). However, training auto-encoder neural networks with gradient based methods is notoriously difficult due to the many local optima in the error surface (Ripley, 1996).

Self-organizing maps. The self-organizing map (SOM) (Kohonen, 2001) extends the k-means clustering algorithm, discussed in Section 2.1.2, so it can be used for dimension reduction. Recall that in the k-means algorithm each cluster i $(i = 1, \ldots, k)$ is represented by its center $\boldsymbol{\mu}_i$. In each iteration two steps are performed: (i) data points are assigned to the cluster with the nearest center and (ii) cluster centers are updated as the mean of the associated data.

In the SOM, each cluster i is additionally assigned a (fixed) location $\mathbf{g}_i$ in a 'latent' space (the space of reduced dimension). A 'neighborhood' function is used to measure prox-

imity between clusters on the basis of their location in the latent space, where larger values of the neighborhood function indicate greater proximity between the clusters. In most applications of the SOM the clusters are configured in a square grid in the latent space and the neighborhood function is taken to be:

$$h_{ij} = \exp(-\lambda \|\mathbf{g}_i - \mathbf{g}_j\|^2), \tag{2.22}$$

where λ controls how fast the neighborhood function decays. The training algorithm of the SOM forces clusters with great proximity h_{ij} to represent similar data.

The training algorithm for SOM cycles trough a given data set, processing one data point after the other. The standard algorithm works on data in a $\mathbb{R}^D$ and each cluster is represented by a vector $\boldsymbol{\mu}_i$ in the data space. For each data point $\mathbf{x}_n$ the best matching, or 'winning', cluster i^* is the cluster that minimizes some distance measure in the data space, usually the Euclidean distance between the data point and $\boldsymbol{\mu}_i$. Then, each cluster center $\boldsymbol{\mu}_j$ is moved toward the data point by an amount proportional to the neighborhood function evaluated at j and the winner, i.e. $\boldsymbol{\mu}_j \leftarrow \boldsymbol{\mu}_j + \alpha h_{i^*j}(\mathbf{x} - \boldsymbol{\mu}_j)$. Thus, not only the center of the best matching cluster i^* moves toward $\mathbf{x}$, also the centers of clusters near i^* in the latent space move toward $\mathbf{x}$. In this manner clusters nearby in the latent space obtain similar cluster centers in the data space. Fig. 2.7 shows the configuration of the cluster centers in the data space during the training of the SOM on data drawn from a uniform distribution on a square in $\mathbb{R}^2$. The clusters are arranged on a two dimensional rectangular grid in the latent space, and clusters neighboring in the grid are connected in the figure.

To map the data to the latent space, i.e. to perform dimension reduction, a data point is mapped to the location of its winning cluster $\mathbf{g}_{i^*}$. In this manner, the SOM maps the coordinates of a data point $\mathbf{x}$ non-linearly to coordinates in the latent space. The SOM is said to provide a topographic data mapping since clusters nearby in the latent space will represent similar data. In Chapter 4 we present a method based on probabilistic mixture models, which is similar to Kohonen's SOM but resolves some of its limitations.

Generative topographic mapping. Generative topographic mapping (Bishop et al., 1998b) is a method similar to the self-organizing map but based on mixture density estimation. As with the SOM, the clusters are typically arranged in a regular grid in the latent space. In the data space each of the k clusters is represented as a Gaussian density. The mixture density p is defined as a weighted sum of the cluster densities, and the goal is to find parameters of the model p that yield maximum data log-likelihood $\mathcal{L}$:

$$p(\mathbf{x}) = \frac{1}{k} \sum_{s=1}^{k} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_s, \sigma^2 \mathbf{I}), \tag{2.23}$$

$$\mathcal{L} = \sum_{n=1}^{N} \log p(\mathbf{x}_n). \tag{2.24}$$

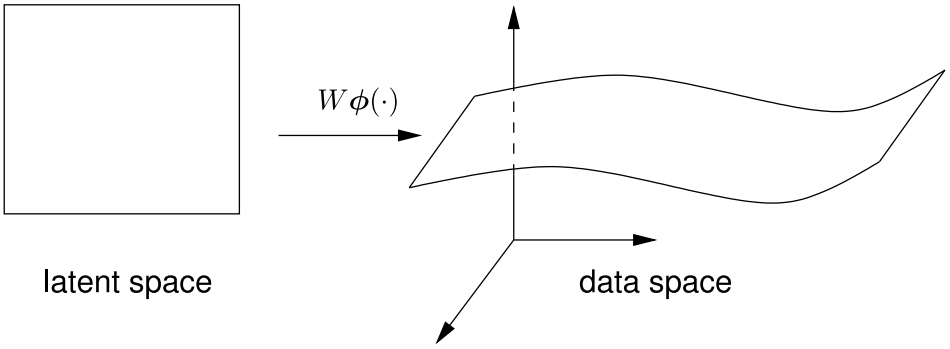


Figure 2.8: A generalized linear function maps the latent space to a low dimensional, non-linear, subspace of the data space. The means $\boldsymbol{\mu}_s$ are constrained to this subspace.

In the above, $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_s, \sigma^2\mathbf{I})$ denotes a multivariate Gaussian density with mean $\boldsymbol{\mu}_s$ and covariance matrix $\sigma^2\mathbf{I}$. Since the covariance matrix is a multiple of the identity matrix each mixture component assumes independence between all variables.

To obtain a topographic mapping of the data, the component means $\boldsymbol{\mu}_s$ are constrained to be smooth functions of their location $\mathbf{g}_s$ in the latent space, thus $\boldsymbol{\mu}_s = \mathbf{f}(\mathbf{g}_s)$. The vector valued function $\mathbf{f}$ is a weighted sum of a set of m fixed smooth non-linear basis functions, i.e. a generalized linear function:

$$\boldsymbol{\mu}_s = \mathbf{f}(\mathbf{g}_s) = \sum_{i=1}^{m} \mathbf{w}_i \phi_i(\mathbf{g}_s). \tag{2.25}$$

If the basis-functions are sufficiently smooth, components nearby in the latent space will be mapped to nearby locations in the data space, as illustrated in Fig. 2.8. Since the location of the components in the latent space and the basis functions are fixed, the only parameters of the model are the parameters controlling $\mathbf{f}$, i.e. the weight vectors $\mathbf{w}_i$ $(i = 1, \ldots, m)$ and the variance of the Gaussian distributions: σ^2. The model parameters can be iteratively re-estimated with the expectation-maximization algorithm (see Section 3.1), which yields guaranteed improvement of the data likelihood in each step.

2.2.3 Similarity based methods

The last group of techniques we consider are methods that directly find low dimensional coordinates (or an 'embedding') for the given high dimensional data, based on pairwise similarities between data points. Rather than working with data represented in a feature space, each data point is represented by comparing it against the other data points. If the data are not directly observed in terms of their pairwise (dis)similarities, we can construct a similarity matrix on the basis of a feature representation to apply these techniques.

We start our overview with multidimensional scaling, a technique dating back to the 1930's (Young and Householder, 1938). Recently, a number of new similarity based methods have been proposed. Most of these rely on finding for each point a set of highly similar points, sometimes called 'nearest neighbors'. The dimension reduction is then based on global analysis of all local neighborhoods simultaneously. We describe four recent DRTs based on nearest neighbors.

Multidimensional scaling. The term multidimensional scaling (MDS) (Cox and Cox, 1994) is used for a number of different techniques. A range of criteria exists that quantify the quality of an embedding, and depending on the used criterion different techniques are employed to optimize it. Below we discuss the three most popular of these criteria.

Given a set of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ in $\mathbb{R}^D$, it is trivial to construct the $N \times N$ matrix with all pairwise squared distances. Classical scaling is a method to achieve the inverse: given the $N \times N$ matrix with pairwise squared distances, it finds a set of points in $\mathbb{R}^D$ giving rise to exactly these distances. Of course, since squared distances are invariant to global translation and orthonormal linear mappings, i.e. rotation and mirroring, the points can only be recovered up to these operations.

The squared Euclidean distance between two data points $\mathbf{x}_i$ and $\mathbf{x}_j$ can be expressed in terms of inner products between data points:

$$\|\mathbf{x}_i - \mathbf{x}_j\|^2 = (\mathbf{x}_i - \mathbf{x}_j)^\top (\mathbf{x}_i - \mathbf{x}_j) = t_{ii} + t_{ij} - 2t_{ij}, \tag{2.26}$$

where we used $t_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$. Reversely, if we let $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^2$ (and assuming the data have zero mean), we can find the inner products from the squared distances:

$$t_{ij} = -\frac{1}{2}\left(d_{ij} - \bar{d}_i - \bar{d}_j + \bar{d}\right), \tag{2.27}$$

$$\bar{d}_i = \frac{1}{N}\sum_{j=1}^{N} d_{ij}, \tag{2.28}$$

$$\bar{d} = \frac{1}{N^2}\sum_{i,j=1}^{N} d_{ij}. \tag{2.29}$$

Given a symmetric matrix with dissimilarities d_{ij} we can construct in this way a symmetric matrix $\mathbf{T}$ with t_{ij} in the (i,j)-th entry, which is the inner product matrix of zero-mean data if $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|^2$. Since $\mathbf{T}$ is symmetric, it has orthogonal eigenvectors and eigendecomposition:

$$\mathbf{T} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top, \tag{2.30}$$

where the columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{T}$ and $\mathbf{\Lambda}$ is the diagonal matrix with the corresponding eigenvalues (Horn and Johnson, 1985). If all eigenvalues of $\mathbf{T}$ are non-negative (which is the case if the d_{ij} were computed as squared Euclidean distances),

then we can write $\mathbf{T}$ as:

$$\mathbf{T} = \mathbf{Y}\mathbf{Y}^\top, \tag{2.31}$$
$$\mathbf{Y} = \mathbf{U}\mathbf{\Lambda}^{1/2}, \tag{2.32}$$

i.e. $\mathbf{T}$ is the inner product matrix of the set of vectors given by the rows of $\mathbf{Y}$. It is not hard to show that the constant vector is an eigenvector of $\mathbf{T}$ with eigenvalue zero. This, together with the orthogonality of the eigenvectors, implies that the columns of $\mathbf{Y}$, as constructed above, are zero mean. Furthermore, assuming all eigenvalues are non-negative, the variance of the recovered data in the different dimensions is given by the corresponding eigenvalues.

In practice, the d_{ij} might be obtained directly as some measure of dissimilarity rather than computed as squared Euclidean distances of a set of points. For example, the d_{ij} could be *estimates* of the squared Euclidean distances. In such cases, $\mathbf{T}$ may have several negative eigenvalues. If $\mathbf{T}$ has negative eigenvalues, there does not exist a set of coordinates that will give rise to the given d_{ij} (and the derived t_{ij}) exactly. In such cases one can proceed by setting all negative entries in $\mathbf{\Lambda}$ to zero an proceeding as before (Cox and Cox, 1994). The justification for this procedure is that the $\mathbf{Y}$ constructed in this manner minimizes the error function associated with classical scaling (Hastie et al., 2001):

$$E_{CS} = \sum_{i,j} \left(t_{ij} - (\mathbf{y}_i - \bar{\mathbf{y}})^\top (\mathbf{y}_j - \bar{\mathbf{y}})\right)^2, \tag{2.33}$$

where $\bar{\mathbf{y}} = \frac{1}{N}\sum_i \mathbf{y}_i$ is the mean of the rows $\mathbf{y}_i$ of $\mathbf{Y}$.

If we constrain the $\mathbf{y}_i$ to be vectors of length, say k, then E_{CS} is minimized by setting all but the k largest eigenvalues to zero. In fact, if the d_{ij} are squared Euclidean distances, this procedure to find a k dimensional representation of the data is equivalent to projecting the data on the k first principal components. Thus, classical scaling is equivalent to PCA when the d_{ij} are obtained as squared Euclidian distances rather than obtained directly as some measure of dissimilarity. Hence, classical scaling is an alternative for PCA if our observations are (approximate) distances between points rather than coordinates of points.

Several other optimality criteria are used for MDS, see (Cox and Cox, 1994) for an extensive overview. Unlike classical scaling, these other criteria in general do not have analytic solutions. Typically, the criteria are minimized by non-linear optimization methods based on the gradient of the criterion w.r.t. the coordinates $\mathbf{y}_i$. Below we briefly describe two of these other criteria.

In 'least-squares' or 'Kruskal-Shephard' scaling distances (not *squared* distances) d_{ij} are approximated rather than inner products and the following error function is minimized:

$$E_{ls} = \sum_{i \neq j} \left(d_{ij} - \| \mathbf{y}_i - \mathbf{y}_j \|\right)^2. \tag{2.34}$$

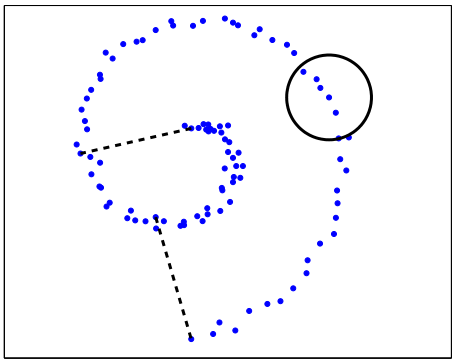


Figure 2.9: Data points distributed along a spiral in $\mathbb{R}^2$. Only if two points are nearby in the $\mathbb{R}^2$, i.e. the first point is within a small circle centered on the second point, this implies that they are also nearby on the spiral. If two points are more distant, their distance in $\mathbb{R}^2$ is not indicative for their distance on the spiral; the two dashed lines are about the same length but connect pairs of points with quite different distances an the spiral.

The Sammon mapping (Sammon, 1969) has a similar error function, but weights each term in E_{ls} with a factor d_{ij}^{-1}. This renders the approximation of small distances more important. The error function is:

$$E_{Sammon} = \sum_{i \neq j} \frac{1}{d_{ij}} \left(d_{ij} - \parallel \mathbf{y}_i - \mathbf{y}_j \parallel \right)^2 . \tag{2.35}$$

The derivative of E_{Sammon} with respect to the coordinates $\mathbf{y}_i$ is given by:

$$\frac{\partial E_{Sammon}}{\partial \mathbf{y}_i} = -4 \sum_j (\mathbf{y}_j - \mathbf{y}_i) \left[\frac{1}{d_{ij}} - \frac{1}{\parallel \mathbf{y}_i - \mathbf{y}_j \parallel} \right] . \tag{2.36}$$

Also several approaches exist that minimize the Sammon mapping error function but constrain the $\mathbf{x}_i$ to be a function (e.g. a feed-forward neural network or a radial basis function network) of the original data (Webb, 1995; de Ridder and Duin, 1997).

The focus of Sammon's mapping on preservation of small distances is best motivated with an example. In Fig. 2.9 data is distributed around a spiral in $\mathbb{R}^2$. Given these data points, we want to 'unroll' the spiral and represent the data using the coordinates on the spiral. Observe that only *locally* the distances along the spiral and the Euclidean distances in $\mathbb{R}^2$ will be similar. This observation motivates emphasis on preservation of small distances. The idea of focussing on the preservation of small distances is a theme that will reappear in the methods described below.

Stochastic neighbor embedding. The stochastic neighbor embedding (SNE) (Hinton and Roweis, 2003) algorithm does not use the dissimilarities d_{ij} directly, but uses them to define a transition matrix $\mathbf{P}$. The i-th row of the $N \times N$ transition matrix $\mathbf{P}$ contains a

distribution p_i, associated with the i-th data point, on the N data points. The transition matrix $\mathbf{P}$ defines a random walk over the data points. Starting at a particular point, each time we draw a next point according to the distribution in the row of $\mathbf{P}$ corresponding to the current point. Each row of $\mathbf{P}$ defines in a stochastic manner the 'neighbors' of a data point in the random walk. From the dissimilarities d_{ij}, the matrix $\mathbf{P}$ is constructed by setting $p_{ii} = 0$ and for $i \neq j$:

$$p_{ij} = \frac{\exp(-d_{ij}/\sigma_i^2)}{\sum_{j' \neq i} \exp(-d_{ij'}/\sigma_i^2)}. \tag{2.37}$$

Thus, with high probability subsequent points in the walk have small dissimilarity. Note that the mapping of the dissimilarities to $\mathbf{P}$ retains only information of similar points; large dissimilarities d_{ij}, relative to other $d_{ij'}$, are all mapped close to zero in $\mathbf{P}$.

When the original data are given as points in a Euclidean space, the authors propose to set d_{ij} to the squared Euclidean distance between the points. The parameters σ_i^2 have to be set and control how fast p_{ij} decays as a function of d_{ij}. The authors propose to set the parameters σ_i^2 by using knowledge of what scale to expect neighbors if such knowledge is available. If such knowledge is not available, they propose to set a desired number of neighbors k, and set the σ_i^2 in such a way that the entropy of the distribution in the i-th row, p_i, equals that of a uniform distribution over k outcomes.[4]

The goal is to find an embedding $\mathbf{y}_1, \ldots, \mathbf{y}_N$ in few dimensions that gives rise to similar stochastic neighbors. With an embedding of the data we associate a second transition matrix $\mathbf{Q}$, which is based on squared Euclidean distances between the embedding coordinates: $q_{ii} = 0$ and for $i \neq j$:

$$q_{ij} = \frac{\exp(- \parallel \mathbf{y}_i - \mathbf{y}_j \parallel^2)}{\sum_{j' \neq i} \exp(- \parallel \mathbf{y}_i - \mathbf{y}'_j \parallel^2)}. \tag{2.38}$$

Thus q_{ij} is large if and only if $\mathbf{y}_i$ and $\mathbf{y}_j$ are relatively nearby. We want the transition matrices $\mathbf{P}$ and $\mathbf{Q}$ to contain similar distributions over neighbors. To this end, the error of the embedding is defined as the sum of all Kullback-Leibler (KL) divergences between the rows of $\mathbf{P}$ and the corresponding rows of $\mathbf{Q}$:

$$E_{SNE} = \sum_{i=1}^{N} \mathcal{D}(p_i \parallel q_i) = \sum_{i=1}^{N} \sum_{j=1}^{N} p_{ij} \log \frac{p_{ij}}{q_{ij}} \geq 0. \tag{2.39}$$

Since in general it is not possible to analytically solve for coordinates for which the gradient is zero, optimization is based on the gradient, which is given by:

$$\frac{\partial E_{SNE}}{\partial \mathbf{y}_i} = -2 \sum_j (\mathbf{y}_j - \mathbf{y}_i)[(p_{ij} + p_{ji}) - (q_{ij} + q_{ji})]. \tag{2.40}$$

[4] The value of σ^2 such that p_i has a certain entropy can not be found analytically. However, the entropy is a monotone decreasing function of σ^2, and a binary search can be performed to find the correct value.

The j-th summand in the gradient can be interpreted as a force between $\mathbf{y}_i$ and $\mathbf{y}_j$. This force is repelling if the transitions between i and j are more likely according to $\mathbf{Q}$ than according to $\mathbf{P}$, i.e. if $(q_{ij} + q_{ji}) > (p_{ij} + p_{ji})$. The force is attractive if the transitions between i and j are less likely according to $\mathbf{Q}$ than according to $\mathbf{P}$.

Intuitively, SNE finds an embedding that keeps points with large p_{ij} nearby and at the same time ensures that points with small p_{ij} are not nearby. This is a consequence of the exponentiation in the mapping from $\|\mathbf{y}_i - \mathbf{y}_j\|^2$ to q_{ij}. The exact distance in the embedding between points with small p_{ij} is relatively unimportant, as long as it is large, since all large distances $\|\mathbf{y}_i - \mathbf{y}_j\|^2$ will mapped to a q_{ij} close to zero. Similarly, the exact distance in the embedding for points with p_{ij} close to one is also not too important as long as they are close, since all small distances $\|\mathbf{y}_i - \mathbf{y}_j\|^2$ are mapped to q_{ij} close to one.

In comparison, Sammon's mapping weights the errors in the preservation of distances by the inverse of the true distance. As a result (very) small errors in the preservation of distances that were originally small will yield extremely large contributions to the gradient. But also errors in the preservation of distances that were originally large will yield a relatively large contribution to the gradient as compared with SNE.

Isomap. The isomap (isometric mapping) algorithm (Tenenbaum et al., 2000) is an extension to classical scaling MDS. As noted before, PCA coincides with MDS if the distance matrix was constructed from points in a Euclidean space. Now suppose the data lies on or close to a low dimensional manifold embedded in the data space, like the spiral of Fig. 2.9. While classical scaling is based on the Euclidean distances in the data space, one can argue that the Euclidean distances are not of interest but rather the geodesic distances on the manifold, i.e. the distance along the spiral. However, if we are given only a set of points and not a description of the spiral, how do we measure the distance between points along the spiral? The isomap algorithm first estimates the geodesic distances and then uses these to apply classical scaling.

In order to estimate the geodesic distances for N data points $\mathbf{x}_i$ $(i = 1, \ldots, N)$ a 'neighborhood graph' with N vertices is constructed; data point $\mathbf{x}_i$ corresponds to vertex v_i. Each node v_i is connected by undirected edges to the k vertices that correspond to the k nearest neighbors of $\mathbf{x}_i$ in the data space. Alternatively, one can connect all pairs with $\| \mathbf{x}_i - \mathbf{x}_j \| < \epsilon$. In both cases, in the limit of infinite data sampled from the manifold, the shortest graph distances can be proven to converge to the geodesic distances (Bernstein et al., 2000). An edge between v_i and v_j is said to have a length that is equal to the Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. Once the graph is constructed, the geodesic distances are estimated as shortest paths through this graph; the length of a path is given by the sum of the length of the edges on the path. To compute shortest paths between all nodes in the graph either Dijkstra's or Floyd's algorithm (Brassard and Bratley, 1996) can be used.

Given the squares of the estimated geodesic distances we can now apply classical scal-

ing to find a low dimensional representation for the data. The parameters of the algorithm which have to be set are (i) the connectivity of the neighborhood graph, i.e. k or ϵ and (ii) the dimensionality of the low-dimensional representation. A choice of the latter can be based on analysis of the eigenvalues computed in the MDS procedure, just as with PCA. Sensitivity of the results for the size of the neighborhoods and the selection of an appropriate size is discussed in (Balasubramanian et al., 2002).

Locally linear embedding. The locally linear embedding (LLE) algorithm (Roweis and Saul, 2000) is similar to isomap in the sense that LLE also optimizes a convex error function constructed from a neighborhood graph of the data. LLE assumes that the high dimensional data lies on (or near) a smooth low dimensional (non-linear) manifold. If this is the case, then *locally* the mapping between the given high dimensional data coordinates and coordinates on the low dimensional manifold should be linear.

In the first step of the LLE algorithm for each high dimensional point $\mathbf{x}_i$ its k nearest neighbors, as measured by Euclidean distance in the high dimensional space, are collected in a set $N(i)$. Then, for each point $\mathbf{x}_i$ weights w_{ij} are found that optimally reconstruct $\mathbf{x}_i$ from its nearest neighbors. The weights are constrained to sum to one, which makes the optimal weights invariant to translations of $\mathbf{x}_i$ and its neighbors. Thus, for each point $\mathbf{x}_i$ we minimize the error:

$$e_i = \|\mathbf{x}_i - \sum_{j \in N(i)} w_{ij}\mathbf{x}_j\|^2. \tag{2.41}$$

All w_{ij} not involved in this optimization are set to zero. Optimal weights are found by solving a system of linear equations. The number of equations is equal to the number of neighbors. If there are more neighbors than dimensions in the input space the optimal weights are not unique. A unique solution can obtained by adding a regularization term that prefers similar weights: $e_i \leftarrow e_i + \sum_{j \in N(i)} w_{ij}^2$.

The optimal weights are invariant to three transformations:

1. **Scaling.** Multiplying all coordinates with a certain factor only scales the error e_i, and thus yields the same weights.
2. **Orthonormal linear mappings.** Distances are invariant to rotation and mirroring and thus so is e_i.
3. **Translation.** Since the weights are constrained to sum to one, an added offset to all coordinates is immediately cancelled in e_i.

Suppose the data points are sampled densely enough from a low dimensional manifold, then locally —i.e. for each point $\mathbf{x}_i$ and its neighbors in $N(i)$— there exists a linear map consisting of translation, rotation and scaling that maps the high dimensional coordinates to the coordinates on the manifold. Then, since the weights are invariant to these

operations, the weights computed in the high dimensional space should also give good reconstructions of the coordinates on the manifold. Since this should be the case for *all* local neighborhoods simultaneously, we set up an optimization problem over low-dimensional coordinates $\mathbf{y}_i$ that aims to minimize the reconstruction error of all the $\mathbf{y}_i$ from $\mathbf{y}_j$ with $j \in N(i)$ using the weights w_{ij} computed in the high dimensional space. In the second step of LLE we minimize the convex error function E_{LLE} over the $\mathbf{y_i}$:

$$E_{LLE} = \sum_i \|\mathbf{y}_i - \sum_{j \in N(i)} w_{ij}\mathbf{y}_j\|^2 = \mathrm{Tr}\{\mathbf{Y}^\top(\mathbf{I}-\mathbf{W})^\top(\mathbf{I}-\mathbf{W})\mathbf{Y}\} \geq 0, \tag{2.42}$$

where $\mathbf{y}_i^\top$ is the i-th row of $\mathbf{Y}$ and the (i,j)-th entry of $\mathbf{W}$ is w_{ij}. This optimization is similar to that in the first step, but here the weights are fixed and we optimize over the coordinates where in the first step the coordinates were fixed and we optimized over the weights.

Note that E_{LLE} is invariant to translation and rotation of the $\mathbf{y}_i$ for the same reasons as e_i. To obtain a unique solution, the $\mathbf{y}_i$ are constrained to be zero mean and have a diagonal covariance matrix. The error E_{LLE} is not invariant to scaling, in fact any scaling of each coordinate with a factor smaller than one will decrease the error, which means that the optimal embedding is one where all $\mathbf{y}_i$ are zero. To obtain non-degenerate solutions with variance in the $\mathbf{y}_i$, the covariance matrix is further constrained to be identity. The optimal embedding in d dimensions satisfying the constraints is found by computing the $(d+1)$ eigenvectors with smallest eigenvalues of the sparse matrix $(\mathbf{I}-\mathbf{W})^\top(\mathbf{I}-\mathbf{W})$. The constant vector is an eigenvector with the smallest eigenvalue, namely zero. The remaining d eigenvectors are then concatenated to form the $N \times d$ matrix $\mathbf{Y}$.

Recently a variant of the LLE algorithm was proposed (Donoho and Grimes, 2003), based on local estimates of the Hessian of the manifold. This method has a better theoretical motivation and also seems to produce better results, however it also requires more data than LLE to obtain sufficiently accurate estimates of the local Hessians.

Laplacian eigenmaps. The Laplacian eigenmaps (LEM) algorithm (Belkin and Niyogi, 2002; Belkin and Niyogi, 2003) is, like isomap and LLE, based on a neighborhood graph of the data. LEM is also closely related to spectral clustering, discussed in Section 2.1.3.

The first step is to build a neighborhood graph of the data by connecting each point to its nearest neighbors. Again, two options are available either we connect each point to the k nearest other points or we connect each point $\mathbf{x}_i$ to all points that have distance smaller than ϵ to $\mathbf{x}_i$. In (Belkin and Niyogi, 2002) the authors propose two alternatives to set weights on the edges of the graph: either we simply set them all to one or we set them to $\exp(-\| \mathbf{x}_i - \mathbf{x}_j \|^2 / \sigma^2)$ which has a theoretical justification. With pairs of points (i,j) not connected in the neighborhood graph we associate a weight $w_{ij} = 0$.

The goal is to find embedding coordinates $\mathbf{y}_i$ $(i = 1, \ldots, N)$ that minimize the sum of pairwise squared distances between the embedded points, weighted by their edge

weight in the neighborhood graph. Let $\mathbf{W}$ be the matrix with the (i,j)-th entry w_{ij} and let $\mathbf{D}$ be the associated 'degree matrix', i.e. $\mathbf{D}$ is diagonal and the (i,i)-th entry is $\sum_j w_{ij}$. The error function can be written as:

$$E_{LEM} = \frac{1}{2}\sum_{i,j} w_{ij} \parallel \mathbf{y}_i - \mathbf{y}_j \parallel^2 = \text{Tr}\{\mathbf{Y}^\top(\mathbf{D}-\mathbf{W})\mathbf{Y}\}. \tag{2.43}$$

Like the LLE error function, this error function is invariant to translation and rotation and a scale of the coordinates has to be fixed. This is achieved by constraining the solutions to satisfy: $\mathbf{Y}^\top\mathbf{D}\mathbf{Y} = \mathbf{I}$. The optimal embedding coordinates in d dimensions are then found through the $(d+1)$ eigenvectors $\mathbf{v}$ with smallest eigenvalues λ given by:

$$(\mathbf{D}-\mathbf{W})\mathbf{v} = \lambda\mathbf{D}\mathbf{v}. \tag{2.44}$$

The constant vector is an eigenvector with eigenvalue zero, which corresponds to mapping all points in the same coordinate. Just as with LLE we discard the smallest eigenvector and embedding coordinates are the rows of the $d \times N$ matrix $\mathbf{Y}$ formed by the remaining d eigenvectors.

Note that the LLE error function can also be written in terms of the LEM error function. We use $\mathbf{W}_{LLE}$ to denote the LLE weight matrix, and let:

$$\mathbf{W} = \mathbf{W}_{LLE} + \mathbf{W}_{LLE}^\top - \mathbf{W}_{LLE}^\top\mathbf{W}_{LLE}. \tag{2.45}$$

This matrix $\mathbf{W}$ has row sums equal to one and thus that the corresponding degree matrix equals identity. We then have that $E_{LEM} = E_{LLE}$. Furthermore, note that the relaxed Ncut problem used in spectral clustering is exactly the minimization of the error function of LEM for an embedding on the real line. See (Brand, 2004) for more details on the relation between LLE, LEM, and other related linear and non-linear methods such as (Brand, 2003; He and Niyogi, 2004).

2.2.4 Comparison of dimension reduction methods

Below we compare the different DRTs we described above using a number of different criteria. After this comparison we conclude the chapter with some general observations.

(Non)-linearity. The most obvious division of DRTs is based on whether they yield a linear or non-linear projection of the original data. In our review we have discussed two Linear techniques: PCA and classical scaling if applied to a distance matrix obtained by computing Euclidean distances for points in $\mathbb{R}^D$. All other DRTs we considered can yield non-linear projections. The ability to produce non-linear projections is in general an advantage, unless we specifically want to find a linear projection. Most non-linear techniques introduce a parameter that regulates the smoothness of the non-linear projection: e.g. the kernel in kernel PCA, the number of clusters in SOM and

generative topographic mapping (GTM), the number of basis functions in auto-encoder networks and GTM, the number of neighbors in LLE, isomap and LEM. In general these parameters have to be set by hand, although some methods for automatic determination of these parameters have been proposed (Seeger, 2000; Balasubramanian et al., 2002; de Ridder and Duin, 2002).

Estimation of intrinsic dimensionality. Another parameter of DRTs is the number of dimensions of the low dimensional representation, although in some applications the desired number of dimensions may be known a-priori. For the methods based on the computation of eigenvectors with largest eigenvalues, the eigenspectrum can be used to determine an appropriate number of dimensions. For PCA, classical scaling, and isomap the dimensionality of the data can be estimated by looking for an 'elbow' in the decrease of the error, i.e. an eigenvalue after which the eigenspectrum error decreases much slower than before that point (Tenenbaum et al., 2000).

For LLE a similar technique could be used: find the number of dimensions after which the error E_{LLE}, c.f. (2.42), starts to increase significantly. However, in (Saul and Roweis, 2003) it has been reported that this method only seems to work for contrived examples and fails for many real data sets. The authors propose to use other techniques that estimate the local dimensionality of data manifold in the original high dimensional space. It is possible to look at the (average) eigenspectrum of the covariance matrix of each local neighborhood (a point and its k neighbors), this however requires to fix a number of neighbors which impacts the spectra (Verveer and Duin, 1995). Other methods are based on measuring how the number of points that are within a distance ϵ of a point $\mathbf{x}_i$ grows as a function of ϵ, c.f. (Brand, 2003; Kégl, 2003). Whitney's theorem (Lee, 2003) states that if the intrinsic dimension of the manifold is d then it can be embedded without self-intersections in $\mathbb{R}^{2d+1}$, and thus gives us an upper bound on the number of required dimensions.

Optimization and computational aspects. An important characteristic of DRTs is the type of error function that is used. The error function, which depends on the given (high-dimensional) data, maps a low-dimensional representation of the data to a real value. The low dimensional representation that minimizes this error function is defined to be the optimal low-dimensional representation. For example the error function of PCA is the negative variance of a linear projection of the given data, which depends both on the given data and the linear projection.

A crucial property is whether or not this error function is convex. A function $f(\mathbf{x})$ is convex if for any $\mathbf{x}_1$ and $\mathbf{x}_2$ and $0 \leq \lambda \leq 1$ it holds that:

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2). \tag{2.46}$$

If a function f is convex, then all local minima of f are also global minima. Thus, to find a minimizer $\mathbf{x}$ of a convex function $f(\mathbf{x})$, it suffices to find a local minimum of f, which

is relatively easy. To find local minima, gradient based methods can be used, or more sophisticated methods if the function has additional properties that can be exploited. If, on the other hand, a function is not convex then a local minimum does not have to be the global minimum. Therefore, it does not suffice to find a local minimum, which makes the minimization considerably harder. Using a well-known quote: *"... the great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity."* (Rockafellar, 1993).

In our review we have encountered several techniques based on convex error functions: PCA, kernel PCA, classical scaling, isomap, LLE, and LEM. These methods all use quadratic error functions with additional constraints for which we can find the minimizers as eigenvectors of a matrix associated with the error function. We also encountered techniques that do not have convex error-functions: principal curves, SOM, GTM, auto-encoder networks, least-squares scaling, Sammon's mapping, and SNE.

Although one can argue which DRT has the most appropriate error function for a specific application, methods based on convex error function come with the significant advantage that their optimization is relatively easy. Non-convex error functions, on the other hand, may be great at quantifying the quality of dimension reduction, but come with potentially insurmountable optimization problems which prevent us from *finding* the optimal dimension reduction according to the error function.

A further distinction can be made in the group of techniques for which solutions are found as eigenvectors of a matrix: is the matrix sparse? If so, and we know something about the eigenvalues corresponding to the solution eigenvectors (e.g. we want the smallest, largest, or values closest to a particular number), then efficient techniques can be used to find them. These methods generally involve multiplications $\mathbf{Mv}$ of vectors $\mathbf{v}$ with a $N \times N$ matrix $\mathbf{M}$. For a dense matrix the multiplications cost $O(N^2)$ computations, but for sparse matrices the number of computations is $O(Nr)$, where r is the average number of non-zero elements in each row. Both LLE and LEM use such sparse matrices. Note that methods based on nearest neighbors (isomap, LLE, and LEM) can benefit from the same efficient nearest-neighbor finding algorithms as mentioned in Section 2.1.4 in the context of spectral clustering methods.

Observations. In this section we have reviewed DRTs that are 'unsupervised' in that they do not take into account tasks to be performed with the output of the dimension reduction technique. Consequently, there are situations where the use of unsupervised techniques to perform dimension reduction as pre-processing for another task can severely impact performance of the latter task. A simple example is given in Fig. 2.10, where data from two classes is plotted with open and closed circles respectively. The dimension with the smallest variance is optimal for classification of the data. However, preprocessing the data with PCA to obtain a one-dimensional representation would yield a representation that is uninformative for prediction of the class for new data.

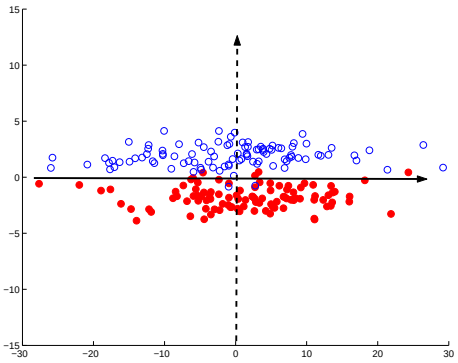


Figure 2.10: An example where the maximum variance direction (solid line) is not relevant for class discrimination. The classes (open circles vs. closed circles) are best separated by projecting on the direction with least variance (dashed line).

Thus, if dimension reduction is needed as a preprocessing step before further processing of the data ideally one should use supervised techniques that have optimality criteria that take into account the later stages of processing. Unsupervised techniques as preprocessing for another (supervised) task may be useful in situations where (i) the task is not yet known when dimension reduction has to be performed or (ii) appropriate supervised DRTs are computationally too demanding. If, unsupervised techniques are used a preprocessing for later task, to reduce the risk of losing relevant information, is a good idea not to choose an extremely low-dimensional representation.

For more extensive reviews of DRTs we refer to (Carreira-Perpiñán, 1997; Fodor, 2002) and chapter 14 of (Hastie et al., 2001).

3

MIXTURE DENSITY ESTIMATION

In this chapter we consider mixture densities, the main building block for the dimension reduction techniques described in the following chapters. In the first section we introduce mixture densities and the expectation-maximization (EM) algorithm to estimate their parameters from data. The EM algorithm finds, from an initial parameter estimate, a sequence of parameter estimates that yield increasingly higher data log-likelihood. The algorithm is guaranteed to converge to a local maximum of the data log-likelihood as function of the parameters. However, this local maximum may yield significantly lower log-likelihood than the globally optimal parameter estimate.

The first contribution we present is a technique that is empirically found to avoid many of the poor local maxima found when using random initial parameter estimates. Our technique finds an initial parameter estimate by starting with a one-component mixture and adding components to the mixture one-by-one. In Section 3.2 we apply this technique to mixtures of Gaussian densities and in Section 3.3 to k-means clustering.

Each iteration of the EM algorithm requires a number of computations that scales linearly with the product of the number of data points and the number of mixture components, this limits its applicability in large scale applications with many data points and mixture components. In Section 3.4, we present a technique to speed-up the estimation of mixture models from large quantities of data where the amount of computation can be traded against accuracy of the algorithm. However, for any preferred accuracy the algorithm is in each step guaranteed to increase a lower bound on the data log-likelihood.

3.1 The EM algorithm and Gaussian mixture densities

In this section we describe the expectation-maximization (EM) algorithm for estimating the parameters of mixture densities. Parameter estimation algorithms are sometimes also referred to as 'learning algorithms' since the machinery that implements the algorithm, in a sense, 'learns' about the data by estimating the parameters. Mixture models,

a weighted sum of finitely many elements of some parametric class of component densities, form an expressive class of models for density estimation. Due to the development of automated procedures to estimate mixture models from data, applications in a wide range of fields have emerged in recent decades. Examples are density estimation, clustering, and estimating class-conditional densities in supervised learning settings. Using the EM algorithm it is relatively straightforward to apply density estimation techniques in cases where some data is missing. The missing data could be the class labels of some objects in partially supervised classification problems or the value of some features that describe the objects for which we try to find a density estimate.

3.1.1 Mixture densities

A mixture density (McLachlan and Peel, 2000) is defined as a weighted sum of, say k, component densities. The component densities are restricted to a particular parametric class of densities that is assumed to be appropriate for the data at hand or attractive for computational reasons. Let us denote by $p(\mathbf{x}; \boldsymbol{\theta}_s)$ the s-th component density, where $\boldsymbol{\theta}_s$ are the component parameters. We use π_s to denote the weighing factor of the s-th component in the mixture. The weights must satisfy two constraints: (i) non-negativity: $\pi_s \geq 0$ and (ii) partition of unity: $\sum_{s=1}^{k} \pi_s = 1$. The weights π_s are also known as 'mixing proportions' or 'mixing weights' and can be thought of as the probability $p(s)$ that a data sample will be drawn from mixture component s. A k component mixture density is then defined as:

$$p(\mathbf{x}) \equiv \sum_{s=1}^{k} \pi_s p(\mathbf{x}; \boldsymbol{\theta}_s). \tag{3.1}$$

For a mixture we collectively denote all parameters with $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_k, \pi_1, \ldots, \pi_k\}$. Throughout this thesis we assume that all data are identically and independently distributed (i.i.d.), and hence that the likelihood of a set of data vectors is just the product of the individual likelihoods.

One can think of a mixture density as modelling a process where first a 'source' s is selected according to the multinomial distribution $\{\pi_1, \ldots, \pi_k\}$ and then a sample is drawn from the corresponding component density $p(\mathbf{x}; \boldsymbol{\theta}_s)$. Thus, the probability of selecting source s and datum $\mathbf{x}$ is $\pi_s p(\mathbf{x}; \boldsymbol{\theta}_s)$. The marginal probability of selecting datum $\mathbf{x}$ is then given by (3.1). We can think of the source that generated a data vector $\mathbf{x}$ as 'missing information': we only observe $\mathbf{x}$ and do not know the generating source. The expectation-maximization algorithm, presented in the next section, can be understood in terms of iteratively estimating this missing information.

An important derived quantity is the 'posterior probability' on a mixture component given a data vector. One can think of this distribution as a distribution on which mixture component *generated* a particular data vector, i.e. "Which component density was this

data vector drawn from?" or "to which cluster does this data vector belong?". The posterior distribution on the mixture components is defined using Bayes rule:

$$p(s|\mathbf{x}) \equiv \frac{\pi_s p(\mathbf{x};\boldsymbol{\theta}_s)}{p(\mathbf{x})} = \frac{\pi_s p(\mathbf{x};\boldsymbol{\theta}_s)}{\sum_{s'} \pi_{s'} p(\mathbf{x};\boldsymbol{\theta}_{s'})}. \tag{3.2}$$

The expectation-maximization algorithm to estimate the parameters of a mixture model from data makes essential use of these posterior probabilities.

Mixture modelling is also known as semi-parametric density estimation and it can be placed in between two extremes: parametric and non-parametric density estimation. Parametric density estimation assumes the data is drawn from a density in a parametric class, say the class of Gaussian densities. The estimation problem then reduces to finding the parameters of the Gaussian that fits the data best. The assumption underlying parametric density estimation is often unrealistic but allows for very efficient parameter estimation. At the other extreme, non-parametric methods do not assume a particular form of the density from which the data is drawn. Non-parametric estimates typically take a form of a mixture density with a mixture component for every data point in the data set. The components, often referred to as 'kernels'. A well known non-parametric density estimator is the Parzen estimator (Parzen, 1962) which uses Gaussian components with mean equal to the corresponding data point and small isotropic covariance. Non-parametric estimates can implement a large class of densities. The price we have to pay is that for the evaluation of the estimator at a new point we have to evaluate all the kernels, which is computationally demanding if the estimate is based on a large data set. Mixture modelling strikes a balance between these extremes: a large class of densities can be implemented and we can evaluate the density efficiently, since only relatively few density functions have to be evaluated.

3.1.2 Parameter estimation with the EM algorithm

The first step when using a mixture model is to determine its architecture: a proper class of component densities and the number of component densities in the mixture. We will discuss these issues in Section 3.1.3. After these design choices have been made, we estimate the free parameters in the mixture model such that the model 'fits' our data as good as possible. The expectation-maximization algorithm is the most popular method to estimate the parameters of mixture models to a given data set.

We define "fits the data as good as possible" as "assigns maximum likelihood to the data". Hence, fitting the model to given data becomes searching for the maximum-likelihood parameters for our data in the set of probabilistic models defined by the chosen architecture. Since the logarithm is a monotone increasing function, the maximum likelihood criterion is equivalent to a maximum log-likelihood criterion and these criteria are often interchanged. Due to the i.i.d. assumption, the log-likelihood of a data set

$\mathbf{X}_N = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ can be written as:

$$\mathcal{L}(\mathbf{X}_N, \boldsymbol{\theta}) = \log p(\mathbf{X}_N; \boldsymbol{\theta}) = \log \prod_{n=1}^{N} p(\mathbf{x}_n; \boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{x}_n; \boldsymbol{\theta}). \tag{3.3}$$

When no confusion arises, the dependence of the log-likelihood on $\mathbf{X}_N$ is not made explicit and we simply write $\mathcal{L}(\boldsymbol{\theta})$.

Finding the maximum likelihood parameters for a single component density is easy for a wide range of component densities and can often be done in closed-form. This is for example the case for Gaussian mixture components. However, if the probabilistic model is a mixture, the estimation often becomes considerably harder because the (log-)likelihood as a function of the parameters may have many local optima. Hence, some non-trivial optimization is needed to obtain good parameter estimates.

The expectation-maximization (EM) algorithm (Dempster et al., 1977) finds parameters at the local optima of the log-likelihood function given some initial parameter values. In our exposition we follow the generalized view on EM of (Neal and Hinton, 1998). The greatest advantages of the EM algorithm over other methods are (i) no parameters have to be set that influence the optimization algorithm, like e.g. the step-size for gradient based algorithms, and (ii) its ease of implementation. The biggest drawback is, as for all local-optimization methods, the sensitivity of the found solution to the initial parameter values. This sensitivity can be partially resolved by either (i) performing several runs from different initial parameter values and keeping the best, or (ii) finding the mixture parameters in a 'greedy' manner by starting with a single component mixture and adding new components one at a time (this is the topic of Section 3.2). Other parameter estimation techniques are gradient and sampling based methods, however these are not treated within this thesis.

Intuitively, the idea of the EM algorithm is to make estimates on the 'missing information' mentioned in the previous section: to which mixture component do we ascribe each data vector? The EM algorithm proceeds by (E-step) estimating to which component each data point belongs and (M-step) re-estimating the parameters on the basis of this estimation. This sounds like a chicken-and-egg problem: given the assignment of data to components we can re-estimate the components and given the components we can re-assign the data to the components. Indeed, this is a chicken-and-egg problem and we simply either start with initial parameters or with an initial assignment. However, after each iteration of EM, we are guaranteed that the re-estimated parameters give at least as high a log-likelihood as the previous parameter values.

Technically, the idea of EM is to iteratively define a lower bound on the log-likelihood and to maximize this lower bound. The lower bound is obtained by subtracting from the log-likelihood a non-negative quantity known as Kullback-Leibler (KL) divergence. KL divergence is an asymmetric dissimilarity measure between two probability distributions p and q from the field of information theory (Cover and Thomas, 1991), it is

defined for discrete distributions p and q with domain $\{1, \ldots, k\}$ as:

$$\mathcal{D}(q\|p) \equiv \sum_{s=1}^{k} q(s) \log \frac{q(s)}{p(s)} = -\mathcal{H}(q) - \sum_{s=1}^{k} q(s) \log p(s) > 0, \tag{3.4}$$

where $\mathcal{H}(\cdot)$ denotes the entropy of a distribution, an information theoretic measure of the 'uncertainty' or 'information' in a distribution. The KL divergence is defined analogously for probability density functions by replacing the sum over the domain with an integral over the domain. The KL divergence is zero if and only if the two distributions p and q are identical. We will use the KL divergence to measure, for each data point, how well the true posterior distribution $p(s|\mathbf{x})$ matches an approximating distribution q for this data point.

Since the KL-divergence is non-negative we can bound each of the terms $\log p(\mathbf{x}_n)$ in the log-likelihood (3.3) from below by subtracting the KL-divergence $\mathcal{D}(q_n\|p(s|\mathbf{x}_n))$. Note that this bound holds for *any* distribution q_n and becomes tight if and only if $q_n = p(s|\mathbf{x}_n)$. We will use q to denote the set of distributions $\{q_1, \ldots, q_N\}$. Combining the bounds on the individual log-likelihoods $\log p(\mathbf{x}_n)$, the complete log-likelihood (3.3) can be bounded by:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{x}_n) \geq \mathcal{F}(\boldsymbol{\theta}, q) = \sum_{n=1}^{N} \left[\log p(\mathbf{x}_n; \boldsymbol{\theta}) - \mathcal{D}(q_n\|p(s|\mathbf{x}_n))\right] \tag{3.5}$$

$$= \sum_{n=1}^{N} \left[\mathcal{H}(q_n) + \mathrm{E}_{q_n} \log \pi_s p(\mathbf{x}_n; \boldsymbol{\theta}_s)\right]. \tag{3.6}$$

Now that we have the two forms (3.5) and (3.6), iterative maximization of the lower bound $\mathcal{F}$ on the log-likelihood becomes easy. In (3.5) the only term dependent on q_n is the KL divergence $\mathcal{D}(q_n\|p(s|\mathbf{x}_n))$. To maximize $\mathcal{F}$, recall that the KL divergence is zero if and only if its two arguments are identical. Therefore, to maximize $\mathcal{F}$ w.r.t. the q_n we should set them as the posteriors: $q_n \leftarrow p(\cdot|\mathbf{x}_n)$. In this case the lower bound equals the data log-likelihood: $\mathcal{F} = \mathcal{L}$.

To maximize $\mathcal{F}$ w.r.t. $\boldsymbol{\theta}$, only the second term of (3.6) is of interest. The maximization is often easy since now we have to maximize a sum of single component log-likelihoods instead of a sum of mixture model log-likelihoods. Let us use the compact notation q_{ns} for $q_n(s)$, then if we consider optimizing the parameters of a single component s the only relevant terms in (3.6) are:

$$\sum_{n=1}^{N} q_{ns} [\log \pi_s + \log p(\mathbf{x}_n; \boldsymbol{\theta}_s)]. \tag{3.7}$$

For many component densities from the exponential family the maximization of (3.7) w.r.t. the parameters $\boldsymbol{\theta}_s$ can be done in closed form.

Summarizing, the EM algorithm consists of iteratively maximizing $\mathcal{F}$, in the E-step w.r.t. q and in the M-step w.r.t. $\boldsymbol{\theta}$. Intuitively, we can see the E-step as fixing a probabilistic assignment of every data vector to mixture components. The M-step then optimizes the mixture parameters given this assignment. Since after each E-step, we have $\mathcal{F} = \mathcal{L}$ and both the E-step and the M-step do not decrease $\mathcal{F}$ we immediately have that iterative application of these steps can not decrease $\mathcal{L}$.

The q_{ns} are often referred to as 'responsibilities', since they indicate for each data point which mixture component models it. With every data vector $\mathbf{x}_n$ we associate a random variable s_n which can take values in $\{1, \ldots, k\}$, indicating which component generated $\mathbf{x}_n$. These variables are often called 'hidden' or 'unobserved' variables since we have no direct access to them. EM treats the mixture density log-likelihood maximization problem as a problem of dealing with missing data. Similar approaches can be taken to fit (mixture) density models to data where there truly is missing data in the sense that certain values are missing in the input vector $\mathbf{x}$.

Generalized and variational EM. The EM algorithm presented above can be modified in two interesting ways, which both give-up maximization of $\mathcal{F}$ and settle for increase of $\mathcal{F}$. The concession can be made either in the M-step (generalized EM) or in the E-step (variational EM). These modifications are useful when computational requirements of either step of the algorithm become intractable.

For some models it may be difficult to maximize $\mathcal{F}$ w.r.t. $\boldsymbol{\theta}$ but relatively easy to find values for $\boldsymbol{\theta}$ that increase $\mathcal{F}$. For example, it might be easy to maximize $\mathcal{F}$ w.r.t. either one of the elements of $\boldsymbol{\theta}$ but not to maximize w.r.t. all of them simultaneously. However, the convergence to local optima (or saddle points) can still be guaranteed when only increasing instead of maximizing $\mathcal{F}$ in the M-step, algorithms that do so are known as 'generalized EM' algorithms.

Above, due to the independence assumption on the data vectors it was possible to construct the EM lower bound by bounding each of the individual log-likelihoods $\log p(\mathbf{x}_n)$. This iterative bound optimization strategy can also be applied to other settings with unobserved variables. However, in some cases hidden variables are dependent on each other given the observed data. If we have N hidden variables with k possible values each, then in principle the distribution over hidden variables is characterized by k^N numbers. In such cases, the number of summands in the expected joint log-likelihood we have to optimize in the M-step also equals k^N. In this case tractability can be obtained if we do not allow for general distributions q over the hidden variables, but only those from a class $\mathcal{Q}$ which gives us a tractable number of summands in the expected joint log-likelihood. For example, we can use distributions over the hidden variables that factor over the different hidden variables. EM algorithms using a restricted class $\mathcal{Q}$ are known as 'variational EM' algorithms. The term refers to the 'variational parameters' that characterize the distributions $q \in \mathcal{Q}$. The variational parameters are optimized

in the E-step of the variational EM algorithm. Most often the variational approach is used to decouple dependencies present in the true distribution over hidden variables which give rise to intractability. An example of a variational EM algorithm is the mean-field algorithm used in hidden Markov random fields, which are applied to image segmentation problems (Celeux et al., 2003).

When using a variational EM algorithm we can no longer guarantee that $\mathcal{F}$ equals the log-likelihood after the E-step, since we cannot guarantee that the true posterior is in $\mathcal{Q}$. Therefore, it is also not guaranteed that the log-likelihood will increase after each EM iteration. However, we can at least still guarantee that the lower bound on the log-likelihood $\mathcal{F}$ will increase after each EM iteration. By restricting the q to a specific class of distributions $\mathcal{Q}$, we effectively augment the log-likelihood objective of the optimization algorithm with a penalty term —the generally non-zero KL divergence— that measures how close the true distribution over the hidden variables is to $\mathcal{Q}$. Thus variational EM parameter estimation algorithms have a bias towards parameters that yield posterior distributions over the hidden variables similar to a member of $\mathcal{Q}$.

3.1.3 Model selection

To apply mixture models, two model selection issues have to be resolved: (i) how many components should be used and (ii) which class of component densities should be used. These two factors together determine the 'model structure'. These type of design choices can be compared with the selection of the number and type of hidden nodes in feed-forward neural networks. A trade-off has to be made when choosing the class of models that is used. More complex models (e.g. more mixture components) allow for modelling of more properties of the data, and in general lead to a better fit to the data. However, when using more complex models spurious artifacts of the data due to a noisy measurement system can also be captured by the model. Capturing accidental properties of the data may degrades the estimates, an effect known as 'overfitting'. Less complexity allows for more robust identification of the best model within the selected complexity class and yields a smaller risk of overfitting the data.

Throughout the previous decades several criteria have been proposed to resolve the model selection problem. However, the problem of model selection seems far from being solved. Methods for model selection can be roughly divided into four groups:

1. Methods using concepts from statistical learning theory, e.g. structural risk minimization (Vapnik, 1995). Most of these methods use worst-case guarantees of performance on future data which are derived by a statistical analysis.

2. Information theoretical methods, such as the minimum description length (MDL) principle (Rissanen, 1989). These methods are based on a data compression principle: a model is selected that allows for maximal compression of the data, where

the rate of compression is measured by the total number of bits required to encode both the model and the data (using the model).

3. Bayesian methods that use a prior distribution on model structures and parameters together with a distribution on data given models to define a posterior distribution $p(\text{model structure}|\text{data})$. To compute the required probabilities several approximation techniques have been proposed, e.g. Bayesian information criterion (Schwarz, 1978), variational Bayesian learning (Beal and Ghahramani, 2003), and a number of sampling techniques (Andrieu et al., 2003).

4. Holdout methods that keep part of the data to assess performance of models learned on the rest of the data, e.g. cross-validation (Webb, 2002).

Note that holdout methods are applicable only in cases where there is abundant data, and one can afford to ignore a part of the data for parameter estimation in order to obtain an independent test-set.

Bayesian techniques are attractive because they allow for unified treatment of model selection and parameter estimation. The application of compression based methods like MDL is sometimes problematic because the so called 'two-part' MDL code requires a particular coding scheme and it is not always clear which coding scheme should be used (Verbeek, 2000). Methods based on worst-case analysis are known to give very conservative guarantees, which limits their use.

In general, to apply a particular model selection criterion, parameter estimation has to be performed for models with different numbers of components. The algorithm for estimating the parameters of a Gaussian mixture model we present in Section 3.2 finds a parameter estimate of a k-component mixture by iteratively adding components, starting from a single Gaussian. Thus, this algorithm is particularly useful when the number of components has to be determined, since the model selection criterion can be applied to mixtures with $1, \ldots, k$ components as components are added to the mixture.

3.1.4 Gaussian mixture models

In this section we discuss the EM algorithm for Mixtures of Gaussian densities (MoG). Recall that we already saw one example of a MoG: generative topographic mapping in Section 2.2.2. We also present a hierarchy of shapes the covariance matrices can take. In the hierarchy, increasingly stringent constraints are placed on the form of the covariance matrix. Then, we discuss how some of the shapes for covariance matrices can be interpreted as linear latent variable models.

Gaussian densities are probably the most commonly used densities to model continuous valued data. The first reason for this popularity is that maximum likelihood parameter estimation can be done in closed form and only requires computation of the data mean and covariance. The second reason is that of all densities with a particular variance, the

Gaussian density has the largest entropy and therefore is the most 'vague' density in this sense. This last property motivates the use of the Gaussian as a default density when there are no reasons to assume that some other parametric density is more appropriate to model the data at hand.

A Gaussian density in a D-dimensional space, characterized by its mean $\boldsymbol{\mu} \in \mathbb{R}^D$ and $D \times D$ covariance matrix $\boldsymbol{\Sigma}$, is defined as:

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\theta}) \equiv (2\pi)^{-D/2} |\boldsymbol{\Sigma}|^{-1/2} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right], \tag{3.8}$$

where $\boldsymbol{\theta}$ denotes the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ and $|\boldsymbol{\Sigma}|$ denotes the determinant of $\boldsymbol{\Sigma}$. In order for (3.8) to be a proper density, it is necessary and sufficient that the covariance matrix be positive definite. Throughout this thesis we implicitly assume that the likelihood is bounded, e.g. by restricting the parameter space such that the determinant of the covariance matrices is bounded, and hence the maximum likelihood estimator is known to exist (Lindsay, 1983). Alternatively, the imperfect precision of each measurement can be taken into account by treating each data point as a Gaussian density centered on the data point and with small but non-zero variance. We then maximize the *expected* log-likelihood, which is bounded by construction.

The EM algorithm for mixture of Gaussian densities. In the E-step of the EM algorithm for a MoG we compute the posterior probabilities $p(s|\mathbf{x})$ according to (3.2) and set $q_{ns} \leftarrow p(s|\mathbf{x}_n)$. In the M-step we maximize $\mathcal{F}$ w.r.t. $\boldsymbol{\theta}$, by setting the partial derivatives of $\mathcal{F}$ w.r.t. the elements of $\boldsymbol{\theta}$ to zero and taking the constraints on the π_s into account, and find the updates:

$$\pi_s \leftarrow \frac{1}{N}\sum_{n=1}^{N} q_{ns}, \tag{3.9}$$

$$\boldsymbol{\mu}_s \leftarrow \frac{1}{N\pi_s}\sum_{n=1}^{N} q_{ns}\mathbf{x}_n, \tag{3.10}$$

$$\boldsymbol{\Sigma}_s \leftarrow \frac{1}{N\pi_s}\sum_{n=1}^{N} q_{ns}(\mathbf{x}_n-\boldsymbol{\mu}_s)(\mathbf{x}_n-\boldsymbol{\mu}_s)^\top. \tag{3.11}$$

The update of the mean uses the new mixing weight and the update of the covariance matrix uses the new mean and mixing weight.

In many practical applications of MoGs for clustering and density estimation no constraints are imposed on the covariance matrix. Using an unconstrained covariance matrix, the number of parameters of a Gaussian density grows quadratically with the data dimensionality. For example, using $16 \times 16 = 256$ pixel gray-valued images of an object as data (treating an image as a 256 dimensional vector), we would need to estimate over

32.000 parameters! To reduce the number of parameters in the covariance matrix, it can be constrained to have a form that involves fewer parameters. In some applications a constrained covariance matrix can be appropriate to reflect certain assumptions about the data generating process. See the discussion below on latent variable models.

Next, we discuss several frequently used ways to constrain the covariance matrix. We will use $\mathbf{I}$ to denote the identity matrix, $\mathbf{\Psi}$ to denote a diagonal matrix, and $\mathbf{\Lambda}$ to denote a $D \times d$ matrix, with $d < D$.

Type 1: factor analysis. The covariance matrices in factor analysis (FA) are constrained to be of the form:

$$\mathbf{\Sigma} = \mathbf{\Psi} + \mathbf{\Lambda}\mathbf{\Lambda}^\top. \tag{3.12}$$

The d columns of $\mathbf{\Lambda}$ are often referred to as the 'factor loadings'. Each column of $\mathbf{\Lambda}$ can be associated with a latent variable (see the discussion on latent variable models below). In the diagonal matrix $\mathbf{\Psi}$ the variance of each data coordinate is modelled separately, and additional variance is added in the directions spanned by the columns of $\mathbf{\Lambda}$. The number of parameters to specify the covariance matrix is $O(Dd)$. In (Ghahramani and Hinton, 1996) the EM algorithm for mixtures of factor analyzers is given. The determinant and the inverse of the covariance matrix can be efficiently computed using two identities:

$$|\mathbf{A} + \mathbf{BC}| = |\mathbf{A}| \times |\mathbf{I} + \mathbf{CA}^{-1}\mathbf{B}|, \tag{3.13}$$

$$(\mathbf{A} + \mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{C}^{-1} + \mathbf{DA}^{-1}\mathbf{B})^{-1}\mathbf{DA}^{-1}. \tag{3.14}$$

Using these identities we only need to compute inverses and determinants of $d \times d$ and diagonal matrices, rather than of full $D \times D$ matrices:

$$|\mathbf{\Psi} + \mathbf{\Lambda}\mathbf{\Lambda}^\top| = |\mathbf{\Psi}| \times |\mathbf{I} + \mathbf{\Lambda}^\top\mathbf{\Psi}^{-1}\mathbf{\Lambda}|, \tag{3.15}$$

$$(\mathbf{\Psi} + \mathbf{\Lambda}\mathbf{\Lambda}^\top)^{-1} = \mathbf{\Psi}^{-1} - \mathbf{\Psi}^{-1}\mathbf{\Lambda}(\mathbf{I} + \mathbf{\Lambda}^\top\mathbf{\Psi}^{-1}\mathbf{\Lambda})^{-1}\mathbf{\Lambda}^\top\mathbf{\Psi}^{-1}. \tag{3.16}$$

Type 2: principal component analysis. Here the constraints are similar to those of FA, but the matrix $\mathbf{\Psi}$ is further constrained to be a multiple of the identity matrix:

$$\mathbf{\Sigma} = \sigma^2\mathbf{I} + \mathbf{\Lambda}\mathbf{\Lambda}^\top \quad \text{with} \quad \sigma > 0. \tag{3.17}$$

A MoG where the covariance matrices are of this type is termed a 'mixture of probabilistic principal component analyzers' (Tipping and Bishop, 1999).

Principal component analysis (Jolliffe, 1986) (PCA) appears as a limiting case if we use just one mixture component and let the variance approach zero: $\sigma^2 \to 0$ (Roweis, 1998). Let $\mathbf{U}$ be the $D \times d$ matrix with the eigenvectors of the data covariance matrix with the d largest eigenvalues, and $\mathbf{V}$ the diagonal matrix with the corresponding eigenvalues

on it diagonal. Then, the maximum likelihood estimate of $\mathbf{\Lambda}$ is given by $\mathbf{\Lambda} = \mathbf{U}\mathbf{V}^{\frac{1}{2}}$. For probabilistic PCA the number of parameters is also $O(Dd)$, as for FA.

To see the difference between PCA and FA note the following. Using PCA we can arbitrarily rotate the original basis of our space: the covariance matrix that is obtained for the variables corresponding to the new basis is still of the PCA type. Thus the class of PCA models is closed under rotations, which is not the case for FA. On the other hand, if some of the data dimensions are scaled, the scaled FA covariance matrix is still of the FA type. Thus the class of FA models is closed under non-uniform scaling of the variables, which is not the case for PCA. Hence if the relative scale of the variables is not considered meaningful, then an FA model may be preferred over a PCA model, and vice versa if relative scale is important.

Type 3: k-subspaces. Here, we further restrict the covariance matrix such that the norm of all columns of $\mathbf{\Lambda}$ is equal:

$$\mathbf{\Sigma} = \sigma^2\mathbf{I} + \mathbf{\Lambda}\mathbf{\Lambda}^\top \quad \text{with} \quad \mathbf{\Lambda}^\top\mathbf{\Lambda} = \rho^2\mathbf{I} \quad \text{and} \quad \sigma, \rho > 0. \tag{3.18}$$

This type of covariance matrix was used in (Verbeek et al., 2002b), for non-linear dimension reduction technique similar to the one described in Chapter 5. The number of parameters is again $O(Dd)$.

If we train a MoG with this type of covariance matrix with EM, then this results in an algorithm known as k-subspaces (Kambhatla and Leen, 1994; de Ridder, 2001) under the following conditions: (i) all mixing weights are equal, $\pi_s = 1/k$, and (ii) the σ_s and ρ_s are equal for all mixture components: $\sigma_s = \sigma$ and $\rho_s = \rho$. Then, if we take the limit as $\sigma \to 0$, we obtain k-subspaces. The term k-subspaces refers to the k linear subspaces spanned by the columns of the $\mathbf{\Lambda}_s$. The k-subspaces algorithm, a variation on the k-means algorithm (see Section 2.1.2), iterates two steps:

1. Assign each data point to the subspace which reconstructs the data vector the best (in terms of squared distance between the data point and the projection of the data point on the subspace).
2. For every subspace compute the PCA subspace of the assigned data to get the updated subspace.

Type 4: isotropic. This shape of the covariance matrix is the most restricted:

$$\mathbf{\Sigma} = \sigma^2\mathbf{I} \quad \text{with} \quad \sigma > 0. \tag{3.19}$$

This model is referred to as isotropic or spherical variance since the variance is equal in all directions. It involves just the single parameter σ.

If all components share the same variance σ^2 and letting it approach zero, the posterior on the components for a given data vector tends to put all mass on the component

closest in Euclidean distance to the data vector (Bishop, 1995). This means that the E-step reduces to finding the closest component for all data vectors. The expectation in (3.6) then just counts the term for the closest component. This means for the M-step that the centers $\boldsymbol{\mu}_s$ of the components are found by simply averaging all the data vectors for which it is the closest. This simple case of the EM algorithm coincides exactly with the k-means algorithm.

Taking $\boldsymbol{\Sigma}$ diagonal, and thus letting all variables be independently distributed, is also a popular constraint yielding $O(d)$ parameters, however it is not used by any of the techniques discussed in this thesis.

Linear latent variable models. Three types of covariance matrix (FA, PCA and k-subspaces) can be interpreted as linear latent variable models. The idea is that the data can be thought of as being generated in a low dimensional latent space, which is linearly embedded in the higher dimensional data space. Some noise may make the observed data deviate from the embedded linear subspace.

For these models it is conceptually convenient to introduce a new, hidden variable g for every data vector. This variable represents the (unknown) d-dimensional latent coordinate of the data vector. Using this variable we can write the density for the data vectors by marginalizing over the hidden variable:

$$p(\mathbf{g}) = \mathcal{N}(\mathbf{g}; 0, \mathbf{I}) \tag{3.20}$$

$$p(\mathbf{x}|\mathbf{g}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu} + \boldsymbol{\Lambda}\mathbf{g}, \boldsymbol{\Psi}) \tag{3.21}$$

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{g})p(\mathbf{g})\mathrm{d}\mathbf{g} = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Psi} + \boldsymbol{\Lambda}\boldsymbol{\Lambda}^\top) \tag{3.22}$$

It is not necessary to introduce g to formulate the model, but it may be convenient to use g for computational reasons. For example, in (Roweis, 1998) this view is adopted to compute PCA subspaces without computing the data covariance matrix or the data inner-product matrix. The cost to compute these matrices is, respectively, $O(ND^2)$ and $O(DN^2)$. By treating the latent coordinates as hidden variables one can perform PCA with an EM algorithm that has a cost of $O(dDN)$ per iteration to find the d principal components. This EM algorithm for PCA involves computing the posterior distribution on g given x, which is given by:

$$p(\mathbf{g}|\mathbf{x}) = \mathcal{N}(\mathbf{g}; \mathbf{m}(\mathbf{x}), \mathbf{C}), \tag{3.23}$$

$$\mathbf{m}(\mathbf{x}) = \mathbf{C}\boldsymbol{\Lambda}^\top\boldsymbol{\Psi}^{-1}(\mathbf{x} - \boldsymbol{\mu}), \tag{3.24}$$

$$\mathbf{C}^{-1} = \mathbf{I} + \boldsymbol{\Lambda}^\top\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}. \tag{3.25}$$

We can use linear latent variable models to reduce the dimensionality of the data by inferring the d-dimensional latent coordinates from the D-dimensional data vectors and using the found latent coordinates for further processing of the data. Another option

is to use these models for clustering if there is reason to expect that the clusters show significant variation only in a few dimensions and very small variation in all others.

3.2 Efficient greedy learning of Gaussian mixtures

The EM algorithm is known to converge to local maxima of the data log-likelihood w.r.t. the parameter vector. However, convergence to a global maximum is not guaranteed. The data log-likelihood under the mixture distribution returned by the EM algorithm is highly dependent on the initial parameter vector used by the EM algorithm. The standard procedure to overcome this initialization dependence is to start the EM algorithm from several random initializations and retain the best obtained result.

In this section we present a greedy EM algorithm to learn mixtures of Gaussian densities, designed to overcome the sensitivity of the normal EM algorithm to initialization of the parameters.[1] Our approach is motivated by an approximation result (Li and Barron, 2000). This result states that we can 'quickly' approximate any target density with a finite mixture model, as compared to the best we can do with any mixture from the same component class. The result also holds if we learn the mixture models in a greedy manner: start with one component and optimally add new components one after the other. The learning algorithm alternates between optimization of the current mixture model and adding a new component to the mixture after convergence.

To determine the optimal new component to insert, the parameters yielding the global maximum of a two-component mixture log-likelihood have to be found, which is again non-trivial. We propose a search heuristic to locate the global maximum. Our search procedure selects the best element from a set of candidate new components. Since the greedy algorithm generates a sequence of mixtures with an increasing number of components, it can be particularly useful when the optimal number of mixture components has to be found automatically.

The rest of this section is organized as follows: Section 3.2.1 discusses greedy learning of a MoG. Our component insertion procedure is discussed in Section 3.2.2. Related work is discussed in Section 3.2.3, and in Section 3.2.4 we present experimental results comparing our method with alternative methods. We draw conclusions in Section 3.2.5.

3.2.1 Greedy learning of Gaussian mixtures

The basic idea of greedy mixture learning is simple: instead of starting with a (random) configuration of all components and improving upon this configuration with the

[1] The material of this section is largely drawn from (Verbeek et al., 2003c).

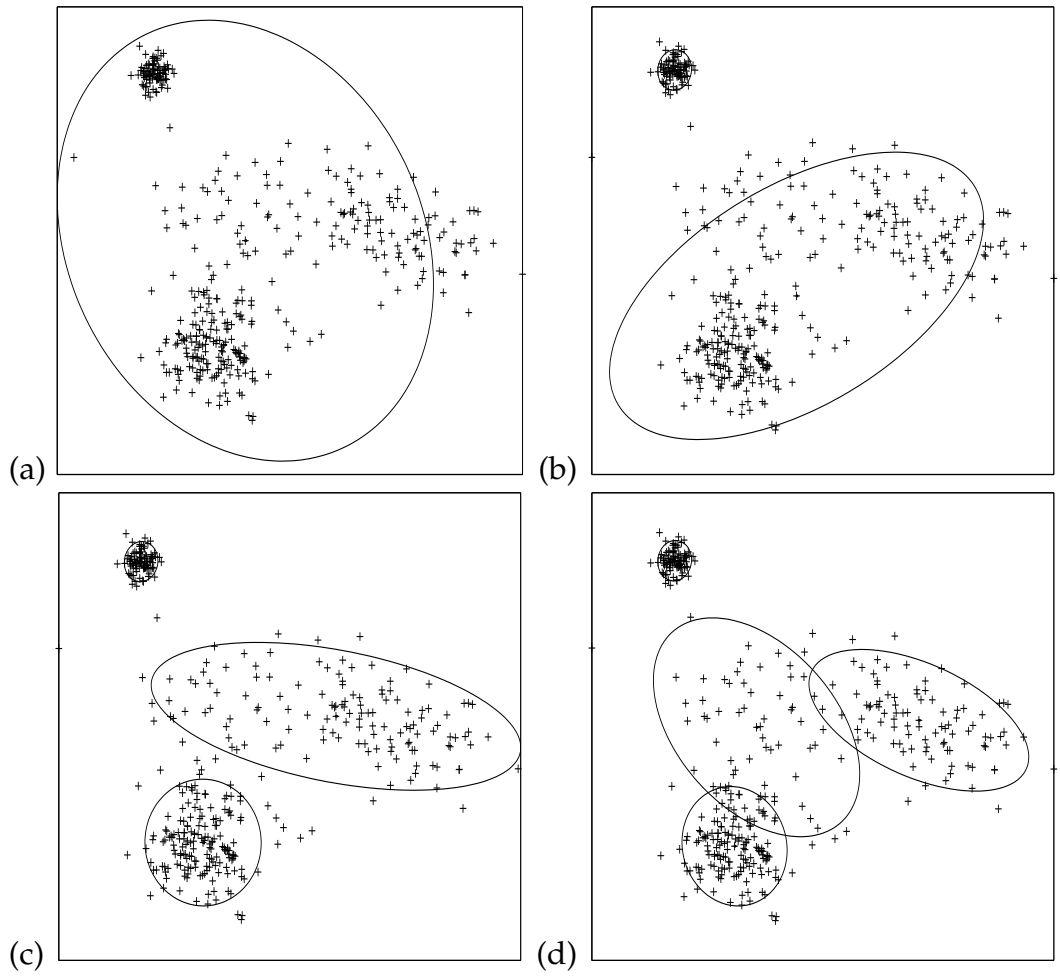


Figure 3.1: Panels (a)-(d) illustrate the construction of a 4-component mixture distribution.

EM algorithm, we build the mixture component-wise. We start with the optimal one-component mixture, whose parameters are trivially computed, then we start repeating two steps until a stopping criterion is met. The steps are: (i) insert a new component and (ii) apply the EM algorithm until convergence. The stopping criterion can implement one of the model selection criteria mentioned in Section 3.1.3, or the algorithm is terminated when a specified number of components is reached. An example of greedy mixture learning is given in Fig. 3.1, which depicts the evolution of a solution for data point plotted as '+'-signs. Mixtures with one up to four components are depicted, and each component is shown as an ellipse which has the eigenvectors of the covariance matrix as axes and radii of twice the square root of the corresponding eigenvalue.

The motivation for the greedy approach is that the optimal component insertion problem involves a factor k fewer parameters than the original problem of finding a near optimal starting configuration, and we therefore expect it to be an easier problem. The intuitive idea is that we can find the optimal (with respect to log-likelihood) $(k+1)$-

component mixture by local search if we start from the mixture obtained by inserting optimally a new component in the optimal k-component mixture. A recent theoretical result provides complementary motivation. Recall the Kullback-Leibler (KL) divergence between two densities p and q, defined as:

$$\mathcal{D}(p\|q) = \int_{\Omega} p(\mathbf{x}) \log\left[p(\mathbf{x})/q(\mathbf{x})\right] \mathrm{d}\mathbf{x}, \tag{3.26}$$

where Ω is the domain of the densities p and q. In this section we will use ϕ to indicate a component density of a mixture and use p_k to refer to a mixture of k components, i.e. $p_k(\mathbf{x}) = \sum_{s=1}^{k} \pi_s \phi(\mathbf{x}; \boldsymbol{\theta}_s)$. In (Li and Barron, 2000) it is shown that for an arbitrary probability density function p there exists a sequence $\{p_i\}$ of finite mixtures such that p_k achieves KL divergence $\mathcal{D}(p\|p_k) \leq \mathcal{D}(p\|p_f) + c/k$ for every (possibly non-finite) mixture $p_f = \int \phi(\mathbf{x}; \boldsymbol{\theta}) f(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}$, with f a density over the parameter space. Hence, the difference in KL divergence achievable by k-component mixtures and the KL divergence achievable by any mixture from the same family of components tends to zero with a rate of c/k, where c is a constant not dependent on k but only on the component family. Furthermore, it is shown in (Li and Barron, 2000) that this bound is achievable by employing the greedy scheme discussed above. This tells us that we can 'quickly' approximate any density by the greedy procedure. Therefore, we might expect the results of the greedy procedure, as compared to methods that initialize all-at-once, to differ more when fitting mixtures with many components.

The sequence of mixtures generated by the greedy learning method can conveniently be used to guide a model selection process in the case of an unknown number of components. We can use the k-component mixture, for $k \in \{1, \ldots, k_{\max}\}$, as an estimate of the maximum-likelihood estimator with k components and combine this with the other factors of the model selection criterion that is employed. When using a non-greedy approach, we would have to fit a k-component mixture for each $k \in \{1, \ldots, k_{\max}\}$ from scratch. This would be computationally unattractive, since for each k we would need to do several runs of the EM algorithm starting from different initial parameters.

A general scheme for greedy learning of mixtures. Let $\mathcal{L}(\mathbf{X}_N, p_k) = \sum_{n=1}^{N} \log p_k(\mathbf{x}_n)$ (we will just write $\mathcal{L}_k$ if no confusion arises) denote the log-likelihood of the data set $\mathbf{X}_n$ under the k-component mixture p_k. The greedy learning procedure outlined above can be summarized as follows:

1. Compute the maximum likelihood one-component mixture p_1. Set $k \leftarrow 1$.
2. Find the optimal new component $\phi(\mathbf{x}; \boldsymbol{\theta}^*)$ and mixing weight α^*:

$$\{\boldsymbol{\theta}^*, \alpha^*\} = \arg\max_{\{\boldsymbol{\theta}, \alpha\}} \sum_{n=1}^{N} \log\left[(1-\alpha) p_k(\mathbf{x}_n) + \alpha \phi(\mathbf{x}_n; \boldsymbol{\theta})\right]. \tag{3.27}$$

Set $p_{k+1}(\cdot) \leftarrow (1-\alpha^*) p_k(\cdot) + \alpha^* \phi(\cdot; \boldsymbol{\theta}^*)$ and $k \leftarrow k+1$;

3. (optional) Update p_k with the EM algorithm until it converges.

4. If a stopping criterion is met then quit, else go to step 2.

The crucial step is component insertion (step 2). In the next section we describe several possibilities to implement this step. Note that step 3 can be implemented by other algorithms than EM. Moreover, step 3 is not needed to obtain the approximation result of (Li and Barron, 2000), but improves practical results dramatically.

3.2.2 Efficient search for new components

Suppose we have obtained a k-component mixture p_k. In step 2 of the above greedy algorithm the component characterized by equation (3.27) has to be found. We refer to this component as the 'optimal' component and to problem of finding it as the 'insertion problem'. It is easily shown that if we fix p_k and $\boldsymbol{\theta}$ then $\mathcal{L}_{k+1}$ is concave as function of α only, allowing efficient optimization. However, as function of $\boldsymbol{\theta}$, $\mathcal{L}_{k+1}$ can have multiple local maxima and no constructive method is known to identify the optimal component. Hence, a search has to be performed to identify the optimal component.

In (Li, 1999) it is proposed to quantize the parameter space to locate the global maximum, but this is infeasible when learning mixtures with high dimensional parameter spaces since the number of required quantization cells quickly becomes too large. We developed our search procedure for the optimal component as an improvement upon the insertion procedure proposed in (Vlassis and Likas, 2002) (we will from now on refer to the latter procedure as VL). Below, we will discuss the VL procedure and identify its drawbacks. Then, we present our improved search procedure. Both methods are based on using a set of candidate components. The best candidate is identified as the candidate component $\phi(\mathbf{x}; \hat{\boldsymbol{\theta}})$ that maximizes the likelihood when mixed into the previous mixture by a factor $\hat{\alpha}$ as in (3.27). Then, in step 3 of the general algorithm, instead of the (unknown) optimal parameter pair $(\boldsymbol{\theta}^*, \alpha^*)$ we use $(\hat{\boldsymbol{\theta}}, \hat{\alpha})$.

The VL insertion procedure. The VL procedure uses N candidate components. Every data point is the mean of a corresponding candidate. All candidates have the same covariance matrix $\sigma^2\mathbf{I}$, where σ is set in an automatic manner. The value of σ depends on the data dimensionality and the number of data points and is based on a non-parametric density estimation procedure (Wand, 1994). For each candidate component, the mixing weight α is set to the mixing weight maximizing the second order Taylor approximation of $\mathcal{L}_{k+1}$ around $\alpha = 1/2$. The candidate yielding highest log-likelihood when inserted in the existing mixture p_k is selected. The selected component is then updated using partial EM steps before it is actually inserted into p_k to give p_{k+1}. With partial EM steps, we mean EM steps in which only the new component is updated rather than the complete

mixture. These partial EM steps can be performed in time that is not dependent on the number of components in the current mixture.

The VL has two drawbacks. First, using N candidates in each step results in a time complexity $O(N^2)$ for the search which is unacceptable in many applications. $O(N^2)$ computations are needed since the likelihood of every data point under every candidate component has to be evaluated. Second, we found experimentally that by using only candidates with covariance matrix $\sigma^2\mathbf{I}$ and small σ, the method keeps inserting small components in high density areas. Components with other covariance matrices that give greater improvement of the mixture log-likelihood are not among the candidates, nor are they found by the EM updates in step 3 following the component insertion.

An improved insertion procedure. Based on our experience with the VL method, we propose a new search heuristic for finding the optimal new mixture component. Two observations motivate the new search method. First, the size (i.e. the determinant of the covariance matrix) of the new component should in general be smaller than the size of the components in the current mixture. Second, as the existing mixture p_k contains more components, the search for the optimal new component should become more thorough. In our search strategy we account for both observations by (i) initializing the candidates based on a subset of the data that is currently captured by a single component and (ii) increasing the number of candidates linearly with k. In the next section we present experimental results that indicate that our new search procedure is faster and yields better results that the VL procedure. This is possible because the small set of candidates that is considered is tailored to the current mixture.

For each insertion problem, our method constructs m candidate components per existing mixture component. Based on the posterior distributions, we partition the data set $\mathbf{X}_N$ in k disjoint subsets $A_1, \ldots A_k$, with $A_i = \{\mathbf{x} \in \mathbf{X}_N : i = \arg\max_s p(s|\mathbf{x})\}$. If EM or a gradient based method is used in step 3 of the scheme given above, then the posteriors $p(s|\mathbf{x})$ are already available. From each set A_i m candidate components are constructed. In our experiments we used $m = 10$, but more candidates can be used at the expense of extra computation. Following (Smola and Schölkopf, 2000), the choice of m can be based on a confidence bound by specifying δ and ϵ: "With probability at least $1 - \delta$ the best split among m uniformly randomly selected splits is among the best ϵ fraction of all splits, if $m \geq \lceil \log \delta / \log (1 - \epsilon) \rceil$."

We want the candidate components to capture a subset of the data currently captured by one of the components in the mixture fitted so far. To this end we generate new candidates from A_i by selecting uniformly at random two data points $\mathbf{x}_l$ and $\mathbf{x}_r$ in A_i. Then, we partition A_i into two disjoint subsets A_{il} and A_{ir}, where A_{il} contains the elements of A_i that are closer to $\mathbf{x}_l$ than to $\mathbf{x}_r$ and A_{ir} contains the remaining elements. The mean and covariance of the sets A_{il} and A_{ir} are used as parameters for the two candidate components. The initial mixing weights for candidates generated from A_i are set

to $\pi_i/2$. This process is repeated until the desired number of candidates is obtained.

The initial candidates can be replaced easily by candidates that yield higher likelihood if they are included in the existing mixture. To obtain these better candidates we perform partial EM searches, starting from the initial candidates. These partial EM searches can be regarded as a lookahead, so we can judge the components on the log-likelihood that is achievable if we insert a candidate rather than on the log-likelihood that is achieved immediately after the insertion. Each iteration of the km partial updates takes $O(Nmk)$ computations, since we have to evaluate the likelihood of each datum under each of the mk candidate components. Below we discuss how we can reduce the amount of computations needed by a factor k. We stop the partial updates if the change in log-likelihood of the resulting $(k+1)$-component mixtures drops below some threshold or if some maximal number of iterations is reached.

Speeding up the partial EM searches. In the partial EM steps we have to maximize the log-likelihood function of a two component mixture $p_{k+1} = (1-\alpha)p_k + \alpha\phi$. The first component is the mixture p_k which we keep fixed and the second component is the new component ϕ which has a mixing weight α. In order to achieve $O(mN)$ time complexity for the partial EM searches initiated at the km initial candidates, we constrain the responsibilities used in the partial EM steps. When doing partial EM steps for a component originating from existing component i we constrain the responsibility of the new component for points $\mathbf{x} \notin A_i$ to be zero.

The usual EM update equations for a MoG (3.11) can now be simplified when updating a candidate generated from A_i. Since we are in a two-component mixture case, we write the responsibility for the new component as q_n and the responsibility for the old mixture is thus $(1-q_n)$. The simplified updates are:

$$q_n \leftarrow \frac{\alpha\phi(\mathbf{x}_n; \boldsymbol{\mu}, \boldsymbol{\Sigma})}{(1-\alpha)p_k(\mathbf{x}_n) + \alpha\phi(\mathbf{x}_n; \boldsymbol{\mu}, \boldsymbol{\Sigma})}, \tag{3.28}$$

$$\alpha \leftarrow \frac{1}{N}\sum_{\mathbf{x}_n \in A_i} q_n, \tag{3.29}$$

$$\boldsymbol{\mu} \leftarrow \frac{1}{\alpha N}\sum_{\mathbf{x}_n \in A_i} q_n \mathbf{x}_n, \tag{3.30}$$

$$\boldsymbol{\Sigma} \leftarrow \frac{1}{\alpha N}\sum_{\mathbf{x}_n \in A_i} q_n (\mathbf{x}_n - \boldsymbol{\mu})(\mathbf{x}_n - \boldsymbol{\mu})^\top. \tag{3.31}$$

Note that these updates only involve the data in A_i. Thus the computational cost of updating a candidate is now proportional to the number of points in A_i rather than N. Therefore, we can update k candidates, one based on each A_i, in time proportional to N rather than kN.

Using the constrained responsibilities, the partial EM steps optimize a lower bound on the data log-likelihood. The constrained responsibilities for points outside A_i will not match the posterior on the new component in general, thus after the E-step the EM lower bound corresponding to the partial EM algorithm will not equal the data log-likelihood. If we use $p(k+1|\mathbf{x})$ to denote the posterior on the new component, then after the E-step the lower bound on the data log-likelihood $\mathcal{L}_{k+1}$ can be written as:

$$\mathcal{L}_{k+1} \geq \mathcal{L}_{k+1} + \sum_{\mathbf{x} \notin A_i} \log(1 - p(k+1|\mathbf{x})) = \sum_{\mathbf{x} \notin A_i} \log p_k(\mathbf{x}) + \sum_{\mathbf{x} \in A_i} \log p_{k+1}(\mathbf{x}). \quad (3.32)$$

The lower bound becomes tight when, for data outside A_i, the posterior on the new component approaches zero. Thus, for data sets containing several widely separated clusters the speed-up is not expected to impact the obtained results.

Time Complexity Analysis. The total runtime of the algorithm is $O(k^2N)$ (or $O(kmN)$ if $k < m$). This is due to the updates of the mixtures p_i, which cost $O(Ni)$ computations each if we use the EM algorithm for these updates. Summing over all mixtures we obtain $O(k^2N)$. This is a factor k slower than the standard EM procedure. The runtime measurements in our experiments confirm this analysis. The new method performed on average about $k/2$ times slower than the standard EM algorithm. However, if we use a single run of the normal EM to learn mixtures consisting of $1, \ldots, k$ components, e.g. for determining the number of components that should be used according to some model selection criterion, then the runtime would also be $O(nk^2)$. Note that if we do not use the aforementioned speed-up, the runtime would become $O(k^2mn)$ since in that case the component allocation step for the i-th mixture component would cost $O(imn)$. This is a factor m more than the preceding EM steps, hence the total runtime increases by a factor m.

3.2.3 Related work

Several other approaches to mixture modelling that fit in the general greedy learning scheme of Section 3.2.1 have been proposed (Böhning, 1995; DerSimonian, 1986; Lindsay, 1983; Wu, 1978). The Vertex Direction Method (VDM) (Lindsay, 1983; Wu, 1978) is similar to our method, although VDM does not perform the intermediate complete mixture updates of step 3 of the greedy learning scheme. VDM makes use of the directional derivative $D_{p_k}(\phi)$ of the data log-likelihood for the current mixture p_k, where

$$D_{p_k}(\phi) = \lim_{\alpha \to 0} [\mathcal{L}(\mathbf{X}_n, (1-\alpha)p_k + \alpha\phi) - \mathcal{L}(\mathbf{X}_n, p_k)]/\alpha.$$

VDM proceeds by selecting $\phi^* = \arg\max_\phi D_{p_k}(\phi)$ and inserting this in the mixture with a factor α^* such that $\alpha^* = \arg\max_\alpha \{\mathcal{L}(\mathbf{X}_n, (1-\alpha)p_k + \alpha\phi^*\}$. Using the directional derivative at the current mixture can be seen as an instantiation of step 2 of the general scheme.

The optimization over both ϕ and α is replaced by (i) an optimization over $D_{p_k}(\phi)$ followed by (ii) an optimization over α. Finding the maximizer of the directional derivative ϕ^* is typically implemented by gridding the parameter space, which becomes infeasible for high dimensional spaces. Note that by moving in the direction of maximum $D_{p_k}(\phi)$ does not guarantee that we move in the direction of maximum improvement of log-likelihood if we optimize over α subsequently.

The split-and-merge EM (SMEM) algorithm (Ueda et al., 2000) applies split and merge operations to locally optimal mixtures found by the EM algorithm. In each split and merge operation three mixture components, r, s and t, are replaced by new components, r', s' and t'. Components r and s are merged into r' by letting the responsibilities for each data point of r' be the sum of the responsibilities of r and s. Component t is split by initializing the parameters of s' and t' with slight random perturbations of the parameters of t. The split and merge operations constitute jumps in the parameter space that allow the algorithm to jump from one local optimum to a region in the parameter space where the EM algorithm will converge to a different local optimum. By again applying the EM algorithm a potentially better optimum is found.

SMEM and our greedy approach can be combined. Split and merge operations can be incorporated in the greedy algorithm by checking after each component insertion (regarded as a split) whether some component s can be removed (regarded as a merge) such that the resulting log-likelihood is greater than the likelihood before the insertion. If so, component s is removed and the algorithm continues, having performed a split-and-merge step. If not, the algorithm just proceeds as usual having performed an insertion step. However, in experimental studies on synthetic data we found that this combination gave hardly any improvement over the individual methods. An important benefit of our method over SMEM is that our algorithm produces a sequence of mixtures that can be used to perform model complexity selection as the mixtures are learned. We also note that our algorithm executes faster than SMEM, which has a runtime of $O(nk^3)$. In our experiments we found SMEM to execute 2 to 8 times slower than our algorithm.

Bayesian methods are also used to learn a MoG. These methods approximate the posterior distribution over mixture models given the data. The posterior is defined using a prior distribution $p(\boldsymbol{\theta})$ over the mixture models, i.e. the parameters, and Bayes rule.

Methods in a sense opposite to the greedy approach to mixture learning are proposed in (Brand, 1999; Figueiredo and Jain, 2002). The idea is to start with a large number of mixture components and to successively remove components with small mixing weights. In both cases the 'pruning' criterion is based on the introduction of a prior distribution $p(\boldsymbol{\theta})$ on the mixing weights of the components; a Dirichlet prior in (Figueiredo and Jain, 2002) and an prior favoring mixing proportions with low entropy in (Brand, 1999). The pruning criterion can be included in an EM algorithm now aiming at maximizing $p(\mathbf{X}_N|\boldsymbol{\theta})p(\boldsymbol{\theta})$ rather than just $p(\mathbf{X}_N|\boldsymbol{\theta})$, yielding different update rules for the mixing proportions only. Comparing the greedy approach with the pruning approach, we note that the greedy approach does not need an initialization scheme since the optimal one-

component mixture can be found analytically. Furthermore, in most EM iterations the pruning approaches need to update the parameters of many components (more than the final number). The greedy approach, however, updates in most EM iterations fewer components than the final number, and thus requires fewer updates in total.

In (Richardson and Green, 1997) the posterior over mixture models is approximated by a repeated sampling using a reversible jump Markov Chain Monte Carlo (RJMCMC) method (Green, 1995). In the limit of an infinite sample, the obtained sample corresponds to a sample from the true posterior. The RJMCMC method allows jumps between parameter spaces of different dimensionality, i.e. parameter spaces for mixtures consisting of differing number of components). However, the experimental results reported in (Richardson and Green, 1997) indicate that such sampling methods are rather slow as compared to constructive maximum likelihood algorithms. It is reported that about 160 'sweeps' (sampling of all model parameters) per second are performed on a SUN Sparc4 workstation. The experiments involve 200.000 sweeps, which yields about 20 minutes runtime. Although it is remarked that the 200.000 sweeps are not needed for reliable results, this contrasts sharply with the 2.8 seconds and 5.7 seconds runtime (allowing respectively about 480 and 960 sweeps) of the standard EM and our greedy EM in a similar experimental setting also performed on a SUN Sparc4 workstation.

Simulated annealing techniques can be used to find the (global) maximum a-posteriori mixture model if an appropriate cooling schedule is used (Kirkpatrick et al., 1983). However, cooling schedules for which theoretical results guarantee convergence to the global optimum are impractically slow (Geman and Geman, 1984). Alternatively, faster deterministic annealing methods (Rose, 1998) can be used but these are not guaranteed to identify the global optimum.

The variational Baysian (VB) learning scheme (Beal and Ghahramani, 2003) also approximates the posterior distribution over parameters, but does so with analytic rather than sample-based approximations. The analytic approximation is computationally much more attractive than sample based approximations. In fact, the VB algorithm requires hardly more computation than the standard EM algorithm. In (Ghahramani and Beal, 2000) the authors combine their VB algorithm with a greedy algorithm to learn MoG and use the results of the VB algorithm to select the number of mixture components and the structure of the covariance matrices in the MoG. The component insertion procedure used is this work is similar to ours. Since we encountered this work only recently we do not have experimental results comparing these methods. However, we expect similar performance due to the similarity of the approaches.

3.2.4 Experimental results

In this section we present results of two experiments. The experiments compare the performance of our new greedy approach with k-means initialized EM, the VL method,

and SMEM. To initialize the EM algorithm with k-means, we initialize the cluster centers with randomly selected data points and then apply the k-means algorithm until convergence. The means and covariance matrices of the clusters found by k-means are then used to initialize the parameters of the MoG. The mixing weights are initialized as the fraction of the data in each cluster. The SMEM algorithm was also initialized with the k-means algorithm.

We present the results of two experiments. In the first experiment we apply the different algorithms to data generated from MoGs. This allows us to asses how the performance differences depend on several characteristics of the generated data. In the second experiment we used 50 dimensional data derived from images of textured surfaces. In this experiment the data is in reality probably not Gaussian distributed, so using this data set we can see how the performance of the algorithms differs on realistic data as encountered in practice.

Synthetic data. In this experiment we generated data sets of 400 points in $\mathbb{R}^D$ for $D \in \{2, 5\}$. The data was drawn from MoGs with $k \in \{4, 6, 8, 10\}$ components and separation $c \in \{1, 2, 3, 4\}$. The separation c of a MoG is defined as (Dasgupta, 1999):

$$\forall_{i \neq j}: \quad \|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2 \geq c \cdot \max\left(\mathrm{Tr}\{\boldsymbol{\Sigma}_i\}, \mathrm{Tr}\{\boldsymbol{\Sigma}_j\}\right). \tag{3.33}$$

For each mixture configuration we generated 50 data sets. We allowed a maximum eccentricity, defined as the ratio between the largest and the smallest singular value of the covariance matrix, of 15 for each component. Also, for each generated mixture, we generated a test set of 200 points not presented to the mixture learning algorithm. In the comparison below, we evaluate the difference in log-likelihood of the test sets under the mixtures estimated with the different methods.

In Figure 3.2 we present experimental results comparing our greedy approach to VL, SMEM, and several runs of normal EM initialized with k-means. The average of the difference in log-likelihood on the test set is given for different characteristics of the generating mixture. The results show that the greedy algorithm performs comparable to SMEM and generally outperforms VL. Both greedy methods and SMEM outperform the k-means initialization.

Texture clustering. In this experiment the task is to cluster a set of image patches of $16 \times 16 = 256$ pixels. The patches are extracted from Brodatz texture images of 256×256 pixels, four of these textures are shown in Figure 3.3. The patches extracted from the different textures are quite different and therefore can be expected to constitute clusters in the 256 dimensional space of possible patches. Since different patches from the same texture differ, roughly speaking, from each other only in a limited number of degrees of freedom (translation, rotation, scaling and brightness), the clusters in the patch-space are also assumed to be confined to a low dimensional subspace that is spanned by these

Vlassis & Likas

D=2	c=1	2	3	4	D=5	c=1	2	3	4
k=4	0.03	0.00	0.01	0.01	k=4	0.05	0.02	-0.01	0.04
6	0.01	0.03	0.05	0.02	6	0.04	0.04	0.13	0.13
8	0.00	0.04	0.06	0.03	8	0.03	0.07	0.13	0.04
10	0.01	0.06	0.07	0.03	10	0.04	0.05	0.11	0.16

Best of k runs of k-means initialized EM

D=2	c=1	2	3	4	D=5	c=1	2	3	4
k=4	0.03	0.14	0.13	0.43	k=4	0.02	0.28	0.36	0.37
6	0.03	0.16	0.34	0.52	6	0.09	0.35	0.48	0.54
8	0.02	0.23	0.38	0.55	8	0.12	0.45	0.75	0.82
10	0.06	0.27	0.45	0.62	10	0.13	0.49	0.75	0.83

SMEM

D=2	c=1	2	3	4	D=5	c=1	2	3	4
k=4	-0.01	0.00	0.00	0.00	k=4	-0.02	0.01	0.00	0.00
6	-0.02	0.00	0.02	0.02	6	-0.06	-0.01	-0.01	0.00
8	-0.02	0.00	0.00	0.03	8	-0.05	-0.04	0.00	0.02
10	-0.00	0.00	0.03	0.04	10	-0.01	0.04	0.01	0.05

Distance to generating mixture

D=2	c=1	2	3	4	D=5	c=1	2	3	4
k=4	0.04	0.03	0.03	0.02	k=4	0.16	0.13	0.14	0.11
6	0.07	0.06	0.05	0.04	6	0.28	0.22	0.19	0.18
8	0.10	0.07	0.07	0.09	8	0.45	0.33	0.32	0.42
10	0.13	0.12	0.10	0.12	10	0.58	0.50	0.45	0.51

Figure 3.2: Detailed exposition of log-likelihood differences. The upper three tables give the log-likelihood of a method subtracted from the log-likelihood obtained using our greedy approach. The bottom table gives averages of the log-likelihood under the generating mixture minus the log-likelihood given by our method.

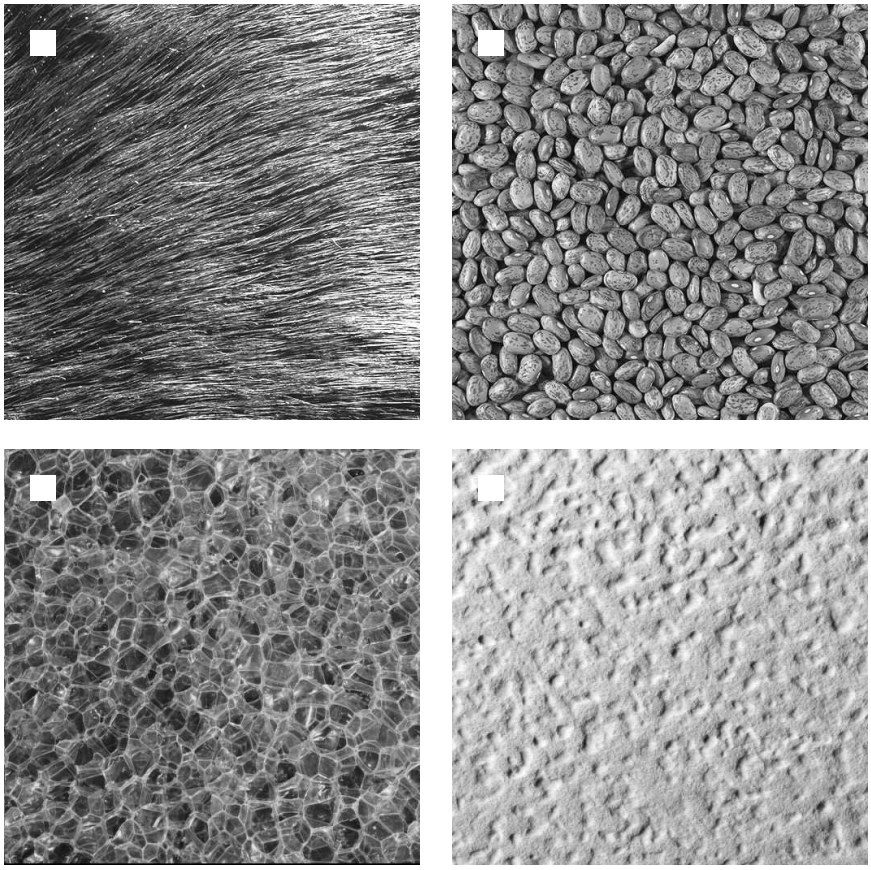


Figure 3.3: Several Brodatz textures. The white squares indicate the patch size.

degrees of freedom. Of course, the changes in the images as a result of making a small change in one of the underlying degrees of freedom should be sufficiently small in order for the patches of a single texture to form a compact cluster in the patch space.

Note that our aim is not to present MoGs as the appropriate approach for texture clustering. Methods based on image analysis techniques probably attain better results. We merely use the texture clustering problem as test case to compare the algorithms on a natural data set where the true clusters are probably not Gaussian distributed.

We conducted experiments where the number of textures from which patches were extracted ranged from $k = 2, \ldots, 6$. For each value of k, we created 100 data sets of $500k$ patches each by uniformly selecting at random 500 patches from each of k uniformly random selected textures. On each data set we fitted a MoG with as many components as different textures collected in the data set. We compared our greedy approach to the same methods as in the previous experiment.

To speed-up the experiment and to reduce the number of parameters that had to be estimated, we projected the data into a linear subspace with principal component anal-

k	2	3	4	5	6
greedy	0.20	0.34	0.48	**0.53**	**0.61**
k-means init	**0.19**	0.46	0.74	0.80	0.94
SMEM	0.21	**0.22**	**0.36**	0.56	0.68
VL	0.93	1.52	2.00	2.29	2.58
uniform	1	1.59	2	2.32	2.58

Figure 3.4: Average conditional entropy for different values of k for the greedy approach, k-means initialized EM and SMEM and VL. The bottom line gives the conditional entropy if the clusters are totally uninformative. Bold face is used to identify the method with minimum conditional entropy for each k.

ysis. We used the smallest number of principal components that could account for at least 80% of the data variance, but fixed the maximum number of dimensions to 50. Once the mixture model was learned, we clustered the patches by assigning each patch to the mixture component having the highest posterior probability for this patch. This yields a confusion matrix between the clusters and the textures. To evaluate the quality of the clusterings discovered by the different learning methods, we considered how informative the clusterings are on the texture label; we measured how predictable the texture label of a patch is, given the cluster of the class. This can be done using the conditional entropy (Cover and Thomas, 1991) of the texture label given the cluster label. The conditional entropy for two variables X and Y is defined as:

$$\mathcal{H}(X|Y) = -\sum_y p(Y=y) \sum_x p(X=x|Y=y) \log p(X=x|Y=y), \tag{3.34}$$

and measures the uncertainty in X (texture label) given the value of Y (cluster label). The joint probability $p(X,Y)$ is estimated from the confusion matrix.

In Figure 3.4 the conditional entropies (averaged over 50 experiments) are given.[2] The results of the greedy method and SMEM are roughly comparable, although the greedy approach was on average about four times faster. Both provide generally more informative segmentations than using the k-means initialization. Note that VL fails to produce informative clusters in this experiment. This is probably due to the high dimensionality of the data that renders candidate components with isotropic covariance rather unfit.

3.2.5 Conclusions

We proposed an efficient greedy method to learn mixtures of Gaussian densities that has runtime $O(k^2N + kmN)$ for N data points, a final number of k mixture components, and m candidates per component.

[2] The logarithm with base 2 was used to compute these entropies.

Our experiments have shown that our greedy algorithm generally outperforms the k-means initialized EM. Both on synthetic and natural data, SMEM performs comparably to our new method. An important benefit of the greedy method, both compared to SMEM and the standard EM algorithm, is the production of a sequence of mixtures. This obviates the need for initialization and facilitates model selection. As compared to VL we note: (i) The $O(N^2k^2)$ time complexity has been reduced by a factor N. (ii) The somewhat arbitrary choice of spherical candidate components with fixed variance has been replaced by a search for candidate components that depends on the data and the current mixture. (iii) Experiments suggest that if the methods yield different performance, then the new method generally outperforms VL.

3.3 A greedy approach to k-means clustering

In this section we propose a greedy approach to k-means clustering similar to the greedy approach for MoG learning presented in the previous section. The proposed clustering method is tested on well-known data sets and compares favorably to the k-means algorithm with random restarts.[3]

In the previous chapter we already discussed the k-means algorithm, one of the most popular clustering algorithms. Suppose we are given a data set $\mathbf{X}_N = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, $\mathbf{x}_n \in \mathbb{R}^D$. The k-clustering problem aims at partitioning this data set into k disjoint subsets (clusters) $A_1, \ldots, A_k$, such that a clustering criterion is optimized. The most widely used clustering criterion is the sum of the squared Euclidean distances between each data point $\mathbf{x}_n$ and the centroid $\boldsymbol{\mu}_s$ (cluster center) of the subset A_s which contains $\mathbf{x}_n$. This criterion is called clustering error and depends on the cluster centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$:

$$E(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k) = \sum_{s=1}^{k} \sum_{\mathbf{x}_n \in A_s} \|\mathbf{x}_n - \boldsymbol{\mu}_s\|^2. \tag{3.35}$$

Given some initialization of the $\boldsymbol{\mu}_s$, the k-means algorithm iterates between two steps analogous to the E-step and M-step of the EM algorithm. In the first step (E) each data point $\mathbf{x}_n$ is assigned to the cluster s given by $s = \arg\min_s \|\mathbf{x}_n - \boldsymbol{\mu}_s\|^2$. The second step (M) updates the cluster centers as the average of the associated data vectors: $\boldsymbol{\mu}_s \leftarrow \frac{1}{|A_s|} \sum_{\mathbf{x}_n \in A_s} \mathbf{x}_n$. It is easy to see that the iterations of these steps can not increase the error in (3.35). However, one can not guarantee that the k-means algorithm will converge to the optimal clustering. The main disadvantage of the method thus lies in its sensitivity to initial positions of the cluster centers (Pena et al., 1999). Therefore, in order to obtain near optimal solutions using the k-means algorithm, several runs must be scheduled differing in the initial positions of the cluster centers.

[3] The material of this section is largely drawn from (Likas et al., 2003).

Other approaches to deal with this problem have been developed that are based on stochastic global optimization methods, e.g. simulated annealing (Banddyopadhyay et al., 2001), genetic algorithms (Krishna and Murty, 1999), or eigenvector analysis of the data inner-product matrix (Zha et al., 2002). However, these techniques have not gained wide acceptance and in many practical applications the clustering method that is used is the k-means algorithm with multiple restarts (Jain et al., 1999). In this section we present the global k-means algorithm, a deterministic algorithm designed to reduce the sensitivity to initialization.

In the next subsection we present the global k-means algorithm and propose heuristics to reduce the computation time in Section 3.3.2. Experimental results comparing the global k-means algorithm to the normal k-means algorithm are presented in Section 3.3.3. Conclusions will be drawn in Section 3.3.4.

3.3.1 The global k-means algorithm

The global k-means clustering algorithm is a deterministic method that does not depend on initial parameter values and employs the k-means algorithm as a local search procedure. Instead of randomly selecting initial values for all cluster centers as is the case with most global clustering algorithms, the proposed technique proceeds in an incremental way attempting to optimally add one new cluster center at each stage.

More specifically, to solve a clustering problem with $k_{\max}$ clusters the method proceeds as follows. We start with one cluster, $k = 1$, and find its optimal position which corresponds to the centroid of the data set. In order to solve the problem with two clusters $(k = 2)$ we perform N executions of the k-means algorithm from the following initial positions of the cluster centers: the first cluster center is always placed at the optimal position for the problem with $k = 1$, while the second center at execution n is placed at the position of the data point $\mathbf{x}_n$ $(n = 1, \ldots, N)$. The best solution obtained after the N executions of the k-means algorithm is considered as the solution to the 2-clustering problem. Let $(\boldsymbol{\mu}_1(k), \ldots, \boldsymbol{\mu}_k(k))$ denote the final solution to the k-clustering problem. Once we have found the solution to the k-clustering problem, we try to find the solution to the $(k+1)$-clustering problem as follows: we perform N runs of the k-means algorithm with $(k+1)$ clusters where each run n starts from the initial state $(\boldsymbol{\mu}_1(k), \ldots, \boldsymbol{\mu}_k(k), \mathbf{x}_n)$. The best solution obtained from the N runs is considered as the solution to the $(k+1)$-clustering problem. By repeating this procedure we finally obtain a solution with $k_{\max}$ clusters having also found solutions to all k-clustering problems with $1 \leq k < k_{\max}$.

The latter characteristic can be advantageous in many applications where the aim is also to discover the 'correct' number of clusters. To achieve this, one has to solve the k-clustering problem for various numbers of clusters and then employ appropriate criteria for selecting the most suitable value of k (Milligan and Cooper, 1985). In this case the proposed method directly provides clustering solutions for all intermediate values of k,

thus requiring no additional computational effort.

The method requires N executions of the k-means algorithm for each value of k ($k = 1, \ldots, k_{max}$), resulting in a total runtime of $O(k_{max}^2 N^2)$. Depending on the available resources and the values of N and k_{max}, the algorithm may be an attractive approach, since, as experimental results indicate, the performance of the method is excellent. Moreover, as we will show later, there are several modifications that can be applied in order to reduce the computational load.

An implicit assumption seems to underly the algorithm: an optimal clustering solution with k clusters can be obtained by applying the k-means algorithm initialized with optimal positions for the $(k-1)$-clustering problem and the remaining k-th center placed at an appropriate position to be discovered. Although this assumption may seem valid at first glance, it was recently shown (Hansen et al., 2002) that in general this assumption does not hold for $k_{max} \geq 3$. Despite this negative result we find near optimal performance in our experiments; the solution obtained by the proposed method was at least as good as that obtained using numerous random restarts of the k-means algorithm.

3.3.2 Speeding up execution of global k-means

In this section we propose two heuristics to reduce the computational load of the global k-means algorithm without significantly affecting the quality of the solution. These heuristics can be used separately from each other or together.

The fast global k-means algorithm. The fast global k-means algorithm constitutes a straightforward method to accelerate the global k-means algorithm. The difference lies in the way a solution for the k-clustering problem is obtained, given the solution of the $(k-1)$-clustering problem. For each of the N initial states $(\boldsymbol{\mu}_1(k-1), \ldots, \boldsymbol{\mu}_{(k-1)}(k-1), \mathbf{x}_n)$ we do not execute the k-means algorithm until convergence to obtain the final clustering error. Instead, we compute the clustering error E_n that is obtained when only a single assignment step is performed after $\mathbf{x}_n$ is added to the set of cluster means. Clearly, this is an upper bound on the clustering error if we would continue the k-means algorithm until convergence. The difference between the clustering error $E_{(k-1)}$ before inserting $\mathbf{x}_n$ and the clustering error E_n after the first assignment step is given by:

$$E_{(k-1)} - E_n = \sum_{i=1}^{N} \max(d_{k-1}^i - \|\mathbf{x}_n - \mathbf{x}_i\|^2, 0), \tag{3.36}$$

where d_{k-1}^i is the squared distance between $\mathbf{x}_i$ and the closest center among the $k-1$ cluster centers obtained so far. We then initialize the position of the new cluster center at $\mathbf{x}_n$ that minimizes E_n, or equivalently that maximizes (3.36), and execute the k-means algorithm to obtain the solution with k clusters. Experimental results presented in the

next section suggest that using the data point that minimizes this bound leads to results almost as good as those provided by the global k-means algorithm.

Initialization with kd-trees. A second possibility to reduce the amount of computation needed by global k-means is to use a small set of appropriately selected potential insertion positions rather than to consider all data points as potential insertion locations. Below, we discuss using a kd-tree to select such a small set of insertion locations.

A kd-tree (Bentley, 1975; Sproull, 1991) is a multi-dimensional generalization of the standard one-dimensional binary search tree that facilitates storage and search over multi-dimensional data sets. A kd-tree defines a recursive partitioning of the data space into half-spaces. To each node of the tree corresponds a half-space of the original data space and the subset of the data contained in this half-space. Each non-terminal node has two successors, each of them associated with one of the two half-spaces obtained from the partitioning of the parent half-space using a cutting hyper-plane. The kd-tree structure was originally used for speeding up distance-based search operations like nearest neighbors queries, range queries, etc. Below we use a variation of the original kd-tree proposed in (Sproull, 1991). There, the cutting hyper-plane is defined as the plane that is perpendicular to the direction of the principal component of the data points corresponding to each node.

In principle the leaves of the tree contains single points. However, for practical purposes the recursion is usually terminated if a node (called a bucket) is created containing less than a pre-specified number of points b (called the bucket size) or if a pre-specified number of buckets have been created. It turns out that the buckets of the kd-tree are very useful to determine potential insertion locations. The idea is to use the bucket centers, which are fewer in number than the data points if a bucket size $b > 1$ is used, as potential insertion locations for the (fast) global k-means algorithm.

From experiments we learned that restricting the insertion locations for the (fast) global k-means to those given by the kd-tree (instead of using all data points) does not significantly degrade performance if we consider a sufficiently large number of buckets in the kd-tree. In general, using more buckets than the final number of clusters gave results comparable to using all data points as insertion locations.

3.3.3 Experimental results

We have tested the proposed clustering algorithms on several well-known data sets, namely the iris data set (Blake and Merz, 1998), the synthetic data set from (Ripley, 1996), and the image segmentation data set from (Blake and Merz, 1998). In all data sets we conducted experiments for the clustering problems obtained by considering only feature vectors and ignoring class labels. The iris data set contains 150 four dimensional

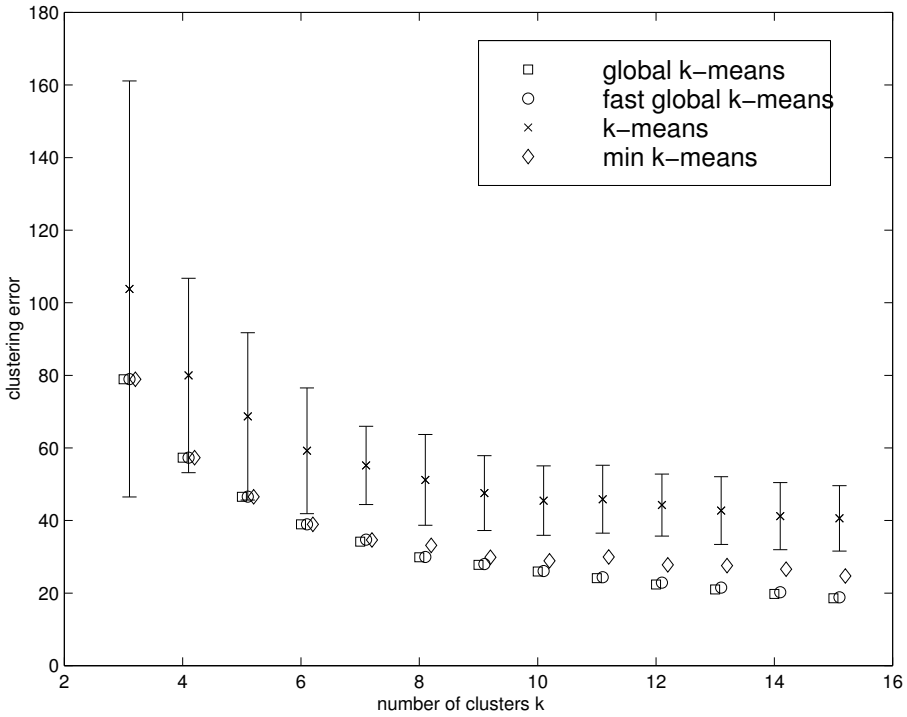


Figure 3.5: Performance results for the iris data set.

data points, the synthetic data set contains 250 two dimensional data points. The image segmentation data set contains 210 six dimensional data points obtained through a PCA projection of the original 18-dimensional data points. The quality of the obtained solutions was evaluated in terms of the values of the final clustering error.

For each data set we conducted the following experiments:

- one run of the global k-means algorithm for $k = 15$.
- one run of the fast global k-means algorithm for $k = 15$.
- the k-means algorithm for $k = 1, \ldots, 15$. For each value of k, the k-means algorithm was executed N times (where N is the number of data points) starting from random initial positions for the k centers. We compute the minimum and average clustering error as well as its standard deviation.

For each of the three data sets the experimental results are displayed in Figures 3.5, 3.6 and 3.7 respectively. Each figure displays the clustering error as a function of the number of clusters. It is clear that the global k-means algorithm is very effective, providing in all cases solutions of equal or better quality with respect to the k-means algorithm. The fast version of the algorithm also provides solutions comparable to those obtained by the original method while executing significantly faster.

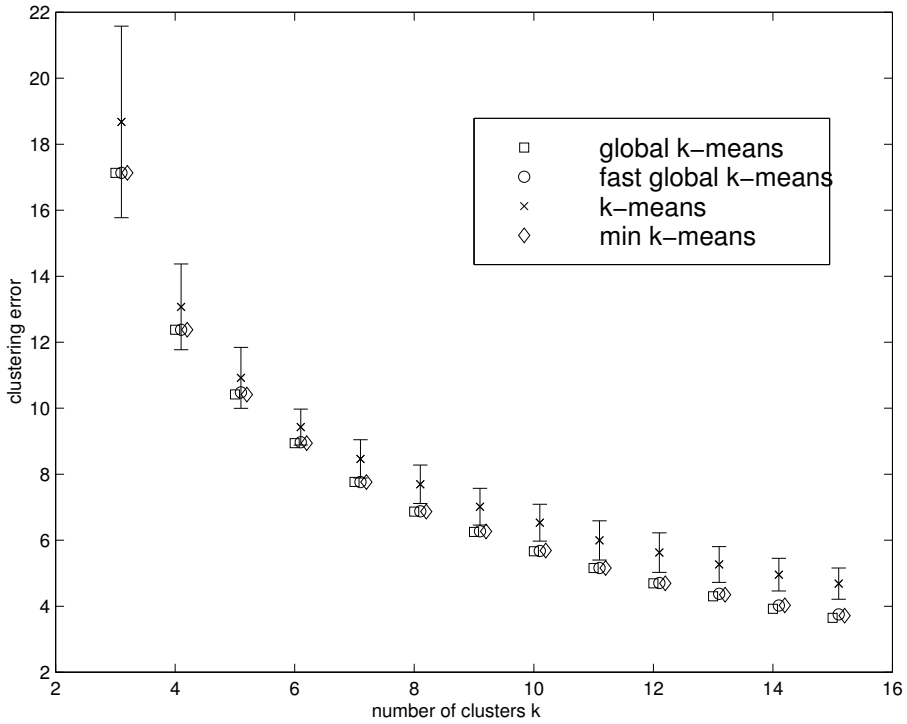


Figure 3.6: Performance results for the synthetic data set.

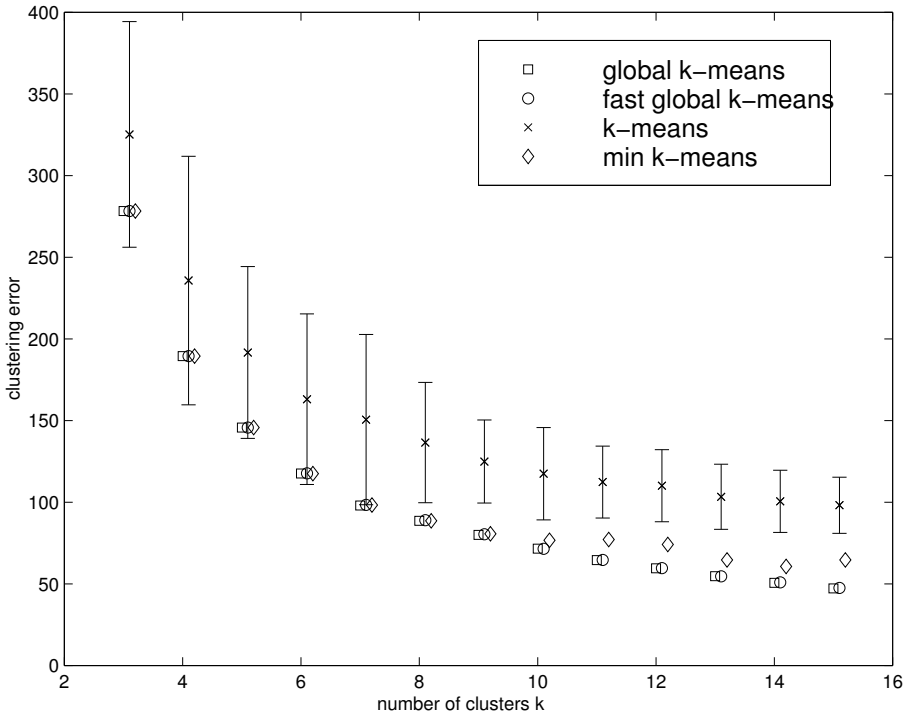


Figure 3.7: Performance results for the image segmentation data set.

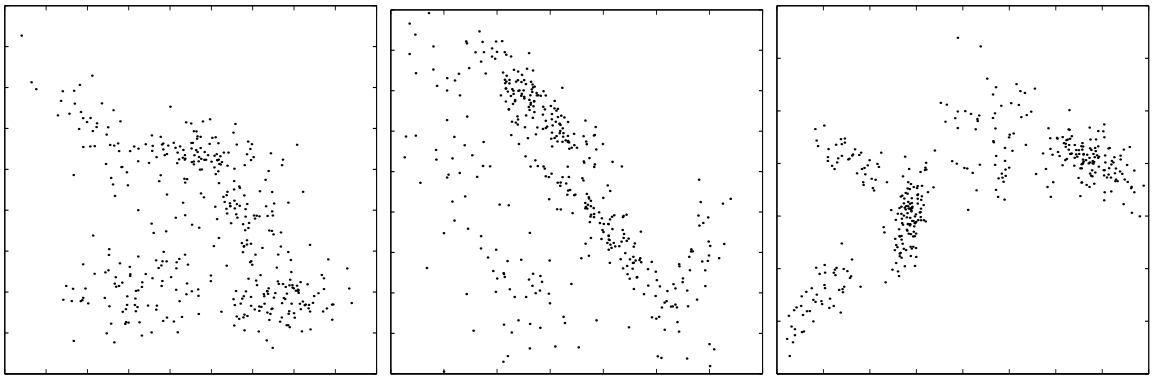


Figure 3.8: Example data sets with $k = 5, D = 2$ and separation (left to right) $\frac{1}{2}$, $\frac{3}{4}$, and 1.

Synthetic data. Here we provide more extensive comparative experimental results using artificially created data sets. The purpose is to compare the randomly initialized k-means algorithm with the fast global k-means algorithm that uses the top $2k$ nodes of the corresponding kd-tree as candidate insertion locations.

The data have been drawn from randomly generated Gaussian mixtures. We varied the number of mixture components k, the dimensionality D of the data space and the separation c, as defined in (3.33), between the sources of the mixture. In our experiments we considered mixtures having a separation in the interval $[\frac{1}{2}, 1]$, these values correspond to clusters that overlap to a large extent. The number of data points in each data set was $50k$, where k is the number of sources. Some example data sets are shown in Fig. 3.8.

We considered 120 problems corresponding to all combinations of the following values: $k = \{2, 7, 12, 17, 22\}$, $D = \{2, 4, 6, 8\}$, $c = \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. For each problem 10 data sets were created. On every data set, the 'greedy' (i.e. fast global k-means with kd-tree initialization) algorithm was applied first. Then the randomly initialized k-means algorithm was applied as many times as possible in the runtime of the greedy algorithm. To evaluate the quality of the solutions found for a specific data set, first the mean clustering error $\bar{E}$ is computed for the runs performed using the k-means algorithm. The results are displayed in Fig. 3.9 and Fig. 3.10.

Both the minimum clustering error value encountered during the runs with the k-means algorithm and the error obtained with the greedy algorithm are given relative to the mean error $\bar{E}$. The minimum error among the k-means runs is given in the 'min.' columns as $(1 - \min/\bar{E}) \times 100$, thus this value says how much better the minimum error is as a percentage of the mean error. The standard deviation σ of the errors is given in the σ column as $\sigma/\bar{E} \times 100$. The 'gr.' column provides the clustering error E_{gr} obtained with the greedy algorithm as: $(1 - E_{gr}/\bar{E}) \times 100$, this number shows how much better E_{gr} was than $\bar{E}$. Finally, the 'trials' column indicates how many runs of the k-means algorithm were allowed in the runtime of the greedy algorithm. Each row of the table displays the averaged results over 10 data sets constructed for the specific

		$D = 2$			$D = 4$		
k	c	gr.	min.	trials	gr.	min.	trials
2	0.5	0.8	**1.0**	4.3	1.6	**2.2**	3.4
2	0.6	1.1	**1.4**	2.7	0.4	**0.7**	2.9
2	0.7	-0.8	**0.1**	3.5	-0.6	**0.2**	2.7
2	0.8	0.2	**0.5**	2.9	0.5	0.5	2.9
2	0.9	0.3	**0.5**	3.4	1.2	**2.1**	3.0
2	1.0	-0.5	**0.3**	3.9	0.5	**0.7**	3.2
7	0.5	**2.7**	2.1	3.6	2.3	**2.7**	3.9
7	0.6	**16.9**	14.7	4.3	1.7	**2.3**	4.1
7	0.7	**3.8**	3.7	3.4	5.3	5.3	4.1
7	0.8	**11.2**	9.9	4.2	**4.9**	4.7	4.3
7	0.9	**17.1**	15.9	4.5	**6.6**	6.3	3.7
7	1.0	**9.9**	8.1	4.4	**7.6**	7.2	4.1
12	0.5	**8.5**	5.9	4.3	**3.3**	2.4	4.4
12	0.6	**14.5**	12.2	4.5	**4.1**	2.8	4.3
12	0.7	**15.7**	13.4	4.5	**12.7**	7.1	4.4
12	0.8	**25.1**	15.4	4.6	**11.1**	8.5	4.9
12	0.9	**20.6**	14.2	4.6	**16.4**	12.3	4.6
12	1.0	**24.9**	18.1	4.1	**20.8**	15.8	5.8
17	0.5	**16.4**	11.1	4.1	**5.3**	3.6	4.3
17	0.6	**17.5**	12.6	4.1	**4.4**	3.6	4.2
17	0.7	**25.4**	18.7	4.9	**8.5**	6.3	4.4
17	0.8	**24.2**	17.6	5.0	**10.7**	6.3	4.9
17	0.9	**46.9**	29.0	5.4	**16.8**	10.7	5.0
17	1.0	**53.2**	37.2	5.8	**25.0**	19.1	5.6
22	0.5	**23.8**	18.5	4.7	**4.8**	3.5	4.6
22	0.6	**23.9**	14.3	5.1	**10.0**	6.3	4.9
22	0.7	**29.1**	18.6	5.1	**12.5**	8.9	5.1
22	0.8	**41.6**	29.0	5.2	**8.5**	5.1	5.2
22	0.9	**42.4**	30.5	5.9	**23.0**	14.4	5.2
22	1.0	**47.7**	34.7	5.8	**26.3**	16.7	5.3

Figure 3.9: Experimental results on synthetic data sets with $D = 2$ and $D = 4$.

		$D = 6$			$D = 8$		
k	c	gr.	min.	trials	gr.	min.	time
2	0.5	0.5	**0.9**	3.2	0.1	**0.6**	2.7
2	0.6	0.4	**0.5**	3.3	1.5	1.5	3.2
2	0.7	**0.9**	0.4	2.5	2.2	**2.6**	2.6
2	0.8	0.5	**0.6**	3.0	1.8	1.8	3.7
2	0.9	0.3	0.3	3.1	-0.0	**0.1**	2.6
2	1.0	0.8	**0.9**	5.1	0.7	**1.1**	3.3
7	0.5	2.3	**2.4**	3.8	**3.4**	3.1	3.9
7	0.6	**4.4**	3.5	4.4	**2.9**	2.3	3.5
7	0.7	**4.9**	3.9	3.9	**3.8**	3.3	4.1
7	0.8	**5.7**	5.5	4.6	**5.0**	4.6	5.2
7	0.9	**4.6**	4.3	4.3	6.9	**7.3**	4.3
7	1.0	**6.9**	7.3	4.7	**9.2**	8.6	4.9
12	0.5	**3.6**	2.9	4.1	**2.5**	1.6	3.8
12	0.6	**4.8**	3.0	4.0	**5.7**	3.1	4.4
12	0.7	**5.1**	2.7	4.3	**4.5**	2.7	4.1
12	0.8	**5.3**	4.2	4.4	**5.7**	4.8	5.0
12	0.9	**8.6**	6.7	4.2	**10.0**	7.5	5.1
12	1.0	**12.3**	8.1	5.3	**9.2**	6.2	5.5
17	0.6	**5.0**	2.8	3.9	**4.9**	3.8	4.2
17	0.7	**5.4**	3.2	4.9	**6.3**	3.4	5.1
17	0.8	**8.8**	5.2	5.0	**8.4**	5.6	5.1
17	0.9	**11.6**	7.0	5.2	**14.4**	8.8	5.6
17	1.0	**22.2**	12.6	5.9	**15.9**	8.7	5.6
22	0.5	**2.8**	1.8	4.7	**2.2**	1.2	4.3
22	0.6	**4.8**	2.5	4.7	**3.0**	2.0	4.8
22	0.7	**9.3**	4.6	4.9	**9.3**	6.2	5.5
22	0.8	**13.0**	7.4	5.5	**11.5**	7.2	5.3
22	0.9	**14.7**	7.1	5.4	**19.5**	11.2	6.1
22	1.0	**21.6**	12.6	5.8	**16.2**	9.4	5.3

Figure 3.10: Experimental results on synthetic data sets with $D = 6$ and $D = 8$.

values of k, D and c. We used bold values in the tables to indicate whether the clustering error of the greedy algorithm or the minimum error achieved for the runs with the k-means algorithm was the smallest.

The results show that the benefit of the greedy method becomes larger if there are more clusters, the separation becomes larger, or the dimensionality gets smaller. In almost all cases the greedy algorithm gives better results. In cases where the greedy method is not superior, both methods yield results relatively close to the average k-means result. It is interesting that the number of trials allowed for the random k-means algorithm grows only slowly as the number of clusters increases.

3.3.4 Conclusions

We have presented the global k-means clustering algorithm, a deterministic clustering method providing excellent results in terms of the clustering error criterion. The method is independent of any starting conditions and compares favorably to the k-means algorithm with multiple random restarts. The deterministic nature of the method is particularly useful in cases where the clustering method is used either to specify initial parameter values for other methods (for example training of radial basis function networks (Webb, 2002)) or as a module in a more complex system. In such cases other parts of the system can be designed and tested while there is no risk of obtaining bad results due to occasional aberrant behavior of the clustering module. Another advantage of the proposed technique is that in order to solve the k-clustering problem, all intermediate l-clustering problems are also solved for $l < k$. This may prove useful in applications where we seek the actual number of clusters and the k-clustering problem needs to be solved for several values of k. We also proposed two modifications of the method that reduce the computational load without significantly affecting solution quality.

3.4 Accelerating the EM algorithm for large data sets

In the previous sections we considered the EM algorithm for Gaussian mixtures and the k-means algorithm. The run-time per iteration of both algorithms is $O(Nk)$ for N data items and k clusters, this limits their usefulness in large scale applications with many data points and many clusters. Recently several authors (Moore, 1999; Moore and Pelleg, 1999; Kanungo et al., 2002; Alsabti et al., 1998) proposed speed-ups of these algorithms based on analysis of sufficient statistics of large chunks of data. The idea is to use geometrical reasoning to determine that for chunks of data a particular prototype is the closest (k-means) or the posterior on mixture components hardly varies in the chunk of data (EM for MoG). Cached sufficient statistics are then used to perform the update step of the algorithms.

In this section we present a constrained EM algorithm that can be used to speed-up large-scale Gaussian mixture modelling. Contrary to related work, our algorithm is in each step guaranteed to increase a lower bound on the data log-likelihood. We derive closed-form and efficiently computable optimal assignments of chunks of data to mixture components (E-step) and parameter estimate update equations (M-step). In our algorithm sufficient statistics of chunks of data are stored in the nodes of a kd-tree. The computational cost of the EM steps is independent of the number of data points and is linear in the number of outer nodes of the tree since both steps only use the sufficient statistics stored in these nodes. Furthermore, our algorithm allows for arbitrary partitions, while existing techniques were limited to use relatively fine partitions of the data to ensure stability of the algorithms. Thus the proposed framework allows more freedom in designing new speed-up algorithms. We present experimental results that illustrate the validity of our approach.[4]

The rest of this section is organized as follows: in Section 3.4.1 we present our accelerated EM algorithm. Then, in Section 3.4.2 we compare our work with related work. In Section 3.4.3 we describe our experiments on speeding up Gaussian mixture learning and end with conclusions in Section 3.4.4.

3.4.1 An accelerated EM algorithm for Gaussian mixtures

We have seen that the EM algorithm iteratively maximizes the lower bound $\mathcal{F}(\boldsymbol{\theta}, q)$ on the data log-likelihood $\mathcal{L}(\mathbf{X}_N, \boldsymbol{\theta})$ with respect to q and $\boldsymbol{\theta}$. The bound is a function of mixture parameters $\boldsymbol{\theta}$ and a set of distributions $q = \{q_1, \ldots, q_N\}$ where each $q_n(s)$ corresponds to a data point $\mathbf{x}_n$ and defines an arbitrary discrete distribution over mixture components indexed by s. Recall that this lower bound can be written in two forms:

$$\mathcal{F}(\boldsymbol{\theta}, q) = \sum_{n=1}^{N} \left[\log p(\mathbf{x}_n; \boldsymbol{\theta}) - \mathcal{D}(q_n(s) \| p(s|\mathbf{x}_n; \boldsymbol{\theta}))\right] \tag{3.37}$$

$$= \sum_{n=1}^{N} \sum_{s=1}^{k} q_n(s) \left[\log p(\mathbf{x}_n, s; \boldsymbol{\theta}) - \log q_n(s)\right]. \tag{3.38}$$

The dependence of p on $\boldsymbol{\theta}$ is throughout assumed, although not always written explicitly. As discussed in Section 3.1.2, the EM algorithm can be modified by constraining the allowed distributions over hidden variables, i.e. the responsibilities $q_n(s)$. Rather than iteratively increasing $\mathcal{L}$, such a constrained, or variational, EM algorithm will iteratively increase a lower bound on $\mathcal{L}$. The idea of our accelerated EM algorithm is to assign equal responsibilities to chunks of data points that are nearby in the input space.

Consider a partitioning of the data space into a collection of non-overlapping cells $\mathcal{A} = \{A_1, \ldots, A_m\}$, such that each point in the data set belongs to a single cell. To all points in

[4] The material of this section is largely drawn from (Verbeek et al., 2003d).

a cell $A \in \mathcal{A}$ we assign the same responsibility distribution $q_A(s)$ which we can compute in an optimal way as we show next. Note from (3.38) that the objective function $\mathcal{F}$ can be written as a sum of local parts $\mathcal{F} = \sum_{A \in \mathcal{A}} \mathcal{F}_A$, one per cell. If we impose $q_n(s) = q_A(s)$ for all data points $\mathbf{x}_n \in A$, then the part of $\mathcal{F}$ corresponding to a cell A reads:

$$\mathcal{F}_A = \sum_{\mathbf{x}_n \in A} \sum_{s=1}^{k} q_A(s) \left[\log p(\mathbf{x}_n|s) + \log p(s) - \log q_A(s)\right] \tag{3.39}$$

$$= |A| \sum_{s=1}^{k} q_A(s) \left[\log p(s) - \log q_A(s) + \langle \log p(\mathbf{x}|s) \rangle_A\right], \tag{3.40}$$

where $\langle \cdot \rangle_A$ denotes average over all points in cell A. Setting the derivatives of $\mathcal{F}_A$ w.r.t. $q_A(s)$ to zero we find the distribution $q_A(s)$ that maximizes $\mathcal{F}_A$:

$$q_A(s) \propto p(s) \exp\langle \log p(\mathbf{x}|s) \rangle_A. \tag{3.41}$$

Such an optimal distribution can be computed separately for each cell $A \in \mathcal{A}$, and only requires computing the average conditional log-likelihood of the points in A.

We now show that it is possible to efficiently compute (i) the optimal $q_A(s)$ for each cell A in the E-step and (ii) the new values of the unknown mixture parameters in the M-step, if some statistics of the points in each cell A are cached in advance. The averaging operation in (3.40) can be written[5]:

$$\langle \log p(\mathbf{x}|s) \rangle_A = \frac{1}{|A|} \sum_{\mathbf{x}_n \in A} \log p(\mathbf{x}_n|s) \tag{3.42}$$

$$= -\frac{1}{2} \left[\log |\boldsymbol{\Sigma}_s| + \frac{1}{|A|} \sum_{\mathbf{x}_n \in A} (\mathbf{x}_n - \boldsymbol{\mu}_s)^\top \boldsymbol{\Sigma}_s^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_s) \right] \tag{3.43}$$

$$= -\frac{1}{2} \left[\log |\boldsymbol{\Sigma}_s| + \boldsymbol{\mu}_s^\top \boldsymbol{\Sigma}_s^{-1} \boldsymbol{\mu}_s + \mathrm{Tr}\{\boldsymbol{\Sigma}_s^{-1} \langle \mathbf{x}\mathbf{x}^\top \rangle_A\} - 2\boldsymbol{\mu}_s^\top \boldsymbol{\Sigma}_s^{-1} \langle \mathbf{x} \rangle_A \right]. \tag{3.44}$$

From this we see that the mean $\langle \mathbf{x} \rangle_A$ and average outer product matrix $\langle \mathbf{x}\mathbf{x}^\top \rangle_A$ of the points in A are sufficient statistics for computing the optimal responsibilities $q_A(s)$. The same statistics can be used for updating the mixture parameters $\boldsymbol{\theta}$. If we set the derivatives of $\mathcal{F}$ w.r.t. $\boldsymbol{\theta}$ to zero we find the update equations similar to those given in (3.11) for the standard EM algorithm:

$$p(s) \leftarrow \frac{|A|}{n} \sum_{A \in \mathcal{A}} q_A(s), \tag{3.45}$$

$$\boldsymbol{\mu}_s \leftarrow \frac{|A|}{np(s)} \sum_{A \in \mathcal{A}} q_A(s) \langle \mathbf{x} \rangle_A, \tag{3.46}$$

$$\boldsymbol{\Sigma}_s \leftarrow \frac{|A|}{np(s)} \sum_{A \in \mathcal{A}} q_A(s) \langle \mathbf{x}\mathbf{x}^\top \rangle_A - \boldsymbol{\mu}_s \boldsymbol{\mu}_s^\top. \tag{3.47}$$

[5] We ignore the additive constant $-\frac{d}{2} \log(2\pi)$, which translates into a multiplicative constant in (3.41).

Note that the sums over the data points have been replaced by sums over the cells of the partition that is used. Therefore, the number of summations and multiplications that has to be performed for each component scales linearly with the number of cells rather than with the number of data points. Note that, whichever partition we choose, the constrained EM algorithm presented above, which interacts with the data only through the cached statistics of chunks of data, is always a convergent algorithm in that each step increases a lower bound on data log-likelihood. In the limit, if we partition all data points into separate cells, the algorithm coincides exactly with the normal EM algorithm and will converge to a local maximum of the data log-likelihood. It is easy to see that if we refine a specific partition and recompute the responsibilities $q_A(s)$ then $\mathcal{F}$ cannot decrease. Refinement of the partition and recomputing the responsibilities can be viewed as an E-step. Clearly, various trade-offs between the amount of computation and the tightness of the bound can be made, depending on the particular application.

A convenient structure for storing statistics in a way that permits the use of different partitions in the course of the algorithm is a kd-tree (Bentley, 1975; Moore, 1999), see Section 3.3.2, a binary tree that defines a hierarchical decomposition of the data space into cells, each cell corresponding to a node of the tree. As in (Moore, 1999) we store in each node of the kd-tree the sufficient statistics of all data points under this node. Building these cached statistics can be efficiently done bottom-up by noticing that the statistics of a parent node are the sum of the statistics of its children nodes. Building the kd-tree and storing statistics in its nodes has cost $O(N \log N)$.

A particular expansion of the kd-tree corresponds to a specific partition $\mathcal{A}$ of the data space, where each cell $A \in \mathcal{A}$ corresponds to an outer node of the expanded tree. Further expanding the tree means refining the current partition, and in our implementations as heuristic to guide the tree expansion we employ a breadth-first search strategy in which we expand all cells in the current partition. Other strategies, like a best-first one are possible as well. We also need a criterion to stop expanding the tree, and one could use, among others, bounds on the variation of the data posteriors inside a node like in (Moore, 1999), a bound on the size of the partition (number of outer nodes at any step), or sampled approximations of the difference between log-likelihood and $\mathcal{F}$. Another possibility, which we used in our implementation, is to control the tree expansion based on the performance of the algorithm, i.e. whether refining the partition improves the value of $\mathcal{F}$ from the previous partitioning by a significant amount.

3.4.2 Related work

The idea of using a kd-tree for accelerating the EM algorithm in large-scale mixture modelling was first proposed in (Moore, 1999). In that work, in each EM step every node in the kd-tree is assigned responsibility distribution equal to the Bayes posterior of the centroid of the data points stored in the node, i.e., $q_A = p(s|\langle \mathbf{x} \rangle_A)$. In comparison, we use q_A proportional to the average joint log-likelihood, c.f. (3.41). If there is little

variation in the posteriors within a node, e.g. when using fine-grained partitions, the approximation will hardly affect the update in the M-step, and therefore the M-step will probably still increase the data log-likelihood. However this is not guaranteed.

In (Moore, 1999) a different tree expansion is computed in each EM step, as stopping criterion for tree expansion bounds are used on the variation of the posterior probabilities of the data inside a node of the kd-tree. Computing these bounds is a nontrivial operation that involves solving a quadratic programming problem.

The main advantage of our method compared to (Moore, 1999) is that we *provably* increase in each EM step a lower bound of the data log-likelihood by computing the *optimal* responsibility distribution for each node given the current parameters. Moreover, this optimal distribution is independent of the size, shape, and other properties of the node, allowing us to run EM even with very coarse-grained partitions. As mentioned above, and demonstrated in the experiments below, by gradually refining the partitioning while running EM we can get close to the optima of the log-likelihood in relatively few EM steps.

3.4.3 Experimental results

In this section we describe our experiments and present the obtained results. In the experiments we applied our constrained EM algorithm to learn a k-component MoG on the data. First, a kd-tree is constructed and we cache in its nodes the data statistics bottom-up, as explained above. The EM algorithm is started with an initial expansion of the tree to depth two. We keep this partitioning fixed and run the accelerated EM until convergence. Convergence is measured in terms of relative increase in $\mathcal{F}$. We then refine the partitioning by expanding the current cells of the partitioning. Then we run again the constrained EM algorithm until convergence, refine the partitioning breadth-first, etc. We stop the algorithm if the relative change in $\mathcal{F}$ between two successive partitions is smaller than 10^{-4}. In all experiments we used artificially generated data sampled from a randomly initialized k-component MoG in D dimensions with a component separation of c, as defined in (3.33).

Difference with suboptimal shared responsibilities. As indicated by theory, the responsibilities computed from (3.41) maximizing $\mathcal{F}$ for a given $\boldsymbol{\theta}$. For fixed $\boldsymbol{\theta}$, the log-likelihood $\mathcal{L}$ is unaffected by the choice of responsibilities, thus we see from (3.37) that the sum of KL divergences should be smaller when using the optimal shared responsibilities (3.41) than when for example using $p(s|\langle\mathbf{x}\rangle_A)$, as in (Moore, 1999). We investigated the difference in summed KL divergences between the responsibilities and the true posteriors when using (3.41) and using $p(s|\langle\mathbf{x}\rangle_A)$.

We generated 20 data sets of 5000 points from a random $k = 20$ component MoG in $D = 10$ dimensions with a separation of $c = 2$. We then initialized a mixture using

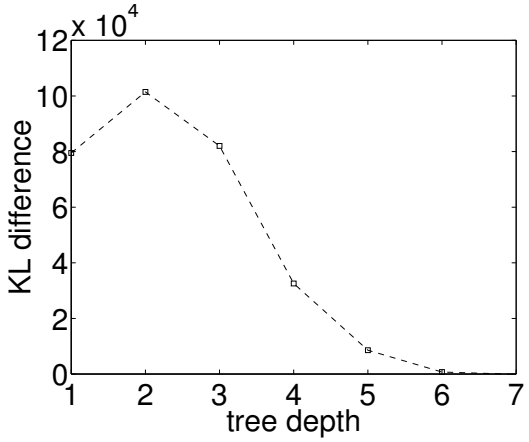


Figure 3.11: Differences in KL divergence as a function of partition size.

k-means and approximated the responsibilities for this mixture. Fig. 3.11 shows the differences in the KL divergence summed over all data points, for different partitioning sizes, averaged over the 20 data sets. Each unit on the horizontal axis corresponds to expanding the tree one level down, and hence doubling the number of nodes used in the partition. The result confirms that our method indeed gives a better approximation, as predicted by theory. The difference is larger for rough partitions, this is as expected since $p(s|\langle x\rangle_A)$ becomes a better approximation of the individual responsibilities as the partition becomes finer.

Gaussian mixture learning: accelerated vs. regular EM. We compared our method to the regular EM algorithm in terms of speed and quality of the resulting mixtures. We looked at how the difference in performance and speed is influenced by the number of data points, dimensions, components, and the amount of separation.

The default data set consisted of 10,000 points drawn from a 10-component 3-separated MoG in 2 dimensions. To compare log-likelihoods we also created a test set of 1000 points from the same mixture. We applied both algorithms to a mixture initialized using k-means. Fig. 3.12 shows the negative log-likelihood of the test set attained by the algorithms and the generating mixture averaged over 20 data sets as well as the speed-up of the accelerated algorithm. Speed was measured using the total number of basic floating point operations (flops) needed for convergence. The graphs show how many times fewer flops were needed by the accelerated algorithm.

The results show that (i) the speed-up is roughly linear in the number of data points (ii) the number of dimensions and components have a negative effect on the speed-up and (iii) the amount of separation has a positive effect. In general the accelerated algorithm requires more iterations to converge but it is still faster than regular EM algorithm since the iterations themselves are executed much faster. The difference in the performance of

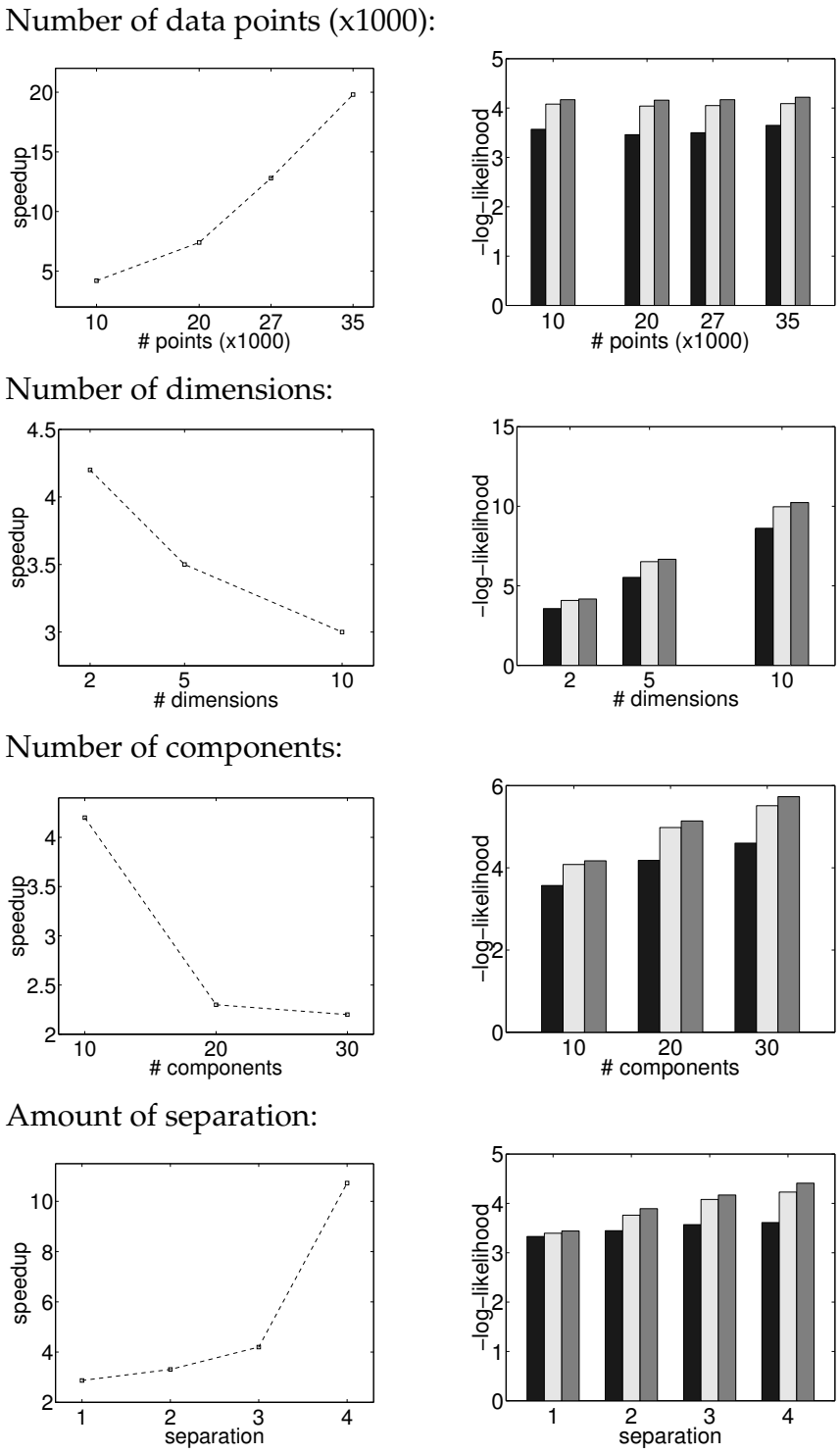


Figure 3.12: Experimental results on Gaussian mixture learning. (Left) Speed-up factor. (Right) Negative log-likelihoods at convergence: generating mixture (black), regular EM (light), accelerated EM (dark).

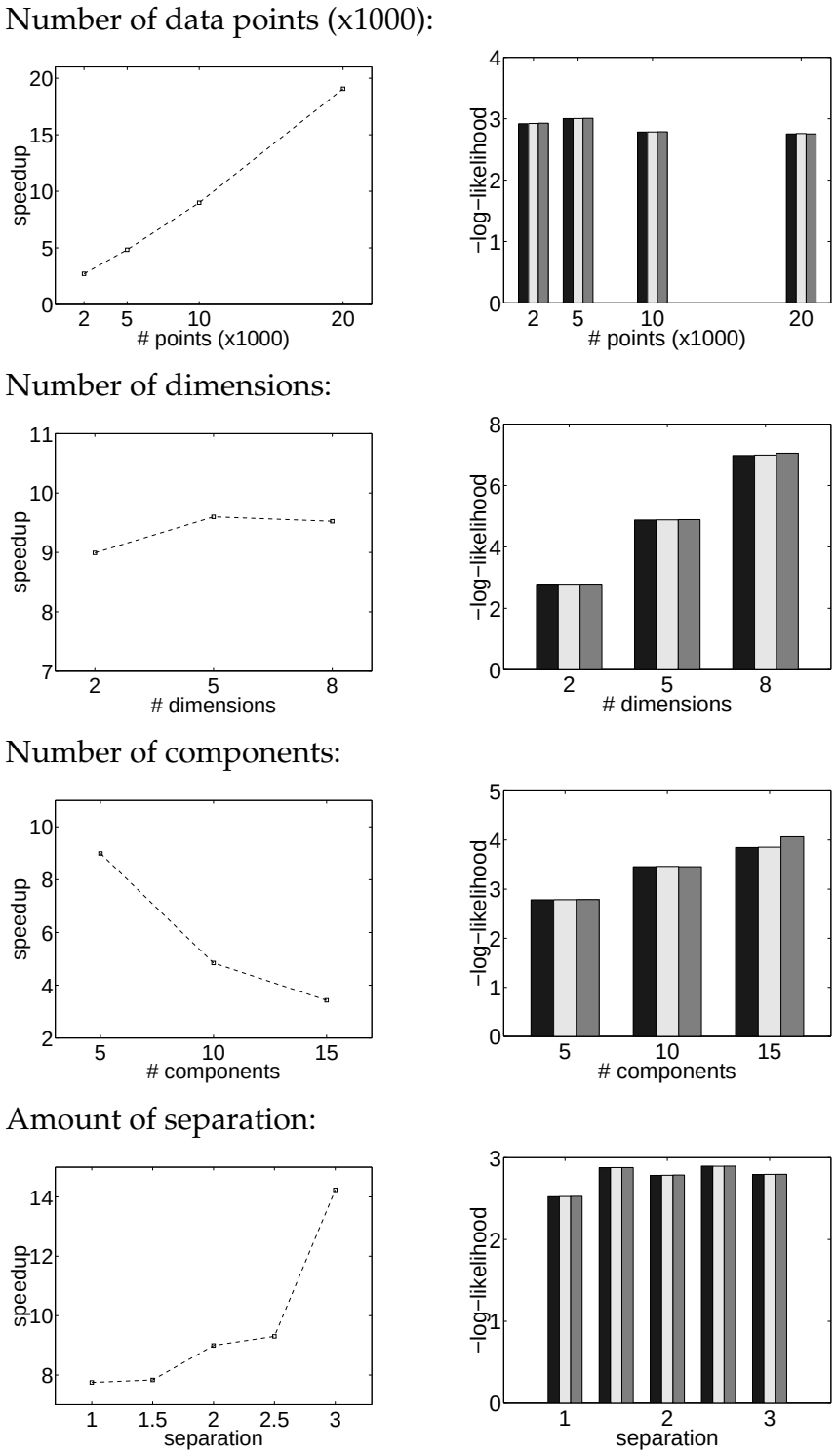


Figure 3.13: Experimental results on greedy Gaussian mixture learning. (Left) Speed-up factor. (Right) Negative log-likelihoods at convergence: generating mixture (black), regular EM (light), accelerated EM (dark).

the two algorithms w.r.t. log-likelihood becomes larger (our method performs worse) as the number of components and data dimensionality increase. However, the difference in log-likelihood between accelerated and regular EM is still small compared to the respective differences with the log-likelihood of the generating mixture, even when using many components in relatively high dimensional spaces.

Gaussian mixture learning: accelerated greedy vs. greedy EM. We also combined the accelerated EM algorithm with the greedy EM algorithm of Section 3.2. Below we present the results of a comparison similar to the comparison presented above, now comparing the standard greedy and accelerated greedy algorithm.

The effects on the speed-up of changing the number of points, dimensions, separation and number of components are similar to those obtained with the non-greedy algorithms. However, the effects when increasing dimension are less pronounced. Furthermore, both greedy algorithms obtain log-likelihoods very close to those of the generating mixture. Thus, using the greedy algorithm improves results for both methods and reduces the difference in log-likelihood between them while obtaining similar speed-ups as compared to the non-greedy algorithms.

3.4.4 Conclusions

We presented an accelerated EM algorithm that can be used to speed-up the EM algorithm to learn MoGs for large data sets. Our contributions over similar existing techniques are two-fold. First, we can show that the algorithm maximizes a lower bound on data log-likelihood and that the bound becomes tighter if we use finer partitions. The algorithm finds mixture configurations near local optima of the log-likelihood surface which are comparable to those found by the regular EM algorithm, and does so considerably faster. Second, the algorithm is convergent and maximizes a lower bound on data log-likelihood for any partitioning of the data. This allows us to use partitions where the true posteriors differ heavily in each part, which are needed when working on very large data sets and only limited computational resources are available. Another option is to start the algorithm with a rough partitioning, for which the EM iterations are performed very fast, and only refine the partitioning when needed because the algorithm converged with the rough partitioning.

Comparing our work with (Moore, 1999), we conclude that with the same computational effort we can use the optimal shared responsibilities instead of the posterior at the node centroid which limits the algorithm to using relatively fine partitions.

The need to store sufficient statistics limits the use of our method, and other methods based on storing sufficient statistics to accelerate MoG learning, to data sets that are of relatively low dimension. This is because the space needed to store the average outer

products grows quadratically with the data dimension. However, if the covariance matrix is constrained to be diagonal then the inner products are sufficient statistics, for which the storage requirements are only linear in the data dimensionality. Finally, note that the proposed technique can directly be applied to other Gaussian mixture models such as the self-organizing mixture models described in the next chapter. An interesting line of further research is to consider whether a similar technique can be developed to speed-up mixture learning for discrete data.

Outlook

In this chapter we have considered Gaussian mixture learning with the EM algorithm. We presented greedy algorithms for MoG learning and k-means clustering that avoid many of the poor local optima found by other methods. The greedy algorithms start with a single component and then iteratively add a component to the mixture and update the mixture parameters with the EM algorithm. The greedy approaches are of particular interest when the number of mixture components (or clusters) has to be determined, since locally optimal mixtures with different numbers of components are generated. We also presented an accelerated EM algorithm that is convergent and guaranteed to increase a lower bound on the data log-likelihood. Our accelerated EM algorithm also supports coarse partitions that cannot be used in existing accelerated EM algorithms.

In the following chapters we will use mixture models for dimension reduction. The dimension reduction techniques developed in those chapters divide the data into several clusters and analyze the relation between the clusters in order to produce a representation of the data in fewer dimensions than the original data. The self-organizing map approach of the next chapter associates with each mixture component a location in a low dimensional latent space. Data points are mapped to the latent space by averaging over the locations of the mixture components according to the posterior $p(s|\mathbf{x})$. In Chapter 5 each mixture component implements a linear map between the data space and the latent space. For each data point, a weighted average of the linear maps of the components is used to map it to the latent space; the weights are given by the posterior on the components.

4

SELF-ORGANIZING MIXTURE MODELS

In this chapter we present a class of models for non-linear data visualization based on probabilistic mixture models. Some of these models exhibit great similarity to Kohonen's self-organizing map, but the fact that ours are based on mixture models allows for several important benefits. In the first section we review Kohonen's self-organizing map and discuss its limitations to motivate our work. Then, in Section 4.2, we present our mixture based approach. Related work on self-organizing maps is discussed in Section 4.3. We provide experimental results obtained when applying our approach to different types of mixture models in Section 4.4, after which we present our conclusions in Section 4.5.[1]

4.1 Self-organizing maps

The self-organizing map, or SOM for short, was introduced by Kohonen in the early 1980s. It extends data clustering in a manner such that it can be used for dimension reduction; each cluster is assigned fixed coordinates in a low dimensional latent space. After estimation of the parameters of a SOM from a given data set, the data can be mapped to the latent space by assigning each data point the latent coordinates associated with the cluster that best matches it. Notably, the mapping from the data space to the latent space implemented by a SOM can be non-linear. A self-organizing map preserves topology in the sense that clusters that are nearby in the latent space will typically contain similar data. We will refer to the clusters also as 'nodes', a term that is common in the SOM literature.

Since their introduction, self-organizing maps have been applied in many engineering problems, see e.g. the list of over 3000 references on applications of SOM in (Kohonen, 2001). One key application area is data visualization. As noted in the introduction, dimension reduction is a valuable tool for visualizing high dimensional data since it is

[1] The material of this chapter is largely drawn from (Verbeek et al., 2003b; Verbeek et al., 2004b).

not possible to use a simple scatter plot for such data. An application of SOMs for for content-based image retrieval can be found in the PICSOM system [2] (Laaksonen et al., 2001). The system provides a search mechanism that allows a user to find images of interest through several interactions with the system. First, the PICSOM system presents some random images from the data-base to the user and then the user identifies relevant images: i.e. images that are most similar to the desired image. Alternatively, the user could directly provide some relevant images to the system. The system uses a SOM to find similar pictures and presents those to the user: it presents images from the node that best matches the relevant images and also from nodes that are in the latent space near the best matching node. The process is then iterated by letting the user add new images (from the presented ones) to the set of relevant images and/or removing others. The system uses several different representations of the images. Among others, color and texture content are used. As the user adds more images to the set of relevant images, it may become apparent that they are all similar in respect to their texture content but not in their color content. The system then identifies that mainly texture content matters and will present new images on the basis of similar texture content.

Kohonen's self-organizing map algorithm. The basic self-organizing map (SOM) algorithm assumes that the data are given as vectors $\mathbf{x}_n \in \mathbb{R}$ $(n = 1, \ldots N)$. With each node s $(s = 1, \ldots, k)$ we associate two vectors: $\mathbf{g}_s$ gives the location in the latent space and $\boldsymbol{\mu}_s \in \mathbb{R}$ is the 'center' in the data space. The latent space is typically two-dimensional in visualization applications of the SOM. The centers $\boldsymbol{\mu}_s$ are estimated in such a way that nearby nodes in the latent space will have nearby centers in the data space. There are two ways to estimate the centers, we can process data items one-by-one (on-line) and all-at-once (batch). The batch algorithm is useful when the data set is fixed. The on-line algorithm is useful when the data comes as a continuous stream. We described the online algorithm in Section 2.2.2.

The neighborhood function is a function that maps two nodes to a real number which encodes spatial relationships of the nodes in the latent space. Typically, the neighborhood function is a positive function which decreases as the distance between two nodes in the latent space increases. An example is the exponent of the negative squared Euclidean distance between the two nodes in the latent space. The neighborhood function evaluated at nodes r and s is written as:

$$h_{rs} = \exp(-\lambda \|\mathbf{g}_r - \mathbf{g}_s\|^2), \tag{4.1}$$

where λ controls the width of the neighborhood function.

Recall that the on-line SOM algorithm processes one data point at a time and for each data point performs two steps. In the assignment step the current data item is assigned to the 'winner': the node r^* with the nearest center (according to some dissimilarity

[2] An on-line demo of the system can be found at `http://www.cis.hut.fi/picsom/`.

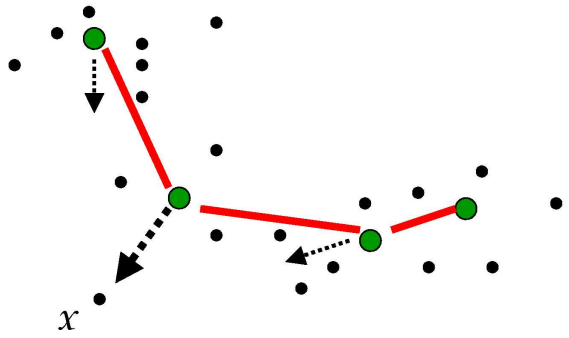


Figure 4.1: The update of the on-line algorithm for a point x. The small discs represent data points and the big discs represent cluster centers. The ordering of the nodes in the latent space is indicated by connecting neighboring nodes with bars. The size of the arrows indicates the force by which the nodes are attracted toward the point x.

measure, often Euclidean distance). In the update step each center $\boldsymbol{\mu}_s$ is moved toward the data vector by an amount proportional to h_{r^*s}: the value of the neighborhood function evaluated at the winner r^* and node s. Thus a center is moved toward the data that is assigned to it, but also toward data that is assigned to nodes nearby in the latent space. Fig. 4.1 illustrates the update for a point indicated with x.

The batch version of the SOM algorithm performs the assignment and update steps for all data points at once. First each data point is assigned to its winning node. Then, in the update step each center $\boldsymbol{\mu}_r$ is placed at the weighted average of all data, where each data vector is weighted by a factor proportional to the neighborhood function evaluated at r and the winner for that data vector. Thus, if we use s_n to indicate the winner for data point $\mathbf{x}_n$, we have:

$$\boldsymbol{\mu}_r \leftarrow \left[\sum_{n=1}^{N} h_{s_n r}\mathbf{x}_n\right] / \left[\sum_{n=1}^{N} h_{s_n r}\right]. \tag{4.2}$$

Limitations of Kohonen's self-organizing map. The Euclidean distance used in the assignment step of the SOM algorithm is not always appropriate, or applicable when the data is not expressed as real-valued vectors. Consider data that consists of vectors with binary entries. The Euclidean distance can still be used, but how will we represent the centers? As binary vectors as well, or should we allow for scalars in the interval $[0, 1]$? In general it is not always clear whether Euclidean distance is appropriate or not. If it is not, then it may not be obvious which dissimilarity measure is more appropriate and whether the centers should be the same type of object as the data vectors (e.g. binary vs. real valued vectors). Hence, an ongoing issue in research on self-organizing maps is to find appropriate dissimilarity measures for different types of data, see e.g. (Kohonen and Somervuo, 2002). Another problematic situation occurs when the data

has several real valued variables but also some discrete variables. Should the dissimilarity be a, possibly weighted, sum of a dissimilarity for the real valued variables and a dissimilarity for the discrete variables?

In some applications values in the data might be missing: a sensor on a robot breaks down, an entry in a questionnaire was left unanswered, etc. In such cases it is common practice when applying Kohonen's SOM algorithm, see e.g. (Kohonen, 2001), to simply ignore the missing values and use a modified dissimilarity measure in order to compute the winning node and to update the centers. However, it is unclear whether and why this is a good idea.

Finally, the Kohonen's SOM algorithm cannot be interpreted as maximizing some objective function and it is not guaranteed to converge (Erwin et al., 1992)

These limitations motivate our mixture model approach to self-organizing maps. In the next section we present our EM algorithm for mixture models that yields topology preserving maps. The algorithm is obtained by using a slightly modified E-step in the standard EM algorithm for mixture models. After we have presented our SOM approach based on mixture models, we compare our approach in Section 4.3 to a number of alternative learning schemes that have been proposed.

4.2 Self-organizing mixture models

In this section we describe how we produce self-organizing maps by using a constrained EM algorithm for parameter estimation of a mixture distribution. First we describe our approach and give an example for modelling data with a Gaussian mixture, then we discuss some algorithmic issues. In Section 4.2.2 we explain how we can deal with missing data values, and in Section 4.2.3 we show how a variation of Kohonen's SOM, the adaptive-subspace SOM, can also be cast in our mixture model approach.

4.2.1 A constrained EM algorithm for self-organizing maps

In the previous chapter we introduced the EM algorithm for mixture models. Recall the EM lower-bound $\mathcal{F}$ on the data log-likelihood $\mathcal{L}$ from the previous chapter:

$$\mathcal{L}(\boldsymbol{\theta}) \geq \mathcal{F}(\{q_n\}, \boldsymbol{\theta},) = \sum_{n=1}^{N} \mathcal{F}_n(q_n, \boldsymbol{\theta}), \tag{4.3}$$

with

$$\mathcal{F}_n(q_n, \boldsymbol{\theta}) = \log p(\mathbf{x}_n; \boldsymbol{\theta}) - \mathcal{D}(q_n \| p(s|\mathbf{x}_n; \theta)) \tag{4.4}$$

$$= \mathrm{E}_{q_n}\left[\log p(\mathbf{x}_n, s; \boldsymbol{\theta})\right] + \mathcal{H}(q_n). \tag{4.5}$$

Where $\mathcal{D}$ and $\mathcal{H}$ denote Kullback-Leibler divergence and entropy respectively. We used $\boldsymbol{\theta}$ to represent the parameters of the mixture model and q_n is a distribution on the mixture components associated with the n-th data point. The standard EM algorithm (Dempster et al., 1977; Neal and Hinton, 1998) can be understood as performing coordinate ascent on the EM lower bound $\mathcal{F}$ on the data log-likelihood function $\mathcal{L}$. In the E-step the bound $\mathcal{F}$ is maximized w.r.t. the distributions q_n for fixed $\boldsymbol{\theta}$, and in the M-step $\mathcal{F}$ is maximized w.r.t. the parameters of the mixture model $\boldsymbol{\theta}$ for fixed q_n. After each E-step the equality $\mathcal{F} = \mathcal{L}$ holds.

In order to obtain self-organizing behavior in the mixture model learning we constrain the distributions q_n to be in a restricted set of distributions $\mathcal{Q}$. The distributions in $\mathcal{Q}$ are similar to the neighborhood function of Kohonen's SOM. Non-negative neighborhood functions h_{rs} can be re-scaled to sum to unity, so that they can be interpreted as a distribution $h_r(s)$ over the components s. The neighborhood function of (4.1) results in:

$$h_r(s) \propto \exp(-\lambda \|\mathbf{g}_r - \mathbf{g}_s\|^2), \qquad \sum_{s=1}^{k} h_r(s) = 1. \tag{4.6}$$

We refer to such neighborhood functions as 'normalized' neighborhood functions. The set $\mathcal{Q}$ contains all the normalized neighborhood functions $h_r(\cdot);\ (r = 1, \ldots, k)$. Using the restricted set of distributions $\mathcal{Q}$, the E-step consists of selecting for each data item n the component r^* such that $\mathcal{F}_n$ from (4.4) is maximized if we use $q_n = h_{r^*}$. As before, we will call r^* the 'winner' for data item n. Since the E-step is constrained, the objective function $\mathcal{F}$ does not have to equal the log-likelihood after the E-step. Instead, the objective function sums data log-likelihood and a penalty term which measures the KL divergence between the (best matching, after the E-step) normalized neighborhood function and the true posteriors $p(s|\mathbf{x})$ for each data point. If the penalty term is low it means that the posteriors do in fact look like the neighborhood function, indicating that components nearby in the latent space represent similar data.

What happens in the M-step if the q_n in the EM algorithm are normalized neighborhood functions? Similar to Kohonen's SOM, the winner for $\mathbf{x}_n$ receives most responsibility and will be forced to model $\mathbf{x}_n$, but also its neighbors in the latent space will be forced, to a lesser degree, to model $\mathbf{x}_n$. So by restricting the q_n in the EM-algorithm to be a normalized neighborhood function centered on one of the nodes, we ensure that the components with large responsibility for a data point are close to each other in the latent space. In this manner we obtain a training algorithm that yields mixture models with self-organizing properties similar to Kohonen's SOM.

The effect can be understood in another way. Consider the second decomposition (4.5) of the terms $\mathcal{F}_n$ in the objective function $\mathcal{F}$ from (4.3). Let us consider also two mixtures, parameterized by $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$, that yield equal data log-likelihood $\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}')$. The objective function $\mathcal{F}$ subtracts from the data log-likelihood a penalty term which consists of KL divergences. Thus if the data log-likelihoods are equal, $\mathcal{F}$ prefers the mixture for

which the KL divergences are the smallest, i.e. the mixture for which the posterior on the mixture components is most similar to the normalized neighborhood function. If the posteriors truly are very similar to the normalized neighborhood function, then this implies that components nearby in the latent space model similar data.

Our EM algorithm for self-organizing mixture models (SOMM) can be summarized as:

SOMM algorithm.
Initialize the parameter vector $\boldsymbol{\theta}$ and then iterate until convergence:

1. **E:** Determine for each $\mathbf{x}_n$ the distribution $q^* \in \mathcal{Q}$ that maximizes $\mathcal{F}_n$, i.e. $q^* = \arg\min_{q \in \mathcal{Q}} D_{KL}(q \| p(s|\mathbf{x}_n; \boldsymbol{\theta}))$, and set $q_n \leftarrow q^*$.
2. **M:** Perform the normal M-step for the mixture, i.e. maximize $\mathcal{F}$ w.r.t. $\boldsymbol{\theta}$ using the q_n computed in the E-step.

In fact, in the M-step we do not need to maximize $\mathcal{F}$ with respect to the parameters $\boldsymbol{\theta}$. As long as we can guarantee that the M-step does not decrease $\mathcal{F}$, we are guaranteed that the iterations of the (modified) EM algorithm will never decrease $\mathcal{F}$.

Finally, in order to project the data to the latent space, we average over the latent coordinates of the mixture components. Using the latent coordinates $\mathbf{g}_s$ ($s = 1, \ldots, k$) of the mixture components, we define the latent coordinates for the n-th data item as:

$$\mathbf{g}_n = \sum_{s=1}^{k} p(s|\mathbf{x}_n) \mathbf{g}_s. \tag{4.7}$$

Example with Gaussian mixture. A particularly simple M-step is obtained when we apply our general algorithm to a mixture of Gaussian densities (MoG) with fixed and equal mixing weights $\pi_s = 1/k$ for all k components and isotropic variance:

$$p(\mathbf{x}|s) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_s, \sigma^2 \mathbf{I}). \tag{4.8}$$

For this simple MoG, the $\mathcal{F}_n$ are given, up to some constants, by:

$$\mathcal{F}_n(q_n, \boldsymbol{\theta}) = -\frac{1}{2} D \log \sigma^2 - \sum_{s=1}^{k} q_{ns} \left[\sigma^{-2} \|\mathbf{x}_n - \boldsymbol{\mu}_s\|^2 / 2 + \log q_{ns} \right]. \tag{4.9}$$

The parameters of the mixture are $\boldsymbol{\theta} = \{\sigma^2, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$. The parameters $\boldsymbol{\theta}$ maximizing $\mathcal{F}$ for given $q_1, \ldots, q_N$ are obtained easily through differentiation of $\mathcal{F}$. In the M-step for this mixture we set:

$$\boldsymbol{\mu}_s \leftarrow \sum_{n=1}^{N} q_{ns} \mathbf{x}_n / \sum_{n=1}^{N} q_{ns} \tag{4.10}$$

$$\sigma^2 \leftarrow \sum_{n=1}^{N} \sum_{s=1}^{k} q_{ns} \|\mathbf{x}_n - \boldsymbol{\mu}_s\|^2 / (ND). \tag{4.11}$$

In comparison to Kohonen's batch SOM algorithm, only the winner definition is different in this case: instead of the the minimizer of $\|\mathbf{x}_n - \boldsymbol{\mu}_r\|^2$ we select the minimizer of $\sum_{s=1}^{k} h_r(s)\|\mathbf{x}_n - \boldsymbol{\mu}_r\|^2$. Hence, the selection of the winners also takes into account the neighborhood function, whereas this is not the case in the standard Kohonen SOM. The update for the means $\boldsymbol{\mu}_s$ given above coincides exactly with that of Kohonen's batch SOM algorithm when using the Euclidean distance.

Shrinking the neighborhood function. Since the objective function might have local optima and the EM algorithm is only guaranteed to give a local optimizer of $\mathcal{F}$, good initialization of the parameters of the mixture model is essential to find good parameters. We can use the parameter $\lambda \in \mathbb{R}$ of the neighborhood function to control the entropy of the neighborhood distributions. In the limit of $\lambda \to 0$, the neighborhood distribution becomes uniform. If all distributions in $\mathcal{Q}$ are uniform then initialization of the parameters $\boldsymbol{\theta}$ is irrelevant since in the E-step all data points are equally assigned to all mixture components. The subsequent M step will give all components very similar parameters.[3] We can start the algorithm with neighborhood distributions with high entropy (small λ) and gradually decrease the entropy until we reach the desired form of the neighborhood function.[4] Other SOM approaches use similar procedures to gradually decrease the entropy of the neighborhood function to avoid poor solutions.

In implementations we started with λ such that the $q \in \mathcal{Q}$ are close to uniform over the components, then we run the EM algorithm until convergence. After convergence we set $\lambda^{new} \leftarrow \eta\lambda^{old}$ with $\eta > 1$, typically η is close to unity, such as 1.1 or 1.01. We initialize the EM procedure for λ^{new}, with $\boldsymbol{\theta}$ found when running EM with λ^{old}.

Reducing computational complexity with sparse winner search. The computational cost of the E-step is $O(Nk^2)$ because for every data point and every component we have to evaluate $\mathcal{F}_n$ which involves a sum over all components. This is a factor k slower than SOM and prohibitive in applications where both N and k are large. However, by restricting the search for a winner in the E-step to a limited number of candidate winners we can obtain an $O(Nk)$ algorithm. A straightforward choice is to use the l components with the largest joint likelihood $p(\mathbf{x}_n, s)$ as candidates, corresponding for our simple example MoG to smallest Euclidean distances to the data point. If none of the candidates yields a higher value of $\mathcal{F}_n(q_n, \boldsymbol{\theta})$, we keep the winner of the previous step. In this way we are guaranteed, in each E-step, not to decrease the objective. We found $l = 1$ to work remarkably well and fast in practice; in this case we only check whether the winner from the previous round should be replaced with the node with highest joint-likelihood $p(s, \mathbf{x}_n)$.

[3] Only when using mixture components for which it is not possible to maximize $\mathcal{F}$ in the M-step w.r.t. $\boldsymbol{\theta}$ initialization is still important.

[4] The entropy of the neighborhood distribution and λ are monotonically related.

4.2.2 Modelling data with missing values

When the given data has missing values the EM algorithm can still be used to train a mixture on the incomplete data (Ghahramani and Jordan, 1994). The procedure is similar to the normal EM algorithm to train a mixture model. Below, we consider the case for i.i.d. data. Let us use $\mathbf{x}_n^o$ to denote the observed part of $\mathbf{x}_n$ and $\mathbf{x}_n^h$ to denote the missing or 'hidden' part. The distributions q_n now range over all the hidden variables of $\mathbf{x}_n$, namely z_n, the generating component, and possibly $\mathbf{x}_n^h$ if there are some unobserved values for $\mathbf{x}_n$. We can again bound the log-likelihood of the observed data:

$$\mathcal{F}(Q, \boldsymbol{\theta}) = \sum_{n=1}^{N} \mathcal{F}_n(q_n, \boldsymbol{\theta}), \tag{4.12}$$

with

$$\begin{aligned} \mathcal{F}_n(q_n, \boldsymbol{\theta}) &= \log p(\mathbf{x}_n^o; \boldsymbol{\theta}) - \mathcal{D}_{KL}(q_n \| p(z_n, \mathbf{x}_n^h | \mathbf{x}_n^o)) && (4.13) \\ &= \mathrm{E}_{q_n} \log p(\mathbf{x}_n^o, \mathbf{x}_n^h, z_n; \boldsymbol{\theta}) + \mathcal{H}(q_n). && (4.14) \end{aligned}$$

In the E-step we set: $q_n \leftarrow p(z_n, \mathbf{x}_n^h | \mathbf{x}_n^o)$. The M-step updates $\boldsymbol{\theta}$ to maximize (or at least increase) the expected joint log-likelihood:

$$\sum_{n=1}^{N} \mathrm{E}_{q_n} \log p(\mathbf{x}_n^o, \mathbf{x}_n^h, z_n; \boldsymbol{\theta}). \tag{4.15}$$

As with the standard EM algorithm for mixture models, after the E-step $\mathcal{F}$ equals the log-likelihood of the observed data, so we see that the EM iterations can never decrease the log-likelihood of the observed data.

To handle missing data in the EM algorithm for self-organizing mixture models we consider the decomposition: $q_n(z_n, \mathbf{x}_n^h) = q_n(z_n) q_n(\mathbf{x}_n^h | z_n)$. We can write (4.14) as:

$$\mathcal{F}_n(q_n, \boldsymbol{\theta}) = \mathcal{H}(q_n(z_n)) + \mathrm{E}_{q_n(z_n)} \left[\mathcal{H}(q_n(\mathbf{x}_n^h | z_n)) + \mathrm{E}_{q_n(\mathbf{x}_n^h | z_n)} \log p(z_n, \mathbf{x}_n^h, \mathbf{x}_n^o) \right]. \tag{4.16}$$

Just as before, we can now constrain the $q_n(z_n)$ to be a normalized neighborhood function. In order to maximize $\mathcal{F}$ w.r.t. Q in the E-step, we set $q_n(\mathbf{x}_n^h | z_n) \leftarrow p(\mathbf{x}_n^h | \mathbf{x}_n^o, z_n)$. The optimization of $\mathcal{F}$ w.r.t. $q_n(z_n)$ is achieved by choosing $q_n(z_n)$ to be the neighborhood function that maximizes (4.16). In the M-step we then have to maximize (or at least increase) the expected joint (complete) data log-likelihood

$$\sum_{n=1}^{N} \sum_{z_n=1}^{k} q_n(z_n) \left[\log \pi_{z_n} + \mathrm{E}_{q_n(\mathbf{x}_n^h | z_n)} \log p(\mathbf{x}_n^o, \mathbf{x}_n^h | z_n) \right] \tag{4.17}$$

w.r.t. the parameters $\boldsymbol{\theta}$. The class of mixture components that is employed determines how easy or difficult it is to derive and perform the M-step, and whether we can maximize or just increase $\mathcal{F}$ w.r.t. $\boldsymbol{\theta}$ in the M step.

Things become particularly simple when the mixture components distribute independently over variables which have missing values and the other variables. In this case, instead of (4.12)-(4.14), we can use the normal EM bound on the data log-likelihood:

$$\mathcal{F}(Q, \boldsymbol{\theta}) = \sum_n \mathcal{F}_n(q_n, \boldsymbol{\theta}) \tag{4.18}$$

with

$$\begin{aligned} \mathcal{F}_n(q_n, \boldsymbol{\theta}) &= \log p(\mathbf{x}_n^o; \boldsymbol{\theta}) - D_{KL}(q_n \| p(z_n|\mathbf{x}_n^o)) && (4.19) \\ &= \mathrm{E}_{q_n} \log p(\mathbf{x}_n^o, z_n; \boldsymbol{\theta}) + \mathcal{H}(q_n). && (4.20) \end{aligned}$$

This is possible since we now have (by assumption) standard expressions for the distributions $p(\mathbf{x}_n^o|z_n)$. The E-step in this case sets $q_n(z_n) \leftarrow p(z_n|\mathbf{x}_n^o) \propto p(z_n)p(\mathbf{x}_n^o|z_n)$, i.e. the E-step simply 'ignores' the missing values. Ignoring missing data in the assignment step of SOM algorithms is common, here we see that within our framework it is only justified to do so if our model distributes over the observed variables independently of the variables with missing values.

4.2.3 The adaptive-subspace self-organizing map.

The adaptive-subspace self-organizing map (ASSOM) (Kohonen et al., 1997) learns invariant features of data patterns. The ASSOM does not process single data points (patterns) but small sets A_i $(i = 1, \ldots, N)$, called 'episodes', of slightly transformed patterns. The data points in an episode are forced to be assigned the same winner, thereby forcing the nodes of the map to fit patterns *and* their transformations. The nodes of the basic ASSOM are parameterized by a set of vectors that together span a linear subspace of the complete signal space. The learning algorithm updates the subspace of each node in such a manner that the patterns, and their transformations, assigned to the node have minimum average squared Euclidean distance to the subspace. In this manner the distance of a pattern to a subspace will tend to be more or less invariant for the transformations included in the episodes. The ASSOM thus ensures that the variation of patterns between the different nodes does not contain the transformations that were used to generate the episodes.

The ASSOM can be readily cast in the probabilistic mixture framework presented here. To do this we force all patterns $\mathbf{x}$ in an episode A_i of transformed patterns to share the same responsibilities $q_{A_i}(s)$ over the components s of the mixture. We can then rewrite

the objective function $\mathcal{F}$ defined in (4.3)-(4.5) as:

$$\mathcal{F} = \sum_{i=1}^{N} \mathcal{F}_{A_i}(\boldsymbol{\theta}, q_{A_i}), \tag{4.21}$$

$$\mathcal{F}_{A_i} = \sum_{\mathbf{x} \in A_i} \sum_{s=1}^{k} q_{A_i}(s) \left[\log p(\mathbf{x}, s) - \log q_{A_i}(s)\right] \tag{4.22}$$

$$= |A_i| \sum_{s=1}^{k} q_{A_i}(s) \left[\log p(s) + \langle \log p(\mathbf{x}|s) \rangle_{A_i} - \log q_{A_i}(s)\right], \tag{4.23}$$

where $\langle \log p(\mathbf{x}, s) \rangle_{A_i} = \frac{1}{|A_i|} \sum_{\mathbf{x} \in A_i} \log p(\mathbf{x}, s)$ and $|A_i|$ denotes the number of patterns in the episode. In the E-step we then set q_{A_i} to maximize $\mathcal{F}(\boldsymbol{\theta}, q_{A_i})$. Compared to computations needed in the basic E-step presented in Section 4.2, we simply replace the complete log-likelihood $\log p(\mathbf{x}, s)$ with the average log-likelihood over the episode: $\langle \log p(\mathbf{x}, s) \rangle_{A_i}$.

Since the ASSOM nodes span subspaces of the original signal space, it is reasonable to use Gaussian components with constrained covariance matrices of the PCA type, see Section 3.1.4. The covariance matrix can then be written as: $\Sigma = \sigma^2 \mathbf{I} + \Lambda \Lambda^\top$. The d columns of the matrix $\boldsymbol{\Lambda}$ span the linear subspace with large variance and the $\sigma^2 \mathbf{I}$ term adds some small variance in all variables that makes the covariance matrix non-singular. Note that here we essentially use the same technique as in Section 3.4 on accelerated Gaussian mixture learning, but for a different purpose.

4.3 Comparison with related work

Several other modifications of the original SOM algorithm have been proposed to overcome its limitations. These modifications can be divided into two categories. First, different dissimilarity measures, used to determine the winning node, have been proposed for different types of data. Second, alternative learning algorithms for SOMs have been proposed which can be shown to converge to a local maximum of some objective function. We discuss some of these methods below and compare them to our approach.

Our approach offers contributions in both these categories. First, since our approach is applicable to any mixture model for which we have a standard EM algorithm, it can be applied to a wide range of data types. The class of mixture components that is used implicitly provides the dissimilarity measure. In our approach, the mixture component densities can be designed based on assumptions or prior knowledge on how the data is distributed. The actual algorithm is then derived as an instantiation of the general algorithm. Hence, we separate the design of the model from the learning algorithm. For example, we could use a mixture of Bernoulli distributions to model binary variables. Since we merely replace the E-step, we can very easily make a self-organizing map

version of any mixture model for which we have a normal EM algorithm. Second, our learning algorithm is guaranteed to converge to a local maximum of the objective function $\mathcal{F}$. The proposed algorithm also offers the other merits of probabilistic models, like the ability to handle missing data values and the ability to learn mixtures (of self-organizing maps).

Kohonen's self-organizing map. Let us first consider the Kohonen's original SOM (KSOM) algorithm. We saw that our algorithm is very close to KSOM when applied on the simple MoG used in the example in Section 4.2: the update of the means of the Gaussians is the same as the update for the centers in KSOM with Euclidean distance as dissimilarity. For this simple model the conditional log-likelihood $p(\mathbf{x}|s)$ is given by the negative squared Euclidean distance $\|\mathbf{x} - \boldsymbol{\mu}_s\|^2$ (up to some additive and multiplicative constants). If we use a sparse search for the winner (E-step) with only $l = 1$ candidate, the difference with Kohonen's winner selection is that we only accept the closest node, in terms of Euclidean distance, as a winner when it increases the objective $\mathcal{F}$ and keep the previous winner otherwise.

In contrast to KSOM, we presented a SOM algorithm applicable to any type of mixture model. We can use prior knowledge to select specific mixture components for given data and then train it with our SOMM algorithm. In particular, the probabilistic framework allows us to model dependencies between the different variables and helps us design SOM algorithms for data types which are difficult to handle in the original SOM framework, e.g. data that consists of variable-size trees or sequences, data that combines numerical with categorical features, etc. Finally, our SOMM algorithm is guaranteed to converge and it finds a (local) maximum of an easy to interpret objective function that sums data log-likelihood and a penalty term. For the KSOM algorithm no such objective function exists (Erwin et al., 1992) and it is not guaranteed to converge.

Soft topographic vector quantization. The soft topographic vector quantization algorithm (STVQ) (Graepel et al., 1998; Heskes, 2001) our SOMM algorithm presented above. Given some dissimilarity measure $d(\mathbf{x}_n, \boldsymbol{\theta}_s)$ between data items and nodes, the following objective function is maximized w.r.t. the q_{ns} and $\boldsymbol{\theta}_s$ by STVQ:

$$\mathcal{F}_{STVQ} = -\sum_{n=1}^{N}\sum_{s=1}^{k} q_{ns} \log q_{ns} - \beta \sum_{n=1}^{N}\sum_{s=1}^{k} q_{ns} \sum_{r} h_{rs} d(\mathbf{x}_n, \boldsymbol{\theta}_s), \tag{4.24}$$

with all $q_{ns} \geq 0$ and $\sum_s q_{ns} = 1$. The error function sums an entropy term and an error term. The error term $d(\mathbf{x}_n, \boldsymbol{\theta}_s)$ can be based on squared Euclidean distance to a reference vector $\boldsymbol{\mu}_s$ for each component s, but other errors d based on the exponential family can be plugged in.[5] Instead of selecting a single 'winner' as in our work, an unconstrained

[5] The exponential family contains all the distributions for which the log-likelihood $\ln p(x|\theta)$ can be written as: $\ln p(x|\theta) = a_0(x) + b_0(\theta) + \sum_{i=1}^{m} a_i(x) b_i(\theta)$, for finite m. It includes many well known distributions

distribution q_{ns} over the nodes is used. Since a complete distribution over the nodes is used rather than selecting a single winner, one cannot apply the 'sparse' winner search speed-up. However, speed-ups can be achieved when for each n not all q_{ns} are optimized, but rather the q_{ns} for only a few components s are optimized (different ones for each data item), while keeping the values q_{ns} for other components fixed.

The parameter β is used for annealing in STVQ. For small β the entropy term, with only one global maximum, becomes dominant, whereas for large β the error term, potentially with many local optima, becomes dominant. A deterministic annealing scheme (Rose, 1998) is followed: by gradually increasing β more structure is added to the objective function until some desired value of β is reached. In comparison, we use the neighborhood function for annealing: we start with a broad neighborhood function and shrink it gradually to the desired shape in order to be less sensitive to the initialization of the parameters. Both approaches have a similar effect.

After optimizing over the q_{ns}, the error function $\mathcal{F}_{STVQ}$ may be interpreted as the sum of log-likelihood of the data under a mixture model p plus a penalty term independent of the data (Heskes, 2001). Using squared Euclidean distance as a dissimilarity measure, the components and mixing weights of the mixture model p are given by :

$$p(\mathbf{x}) = \sum_{s=1}^{k} \pi_s \mathcal{N}(\mathbf{x}; \mathbf{m}_s, \beta^{-1}\mathbf{I}), \tag{4.25}$$

where the parameters of this mixture are given by:

$$\pi_s = \frac{\exp(-\beta v_s)}{\sum_{s'=1}^{k} \exp(-\beta v_{s'})}, \qquad \mathbf{m}_s = \sum_{r=1}^{k} h_{sr}\boldsymbol{\mu}_r, \qquad v_s = \sum_{r=1}^{k} h_{sr}\|\boldsymbol{\mu}_s - \boldsymbol{\mu}_r\|^2. \tag{4.26}$$

The validity of this interpretation is of course dependent on the particular dissimilarity measure that is used. The dissimilarity measure should be derived from distributions in the exponential family to validate the mixture model interpretation.

In comparison, the relation of our objective function $\mathcal{F}$ to the log-likelihood of the data under a mixture model (log-likelihood plus a penalty between the true posterior and the neighborhood function) also holds for models outside the exponential family. Note that in STVQ the annealing parameter β, which is set by hand or through an annealing schedule, controls parameters (the mixing weights and the variance of the mixture components) in the corresponding mixture model. In our approach the neighborhood function is used for annealing and it appears in the objective function only in the penalty term and does not control the parameters of the mixture model. Thus our objective function $\mathcal{F}$ is related to data log-likelihood under a mixture model in a simpler way than $\mathcal{F}_{STVQ}$. Also, in our approach we can optimize over all the parameters of the mixture model, while in STVQ the final value of the variance is set by hand.

such as the Gaussian, inverse Gaussian, gamma, Poisson, and binomial distribution.

Generative Topographic Mapping. Generative Topographic Mapping (GTM) (Bishop et al., 1998b) achieves topographic organization of the cluster centers in a quite different manner than the self-organizing map approaches discussed so far, see also Section 2.2.2 where we already reviewed the GTM. GTM fits a constrained probabilistic mixture model to given data. The parameters $\boldsymbol{\theta}_s$ $(s = 1, \ldots, k)$ of the k mixture components are parameterized as a linear combination of a fixed set of m smooth nonlinear basis functions ϕ_i of the fixed coordinates $\mathbf{g}_s$ of the components in the latent space. For a Gaussian mixture model the means $\boldsymbol{\mu}_s$ are then of the form:

$$\boldsymbol{\mu}_s = \sum_{i=1}^{m} \alpha_{is} \phi_i(\mathbf{g}_s) \tag{4.27}$$

Due to the smooth mapping from latent coordinates of the mixture components to their parameters, nearby components in the latent space will have similar parameters. The parameters α_{is} of the linear map are fitted by a standard EM algorithm to maximize the data log-likelihood. Although originally presented for Gaussian mixtures, the GTM can be extended to mixtures of other members of the exponential family (Girolami, 2001; Kaban and Girolami, 2001). To our knowledge it is unknown how to extend the GTM to mixtures of component densities outside the exponential family.

The main benefit of GTM over SOMM is that the parameter fitting can be done directly on the basis of maximum likelihood. For SOMM we need to specify the number of mixture components and the neighborhood function. For GTM we need to specify: the number of mixture components, the number of basis functions and their smoothness and shape. The GTM parameters can be dealt with in a Bayesian manner by specifying appropriate prior distributions over the parameters. A a-posteriori distribution over parameter settings given the data can then be found using (approximate) inference techniques when enough computational resources are available (Bishop et al., 1998a). For the SOMM approach it is not clear whether such techniques can be used to set the parameters of the algorithm. Since the SOMM has fewer parameters to be set, it may be preferred in situations where no prior distributions on the parameters are available or when limited computational resources prevent the use of approximate inference techniques to estimate the parameters of the GTM.

Other mixture based SOM algorithms. Another approach to learn mixture models that implements topology preserving maps, similar to GTM, is presented (Utsugi, 1998). A special smoothness prior on the parameters is used that favors nearby components in the latent space to have similar parameter. The type of prior that is appropriate depends on the type of mixture components that is used. Thus, a new class of mixture components potentially also requires a new prior distribution. This makes it less straight-forward to apply this approach to new types of mixture components than our approach for which we only need to derive the new M-step. For this method it is not

clear whether it generalizes directly to mixture models outside the exponential family and whether speed-ups can be applied to avoid $O(nk^2)$ run-time.

A different way to cast the SOM in a probabilistic framework is given by (Anouar et al., 1998). The log-likelihood function that is maximized is:

$$\mathcal{L}' = \sum_{n=1}^{N} \log \sum_{s=1}^{k} p_{ns} p(\mathbf{x}|\boldsymbol{\theta}_s). \tag{4.28}$$

Here, each data point has its likelihood evaluated under its own mixture of the k component densities $p(\cdot|\boldsymbol{\theta}_s)$ $(s = 1, \ldots, k)$ with mixing weights p_{ns}. The algorithm iterates two steps. In the first step, we set:

$$p_{ns} \leftarrow h_{r_n s} \quad \text{with: } r_n = \arg\max_r \sum_{s=1}^{k} h_{rs} p(\mathbf{x}_n|s). \tag{4.29}$$

In the second step a new parameter vector is computed for given p_{ns}. However, the new parameter vector is in general not guaranteed to increase $\mathcal{L}'$ for the fixed p_{ns}. Just as our approach, this model in principle allows one to use any type of mixture components. Contrary to our approach, this algorithm does not optimize data log-likelihood under a *single* mixture, since the mixing weights vary for each data item and change throughout the iterations of the algorithm. Moreover, the second learning step is not guaranteed to improve the objective function. The algorithm has run-time $O(Nk^2)$, but can benefit from the same speed-up as our approach.

Another probabilistic SOM-like model, based on isotropic Gaussian components, is given in (Kostiainen and Lampinen, 2002). The maximum-likelihood estimation procedure of the means of the Gaussian densities coincides with the estimation of the centers in the batch version of Kohonen's SOM algorithm. There are, however, some difficulties with this approach. First, the final density estimate is not smooth throughout the data space. Second, the estimation of the variance of the Gaussian densities cannot be performed in closed form but requires numerical optimization techniques. Third, to evaluate the likelihood a normalizing constant has to be computed which is defined through integrals that cannot be analytically evaluated. Therefore sampling techniques are required to approximate it. Furthermore, the density model appears to be restricted to Gaussian mixtures with the corresponding squared Euclidean distance as dissimilarity measure.

4.4 Experimental results

In this section we present experimental results obtained with the SOMM algorithm. We start with an experiment using Gaussian component densities. The second example

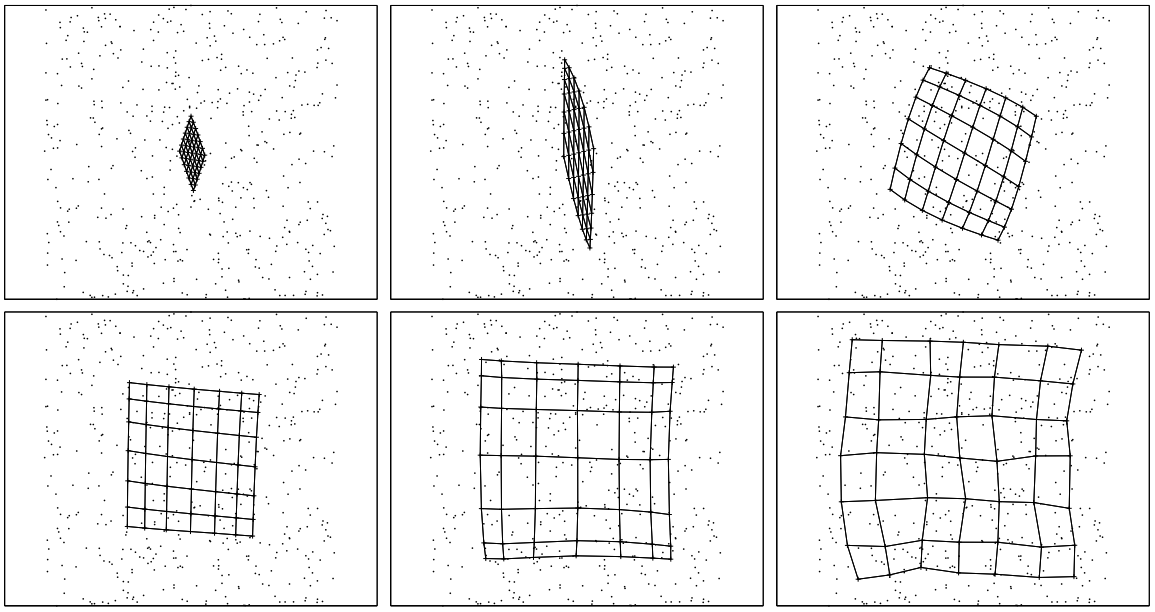


Figure 4.2: Configuration of the means $\boldsymbol{\mu}_s$ during training after $10, 20, \ldots, 60$ EM iterations.

uses Bernoulli distributions for binary data and in the third example we model data with both discrete and continuous features.

The simple Gaussian mixture. We generated a data set by drawing 500 points at random from a uniform distribution over the unit square and trained two different self-organizing mixture models. The first model consisted of 49 Gaussian densities placed on a 7 by 7 rectangular grid in the latent space and we used a Gaussian neighborhood function as in Section 4.2. In Fig. 4.2 we show the means $\boldsymbol{\mu}_s$ of the Gaussian components, connected according to the grid in the latent space. Configurations after each 10 EM iterations are shown for the first 60 iterations. It can be seen how the SOMM spreads out over the data as the neighborhood function is shrunk until it has about 80% of its mass at the winner at iteration 60. Fig. 4.3 shows the same experiment for the second model, which has 50 Gaussian components placed on a regular grid in a one dimensional latent space. Here we showed the configuration after every 20 iterations. Again, we can observe the SOM spreading out over the data as the algorithm progresses. In this case the SOM has to fold itself in order to approach the uniform distribution of the data.

To quantitatively determine the similarity with Kohonen's SOM (KSOM) we compared the results when using our SOMM and KSOM for the unit square data. For both methods we evaluated the objective function $\mathcal{F}$ of SOMM, as defined in (4.3)-(4.5). To evaluate the objective function for KSOM, we used the means as obtained with KSOM and the variances obtained with SOMM to specify a mixture model. Both methods used the same neighborhood shrinking scheme: λ is multiplied by 1.1 after convergence with the

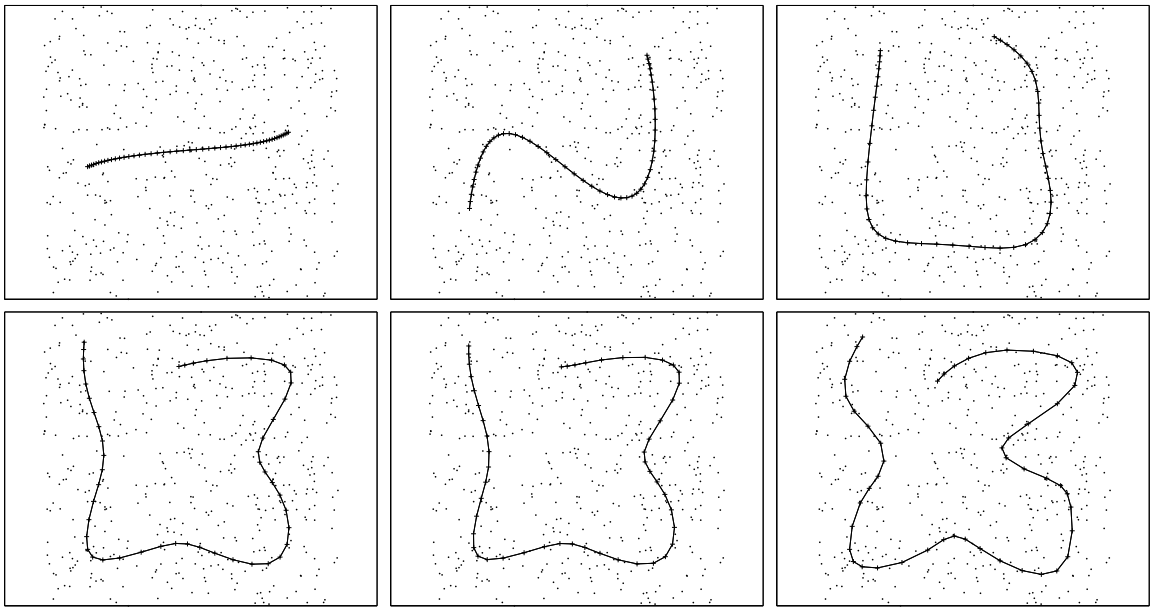


Figure 4.3: Configuration of the means $\boldsymbol{\mu}_s$ during training after $20, 40, \ldots, 120$ EM iterations.

current neighborhood function. For SOM the neighborhood function was also changed if the algorithm did not converge after 1000 iterations. In Fig. 4.4 we summarize the results for different numbers of mixture components k and dimensions of the latent space d. We used a standard rectangular grid for the components in the latent space and a Gaussian neighborhood function. The reported results on $\mathcal{F}_{SOMM}$ and $\mathcal{F}_{KSOM}$ are averages over 50 experiments; the standard deviations are given as well.

In general, SOMM and KSOM give similar values of the objective function. The difference in the average objective function between the methods is in general smaller than the standard deviations in the results. Recall that the objective function $\mathcal{F}$ is the sum of the data log-likelihood and a penalty term $\mathcal{D}$ which sums the KL divergences between the neighborhood function and the true posteriors on the components. If we consider the penalty terms $\mathcal{D}_{SOMM}$ and $\mathcal{D}_{KSOM}$, obtained respectively with SOMM and KSOM, we again see that the differences are small and close to the standard deviation in the penalty terms observed for each method. We conclude that for the MoG considered here there is no significant difference between the performance of SOMM and KSOM.

A mixture of Bernoulli distributions. In this example we apply the SOMM algorithm to a mixture with non-Gaussian components. We model word occurrence data in documents obtained from the 20 newsgroup data set.[6] The data set contains occurrence of 100 words in 16242 newsgroup postings. The data is represented as a 100×16424 matrix with entries that are either zero or one. Each row represents a word and each

[6] This data set is available at: `http://www.ai.mit.edu/~jrennie/20Newsgroups/`.

k	d	$\mathcal{F}_{SOMM}$	$\mathcal{F}_{KSOM}$	$\mathcal{D}_{SSOM}$	$\mathcal{D}_{KSOM}$
10	1	725.0 ± 16.0	728.1 ± 13.1	99.5 ± 5.9	108.2 ± 7.6
25	1	751.1 ± 14.1	758.2 ± 12.8	120.2 ± 5.5	124.1 ± 6.3
100	1	840.0 ± 15.0	848.7 ± 12.7	119.6 ± 5.7	124.6 ± 4.4
9	2	730.0 ± 9.0	731.6 ± 9.0	115.9 ± 2.6	115.0 ± 2.8
25	2	753.2 ± 8.6	754.3 ± 8.4	133.9 ± 3.1	135.4 ± 3.8
100	2	797.7 ± 9.7	801.9 ± 8.4	149.7 ± 4.3	153.9 ± 4.1

Figure 4.4: Comparison of KSOM and SOMM in terms of the SOMM objective $\mathcal{F}$ and penalty $\mathcal{D}$.

column represents a document. A `1` indicates that the specific word occurs in a specific document. Thus, each word is a data point, and its feature vector indicates in which documents this word occurs.

We use mixture components that assume independence between all the bits in the feature vector, and each bit is Bernoulli distributed. We use N to indicate the number of words and D to denote the number of documents and $x_n^d \in \{0, 1\}$ to denote the value of the d-th bit of the feature vector (presence in document d) of the n-th word $\mathbf{x}_n$. The probability of the d-th feature being '1' according to the s-th mixture component is denoted p_{sd}. Thus, the likelihood of $\mathbf{x}$ under component density s is given by:

$$p(\mathbf{x}_n|s) = \prod_{d=1}^{D} p_{sd}^{x_n^d} (1 - p_{sd})^{(1-x_n^d)}. \tag{4.30}$$

The mixing weights of all $k = 25$ components were taken equal. To perform the E-step in our SOMM algorithm we compute the conditional log-likelihoods $\log p(\mathbf{x}_n|s)$ as:

$$\log p(\mathbf{x}_n|s) = \sum_{d=1}^{D} \left[(1 - x_n^d) \log(1 - p_{sd}) + x_n^d \log p_{sd} \right], \tag{4.31}$$

and proceed as usual. In the M-step we have to maximize the expected joint log-likelihood w.r.t. the parameters p_{sd}. This yields:

$$p_{sd} \leftarrow \frac{\sum_{n=1}^{N} q_{ns} x_n^d}{\sum_{n=1}^{N} q_{ns}}, \tag{4.32}$$

i.e. we set p_{sd} to the relative frequency of occurrence of the words in document d where all the words are weighted by the responsibilities q_{ns}. Note that the update of the p_{sd} is similar to that for the means $\boldsymbol{\mu}_s$ of the simple MoG given in (4.10). This suggests that we might as well treat the data as if it were continuous valued and use Gaussian mixture components. In the E-step, however, the conditional log-likelihoods are quite different: they are given for the simple MoG, up to some constant, as squared Euclidean distances,

while here by (4.32). Thus the Bernoulli distributions yield a different 'dissimilarity' between data points and mixture components (or nodes).

In Fig. 4.5 we plotted the words at the corresponding $\mathbf{g}_n$; the bottom plot zooms in on the area indicated by the box in the top plot. The visualization of the data allows us to identify different regions of the map that contain words that have similar occurrence patterns in the documents. Due to the huge dimensionality of this data, the posteriors are rather peaked, i.e. the posteriors have very low entropy resulting in $\mathbf{g}_n$, as computed from (4.7), that are almost equal to one of the $\mathbf{g}_s$. In a visualization this results in placing the words almost exactly at the locations of the mixture components in the latent space. If we print the words on their location in the latent space, they would be printed on top of each other. Therefore, for visualization we did not use the actual posterior but distributions $p'(s|\mathbf{x}_n) \propto p(s|\mathbf{x}_n)^\alpha$ for $0 < \alpha < 1$ such that the entropies of the $p'(s|\mathbf{x}_n)$ were close to two bits.[7]

Modelling credit card application data. In this section we consider a self-organizing mixture model for data set describing 690 credit card applications, available from the UCI machine learning repository (Blake and Merz, 1998). The data has 6 numerical features and 9 nominal attributes.[8] The meaning of the attributes is not provided with the database for privacy reasons, except for the attribute that indicates whether the credit card application was approved or not. Therefore, interpretation of modelling results in terms of customer profiles is difficult. However, below we summarize and compare results we obtained when applying different SOM algorithms on this data. For each model we used $k = 25$ components arranged in a two dimensional square grid and a Gaussian neighborhood function.

A simple way to extend the standard Kohonen SOM (KSOM) to data that has nominal attributes is to use the one-of-n encoding: each nominal variable is represented with a binary vector, one bit for each possible value, with a `1` in the location of the particular value a data item has for the nominal variable. Thus a variable that can take as values 'red', 'green' or 'blue' is encoded with a bit vector of length three, where the values are encoded with the vectors `001, 010` and `100`. The final data vector then becomes the concatenation of the continuous data attributes and the binary vectors of the nominal attributes. The KSOM model can be trained by either proceeding as usual on the one-of-n ($1/n$) encoding, or by constraining the means of the nodes to be in the $1/n$ encoding as well (Negri and Belanche, 2001). The latter corresponds to a normal batch KSOM update step, and then mapping the updated mean vector to the $1/n$ encoding with a `1` in the position of the maximum value of the mean vector. In the results below the

[7] The justification for using the normalized exponentiated posteriors is again that if we have a distribution and want the distribution that is closest in the sense of KL divergence to the original under a constraint on the entropy, then we should use the normalized exponentiated distribution.

[8] For 37 records the values of one or more attributes are missing, in the example below we removed these cases from the data set for simplicity.

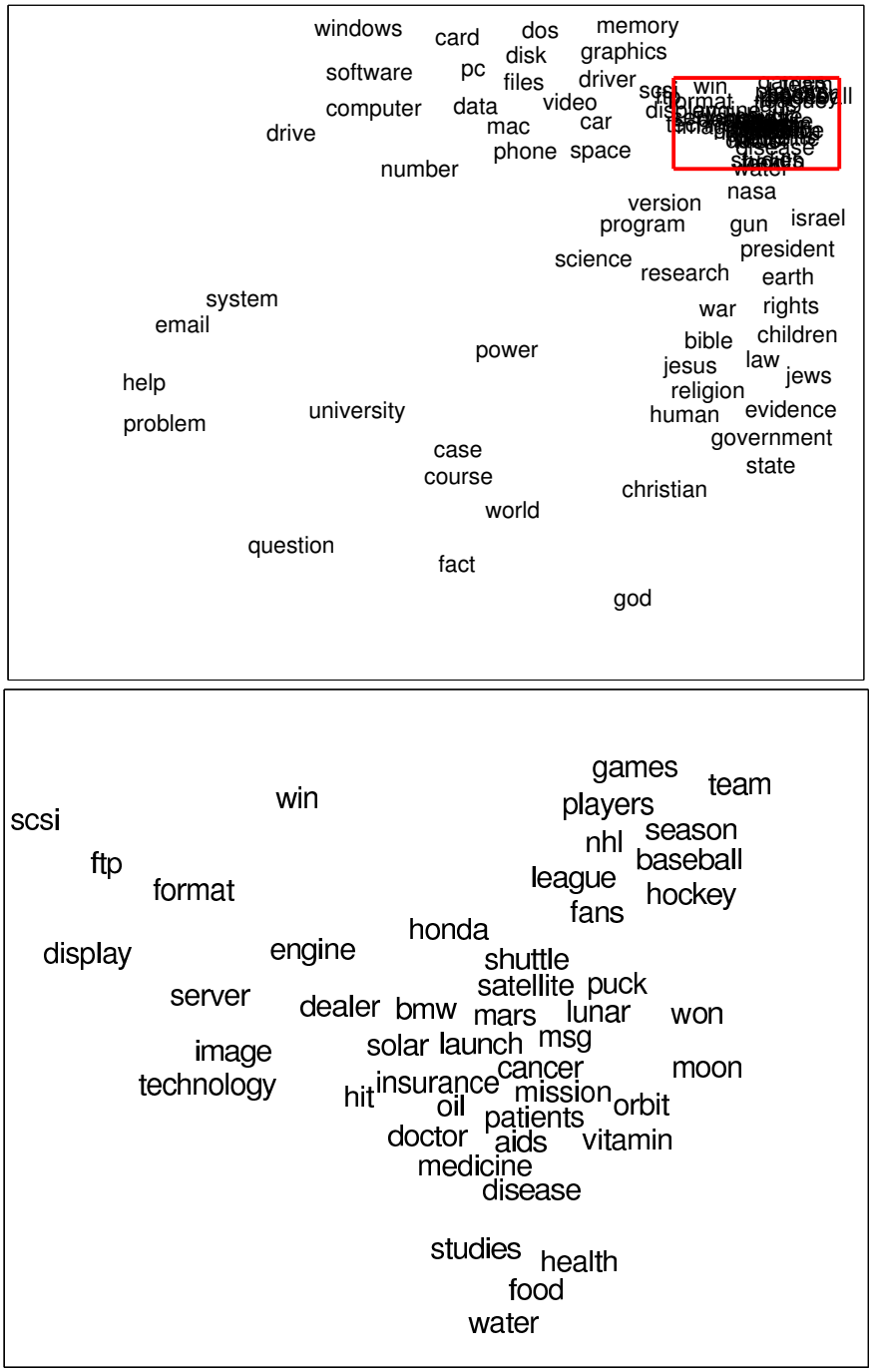


Figure 4.5: Self-organizing Bernoulli models, $k = 25$. The bottom plot displays the area in the box in the top plot. Note that the ambiguous word 'win' (referring to the operating system 'windows' and to 'winning' a game) is located between the computer terms (top left area) and the sports terms (top right area).

normal KSOM is indicated by KSOM and the version with means constrained to the $1/n$ encoding is indicated with KSOM-$1/n$.

In this experiment we considered variants of SOMM with different combinations of densities for the continuous variables and distributions for the discrete variables. For the continuous variables we used multivariate Gaussian densities with three types of covariance matrices : isotropic (i), diagonal (d) and general (g). We used two different distributions on the nominal variables. The first distribution modelled all nine nominal variables independently (i). The second distribution modelled some pairs of nominal variables jointly (p). The different combinations are indicated by SOMM-xy, where x denotes the covariance matrix structure (i, d, or g) and y denotes the nominal variable model (i or p).

The four pairs of nominal variables that were modelled jointly were selected by computing (based on the complete data set) the mutual information[9] between pairs of nominal variables, and we selected the pairs with maximum mutual information in a greedy manner. For two of the four pairs the mutual information was in the order of the entropy of one of the variables in the pair, indicating strong correlations between the variables. The continuous variables had considerably different ranges of values (differing by several orders of magnitude). Therefore, we scaled all continuous variables to have unit variance and zero mean before learning the different models. Note that for diagonal and general covariance matrices this scaling is superfluous, since the model can take the different variances into account.

In order to compare the KSOM models with the SOMM models, we used the objective function $\mathcal{F}$ of SOMM. When we train a KSOM model this yields a final assignment of data items to nodes after the last training step. To obtain a mixture model for which we can evaluate $\mathcal{F}$, we used the neighborhood function of the winning node for a data point as its responsibility over mixture components. We then performed a single M-step of SOMM with isotropic covariance matrix and an independent model for the nominal variables. We used such components as they are most closely related to the results of the KSOM algorithms. We then evaluated $\mathcal{F}$ using the resulting mixture models. The resulting averages of $\mathcal{F}$ and the penalty term $\mathcal{D}$ for the different models are presented in Fig. 4.6. Averages were taken over 20 experiments, each time selecting a random subset of 620 of the total 653 data points.

The results support two conclusions. First, the SOMM-ii algorithm obtains higher objective values than both KSOM algorithms. The KSOM algorithm yields better results than the KSOM-$1/n$ algorithm. We also compared the penalty terms obtained with the different algorithms. We see that the SOMM-ii models yield lower penalty terms than the KSOM models. However, the differences in the objective function are not completely explained by the smaller penalty. On average the SOMM-ii models also yield

[9] The mutual information $\mathcal{I}(X;Y)$ between two variables X and Y, jointly distributed according to p is defined in terms of KL divergence as follows: $\mathcal{I}(X;Y) = \mathcal{D}(p(X,Y)\|p(X)p(Y))$.

method	$\mathcal{F}$	$\mathcal{D}$
KSOM	-6070.0 ± 79.3	443.1 ± 20.7
KSOM-$1/n$	-6389.8 ± 81.2	532.5 ± 31.4
SOMM-ii	-5678.8 ± 47.3	297.2 ± 24.4
SOMM-di	-1481.3 ± 118.9	380.3 ± 50.8
SOMM-gi	-1398.3 ± 94.7	411.7 ± 56.9
SOMM-ip	-5147.2 ± 46.7	232.1 ± 13.3
SOMM-dp	-1001.9 ± 81.8	342.8 ± 44.1
SOMM-gp	-1096.4 ± 248.7	342.5 ± 42.4

Figure 4.6: Comparison in terms of the SOMM objective $\mathcal{F}$ and the penalty term $\mathcal{D}$.

a higher likelihood. We conclude that on average the SOMM-ii models yield both a higher likelihood and better topology preservation than the KSOM models. Second, using SOMM models that use more expressive component densities we can obtain considerably higher scores of the objective function. In particular the models that do not assume equal variance in all continuous variables obtain relatively high scores.[10] This indicates that these models give a much better fit on the data and thus that they can provide more realistic descriptions of the credit card applicants in the clusters.

4.5 Conclusions

We presented a constrained EM algorithm to learn self-organizing maps based on mixture models with any type of component densities. Our algorithm is convergent and maximizes an objective function that sums the data log-likelihood and a penalty term. Our mixture model approach to self-organizing maps offers several benefits as compared to existing SOM algorithms. First, the use of mixture models as a basis for SOMs allows for easy design of dissimilarity measures by choosing the component densities based on prior knowledge and/or assumptions. In particular, it is easy to model dependencies between variables, which is non-trivial in the standard approach to self-organizing maps: it would correspond to using dissimilarity measures that are non-additive over the features. Second, our approach can be applied to arbitrary component densities. Third, the EM framework allows one to deal with missing data in a principled way, by estimating in each EM iteration the missing values based on the current parameters of the mixture model.

[10] Note that the non-isotropic models achieve relatively large penalty terms. This is caused by several 'overlapping' mixture components with similar parameters. Consequently, their posteriors are similar which conflicts with the neighborhood function. In the isotropic mixtures this happens less, because overlapping components incur too much loss in data log-likelihood.

5

COMBINING LOCAL LINEAR MODELS TO FORM GLOBAL NON-LINEAR MODELS

In this chapter we consider a method to combine several local linear mappings to obtain a globally non-linear mapping. The method is based on a mixture of Gaussian (MoG) density for which we can estimate the parameters with an EM-like algorithm. First, we use this method for unsupervised non-linear dimension reduction. We contribute an improvement to the existing parameter estimation procedure, replacing an iterative parameter estimation procedure with a closed-form estimation. Second, we consider how this method can used when high dimensional outputs have to be predicted from high dimensional inputs.[1]

5.1 Introduction

Recently various nonparametric spectral methods, such as Isomap (Tenenbaum et al., 2000), LLE (Roweis and Saul, 2000), Kernel PCA (Schölkopf et al., 1998) and Laplacian Eigenmaps (Belkin and Niyogi, 2002) have been proposed (see Section 2.2) which reduce the dimensionality of a fixed training set in a non-linear way that maximally preserves certain inter-point relationships. These methods allow generalization of the mapping from the high dimensional space to the low dimensional space to new data points. We have seen this for kernel PCA in Section 2.2.1, and procedures for the other methods can be found in (Bengio et al., 2004). However, these methods generally do not provide an inverse mapping from the low dimensional space to the high dimensional space.

Linear dimension reduction techniques have several attractive properties: they can be implemented very efficiently and produce a functional mapping, in both directions, between the high and low dimensional space, that generalizes to data not included

[1] Part of the material presented in this chapter is drawn from (Verbeek et al., 2002b; Verbeek et al., 2004a; Verbeek et al., 2003a).

when the mapping was learned. Their practical applicability is limited, however, since in practical situations the data may be confined to a non-linear subspace of the original feature space.

In this chapter, we consider a method to combine several, locally valid, linear mappings between the data space and the space of reduced dimension. The presented method delivers, after training, a non-linear mapping which can be used to convert previously unseen high dimensional observations into their low dimensional global coordinates and vice versa, without the need to store the original training data.

Intuitively, the approach taken in this chapter can be understood as a combination of clustering and principal component analysis (PCA). The idea is illustrated in Fig. 5.1; although globally the data is spread out in three directions, *locally* the data is concentrated around a two dimensional subspace. This allows us, locally, to describe the data accurately by two coordinates in the appropriate subspace (plane). Thus, clusters need to be identified in which the data are concentrated around a low dimensional linear subspace. Several authors, e.g. (Kambhatla and Leen, 1994; Tipping and Bishop, 1999), have reported that such a combination of clustering and PCA allows for significantly better reconstructions of images when they are reconstructed from their PCA projections, as compared to a reconstruction using one single linear PCA subspace. Others (Hinton et al., 1997) successfully used such a 'mixture of PCAs' (MPCA) to model images of hand written digits; an MPCA model was learned on images of each digit $i = 0, \ldots, 9$, yielding density models p_i $(i = 0, \ldots, 9)$. The learned models were used to classify new images $\mathbf{x}$ of handwritten digits by classifying them as digit $i = \arg\max_j p_j(\mathbf{x})$.

The problem with this approach is that each cluster provides a *separate* low dimensional coordinate system that is only used for data in that cluster rather than one — global— low dimensional coordinate system for all data. Thus, if our goal is to find a single low-dimensional representation of the data, e.g. for data visualization, then the combination of clustering and PCA is not sufficient. We need a method to combine the coordinate systems of the different local PCA subspaces into a single low dimensional representation. In Section 2.2.1 we mentioned a method to find principal curves (Verbeek et al., 2002a) that (i) finds clusters of data that are concentrated around a line segment and (ii) connects the line segments to find a piece-wise linear curve around which the complete data set is concentrated. This approach is limited to find one-dimensional representations of the data. The techniques described in this chapter also integrate several local low-dimensional representations, even if the local subspaces have a dimensionality greater than one, as in the example in Fig. 5.1.

In the next section we describe a recently introduced method to integrate the local low dimensional representations. Our contribution in this section is an improvement of the parameter estimation technique, which replaces an iterative procedure with a closed-form solution. We compare this method with several other dimension reduction techniques: PCA, the self-organizing mixture models approach of the previous chapter, and generative topographic mapping.

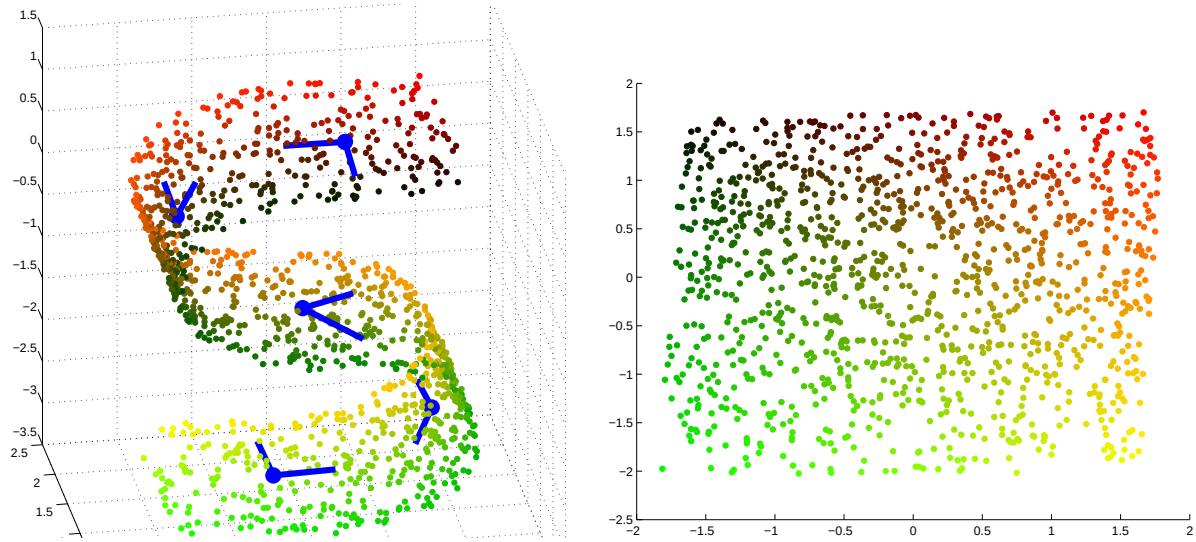


Figure 5.1: Data in $\mathbb{R}^3$ with local two dimensional subspaces indicated by the axes (left). The Desired global two dimensional data representation unfolds the non-linear subspace in which the data resides (right).

In Section 5.3, we consider how this method applies to a setting where one has two sets of high dimensional points in different spaces that are related by several correspondences, i.e. for some points in one set we know that they should have the same low dimensional coordinates as a point in the other set. Non-linear dimension reduction for the two sets of points simultaneously allows us to predict the correspondences for other points through the discovered low dimensional representation. Since this approach also exploits points without a correspondence, good predictions can be made even when only a few correspondences are given. Note that this technique is useful in sensor fusion problems, where each high dimensional set is a collection of measurements from a different sensor. The correspondences are then simultaneously recorded readings from the different sensors; these different sensor readings are thus recorded while the measured system is in a single state. Then, if one sensor fails, the missing high dimensional sensor reading may be reconstructed from the other high dimensional sensor reading by a mapping through the low dimensional space. We extend the approach of Section 5.2 for this new setting and compare it to a more straightforward approach using a mixture of factor analyzers. In Section 5.4 we present our conclusions on the methods considered in this chapter.

5.2 Combining local linear models

In Section 5.2.1 we describe in detail two types of local linear models that can be combined to form global non-linear models: mixtures of probabilistic factor analyzers and

mixtures of probabilistic principal component analyzers. Then, in Section 5.2.2, we describe a method to simultaneously estimate the parameters of such mixtures and integrate the local coordinate systems into a single global representation. In Section 5.2.3 we present our improved parameter estimation procedure. Since the optimization procedure is only guaranteed to find local optima of the objective function, careful initialization of the parameters is required. In Section 5.2.4 we consider several parameter initialization methods. Finally, in Section 5.2.5, we present experimental results obtained with this approach and compare them to results obtained with other methods.

5.2.1 Mixtures of linear models

We assume that the high dimensional data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ is sampled independently and identically from a smooth (non-linear) manifold and is potentially corrupted by some additive Gaussian noise. Thus, the data is distributed on, or near, a low dimensional manifold in a high dimensional space. If the manifold is sufficiently smooth, i.e. *locally* it is almost linearly embedded in the high dimensional space, then we can model the data density with a mixture of densities that concentrate their mass in a linear subspace. To this end we can use mixtures of factor analyzers (MFA) or mixtures of principal component analyzers (MPCA).

Both an MFA and an MPCA density over data vectors $\mathbf{x} \in \mathbb{R}^D$ can be specified by introducing two *hidden* (or unobserved) variables s and $\mathbf{z}$. The first variable, $s \in \{1, \ldots, k\}$, is an index over the k components of the mixture. The second variable $\mathbf{z} \in \mathbb{R}^d$ is a coordinate in the d-dimensional subspace associated with the mixture component with index given by s. Thus, $\mathbf{z}$ may be interpreted as a coordinate on one of the local linear approximations of the manifold. The density over vectors $\mathbf{x}$ follows from the joint density over $(\mathbf{x}, \mathbf{z}, s)$:

$$p(\mathbf{x}, \mathbf{z}, s) = p(\mathbf{x}|\mathbf{z}, s)p(\mathbf{z}|s)p(s), \tag{5.1}$$

$$p(\mathbf{z}|s) = \mathcal{N}(\mathbf{z}; 0, \mathbf{I}), \tag{5.2}$$

$$p(\mathbf{x}|s, \mathbf{z}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_s + \boldsymbol{\Lambda}_s\mathbf{z}, \boldsymbol{\Psi}_s). \tag{5.3}$$

The distribution on $\mathbf{x}$ given s is obtained by marginalizing over $\mathbf{z}$ and given by:

$$p(\mathbf{x}|s) = \int_z p(\mathbf{x}|\mathbf{z}, s)p(\mathbf{z}|s)\ \mathrm{d}\mathbf{z} = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_s, \boldsymbol{\Lambda}_s\boldsymbol{\Lambda}_s^\top + \boldsymbol{\Psi}_s). \tag{5.4}$$

The marginal distribution on $\mathbf{x}$ is then given by the mixture:

$$p(\mathbf{x}) = \sum_{s=1}^{k} p(s)p(\mathbf{x}|s). \tag{5.5}$$

The columns of the matrix $\boldsymbol{\Lambda}_s$ span the subspace of component s which has an offset $\boldsymbol{\mu}_s$ from the origin. The uncertainty of the latent coordinate $\mathbf{z}$ is mapped by $\boldsymbol{\mu}_s$ and $\boldsymbol{\Lambda}_s$

into uncertainty of $\mathbf{x}$ in the subspace. The positive diagonal matrix $\mathbf{\Psi}_s$ adds variance outside the subspace which makes the covariance matrix of $p(\mathbf{x}|s)$ positive definite so that $p(\mathbf{x}|s)$ is a proper density. The MPCA density differs from the MFA density by the restriction that $\mathbf{\Psi}_s = \sigma_s^2\mathbf{I}$.[2]

The subspace spanned by the columns of $\mathbf{\Lambda}_s$ provides a coordinate system which can be used to reconstruct the data from low dimensional coordinates $\mathbf{z} \in \mathbb{R}^d$, at least for data that receives high likelihood under $p(\mathbf{x}|s)$. The joint model $p(\mathbf{x}, \mathbf{z}, s)$ induces a Gaussian density on $\mathbf{z}$ given $\mathbf{x}$ and s:

$$p(\mathbf{z}|\mathbf{x}, s) = \mathcal{N}(\mathbf{z}; \mathbf{\Gamma}_s^{-1}\mathbf{\Lambda}_s^\top \mathbf{\Psi}_s^{-1}(\mathbf{x} - \boldsymbol{\mu}_s), \mathbf{\Gamma}_s^{-1}), \tag{5.6}$$

$$\mathbf{\Gamma}_s = \mathbf{I} + \mathbf{\Lambda}_s^\top \mathbf{\Psi}_s^{-1}\mathbf{\Lambda}_s, \tag{5.7}$$

which can be used to infer the coordinates $\mathbf{z}$ in the local subspace given a data point $\mathbf{x}$ and a mixture component s.

5.2.2 Aligning local linear models

In order to combine the local coordinate systems of each mixture component into a single global coordinate system, we assume that each data point has unique (but unknown) global coordinates $\mathbf{g}$ on the manifold. Furthermore, we assume that locally —in a region that is assigned high likelihood by a single mixture component, say component s— there is a linear transformation that maps the local coordinates $\mathbf{z}$ to the global coordinates $\mathbf{g}$; i.e. locally we have:

$$\mathbf{g} = \boldsymbol{\kappa}_s + \mathbf{A}_s\mathbf{z}, \tag{5.8}$$

for some offset $\boldsymbol{\kappa}_s \in \mathbb{R}^d$ and linear map $\mathbf{A}_s$. If a data point $\mathbf{x}$ is assigned high likelihood by two or more mixture components, then the global coordinates as computed from the different coordinate systems with (5.8) should be approximately the same. In other words: mixture components with high posterior $p(s|\mathbf{x})$ should 'agree' on the global coordinate of point $\mathbf{x}$. Since the local coordinates $\mathbf{z}$ are not known with certainty, a formal notion of 'agreement' between the mixture components needs to take into account the uncertainty in the local coordinates $\mathbf{z}$ as given by $p(\mathbf{z}|\mathbf{x}, s)$.

Conditioned on a mixture component s we have posited a deterministic linear relation between local and global coordinates in (5.8). Therefore, the Gaussian density $p(\mathbf{z}|\mathbf{x}_n, s)$ induces a Gaussian density on $\mathbf{g}$ given $\mathbf{x}_n$ and s:

$$p(\mathbf{g}|\mathbf{x}_n, s) = \mathcal{N}(\mathbf{g}; \mathbf{m}_{ns}, \mathbf{V}_s^{-1}), \tag{5.9}$$

$$\mathbf{V}_s = \mathbf{A}^{-\top}\mathbf{\Gamma}_s\mathbf{A}^{-1} \tag{5.10}$$

$$\mathbf{m}_{ns} = \boldsymbol{\kappa}_s + \mathbf{V}_s^{-1}\mathbf{A}_s^{-\top}\mathbf{\Lambda}_s^\top\mathbf{\Psi}_s^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_s). \tag{5.11}$$

[2] See Section 3.1.4 for more details on the difference between PCA and FA.

Since s is also not known with certainty we marginalize over it to obtain the conditional density on $\mathbf{g}$ given $\mathbf{x}$. This density takes the form of a MoG:

$$p(\mathbf{g}|\mathbf{x}) = \sum_{s=1}^{k} p(s|\mathbf{x}) p(\mathbf{g}|\mathbf{x}, s). \tag{5.12}$$

Thus, if several mixture components with non-negligible posterior $p(s|\mathbf{x})$ for a data point $\mathbf{x}$ yield quite different corresponding densities $p(\mathbf{g}|\mathbf{x}, s)$ on the global coordinates, then they do not 'agree' and the mixture $p(\mathbf{g}|\mathbf{x})$ will exhibit several modes. On the other hand, if all components with non-negligible posterior $p(s|\mathbf{x})$ yield exactly the same density $p(\mathbf{g}|\mathbf{x}, s)$ on the global coordinates, the components are in perfect agreement and the mixture $p(\mathbf{g}|\mathbf{x})$ will be exactly a single Gaussian.

Therefore, to include the notion of agreement in the learning algorithm for the mixture model we can penalize models which —on average— yield multi-modal $p(\mathbf{g}|\mathbf{x}_n)$. To this end we add to the data log-likelihood a penalty term. The penalty term measures how much the mixture $p(\mathbf{g}|\mathbf{x}_n)$ resembles a Gaussian $q_n(\mathbf{g}) = \mathcal{N}(\mathbf{g}; \mathbf{g}_n, \mathbf{\Sigma}_n)$, through the KL divergence $\mathcal{D}(q_n(\mathbf{g}) \| p(\mathbf{g}|\mathbf{x}_n))$. The penalized log-likelihood objective function reads:

$$\mathcal{L}' = \sum_{n=1}^{N} \left[\log p(\mathbf{x}_n) - \mathcal{D}(q_n(\mathbf{g}) \| p(\mathbf{g}|\mathbf{x}_n)) \right] \tag{5.13}$$

$$= \sum_{n=1}^{N} \left[\mathcal{H}(q_n(\mathbf{g})) + \int q_n(\mathbf{g}) \log p(\mathbf{x}_n, \mathbf{g}) \, \mathrm{d}\mathbf{g} \right], \tag{5.14}$$

where $\mathcal{H}$ denotes the entropy of a distribution.[3] Note that the objective $\mathcal{L}'$ does not only depend on the parameters of the mixture model p, but also on the distributions q_n which can be regarded as probabilistic estimates of the global coordinates of the data.

Remarks. Note the similarity between $\mathcal{L}'$ and the objective function (4.3) that we introduced in the previous chapter to enforce self-organization in mixture models. There, we associated a *discrete* hidden variable with each data point —an index over the mixture components— and with each mixture component s we associated a location $\mathbf{g}_s$ in the low dimensional latent space. We used the penalty term $\mathcal{D}(q_n(s) \| p(s|\mathbf{x}_n))$, to encourage the $p(s|\mathbf{x})$ to be 'localized' in the latent space since the q_n were constrained to be of the form $q_n(s) \propto \exp(-\lambda \|\mathbf{g}_s - \mathbf{g}_t\|)$ for some $t \in \{1, \ldots k\}$. In contrast, here we aim to find a mixture of *linear* models such that the smooth density over latent coordinates, $p(\mathbf{g}|\mathbf{x})$, is uni-modal.

The *mixture of experts* approach to classification and regression (Jacobs et al., 1991) is also similar to the method presented in this chapter. This approach constructs a complex (e.g. non-linear) classification or regression function by combining several *experts*.

[3] See Section 3.1.2 for the definitions of entropy and KL divergence.

Each expert implements a simple (e.g. linear) classification or regression function that is suitable for some subset of the data space. To determine which expert should be used to classify a new data point, a *gating network* is used. The gating network produces weighting factors (that are positive and sum to one) by which the predictions of the individual experts are averaged. Of course, the output of both the experts and the gating network depend on the presented data point. Analogously, in the model described above, the non-linear mapping between high dimensional coordinates $\mathbf{x}$ and latent coordinates $\mathbf{g}$ defined in (5.12) is also a weighted average of simple linear Gaussian dependencies $p(\mathbf{g}|\mathbf{x}, s)$ weighted by factors $p(s|\mathbf{x})$ that switch between the 'experts'.

Optimization. In Section 3.1.2 we encountered the problem of local maxima in mixture log-likelihoods and discussed the EM algorithm to identify these local maxima. The second term in the summands of our objective function defined in (5.14) is an expectation of the logarithm of the mixture likelihood $p(\mathbf{x}_n, \mathbf{g}) = \sum_{s=1}^{k} p(s)p(\mathbf{x}_n, \mathbf{g}|s)$, and as a result $\mathcal{L}'$ also exhibits local maxima that are not global. Below, we consider how we can use an EM-like algorithm to (locally) maximize the objective $\mathcal{L}'$. To this end we introduce for each data point a distribution $q_n(s)$ over the mixture components that is used to define a bound on the mixture log-likelihoods:

$$\log p(\mathbf{x}_n, \mathbf{g}) \geq \log p(\mathbf{x}_n, \mathbf{g}) - \mathcal{D}(q_n(s) \| p(s|\mathbf{x}_n, \mathbf{g})) \tag{5.15}$$

$$= \mathcal{H}(q_n(s)) + \sum_{s=1}^{k} q_n(s) \log p(\mathbf{x}_n, \mathbf{g}, s). \tag{5.16}$$

The bounds on the individual mixture log-likelihoods can be combined to bound the complete objective function:

$$\mathcal{L}' \geq \Phi = \sum_{n=1}^{N} \left[\mathcal{H}(q_n(s)) + \mathcal{H}(q_n(\mathbf{g})) + \sum_{s=1}^{k} q_n(s) \int q_n(\mathbf{g}) \log p(\mathbf{x}_n, \mathbf{g}, s) \, \mathrm{d}\mathbf{g} \right]. \tag{5.17}$$

We can now iteratively increase Φ, analogous to the EM algorithm, by maximizing it in turn with respect to the parameters of $q_n(s)$, $q_n(\mathbf{g})$ and p respectively. We coin this EM-like maximization of Φ the Coordinated Factor Analysis (CFA) algorithm. Using the abbreviations, $q_{ns} = q_n(s)$, $\mathbf{x}_{ns} = \mathbf{x}_n - \boldsymbol{\mu}_s$, and $\mathbf{g}_{ns} = \mathbf{g}_n - \boldsymbol{\kappa}_s$, we can write Φ in terms of the parameters as:

$$\Phi = \sum_{n=1}^{N} \sum_{s=1}^{k} q_{ns}[\mathcal{S}_{ns} - \mathcal{E}_{ns}], \tag{5.18}$$

$$\mathcal{S}_{ns} = \frac{1}{2} \log |\boldsymbol{\Sigma}_n| - \log q_{ns} + \frac{d}{2} \log(2\pi), \tag{5.19}$$

$$\mathcal{E}_{ns} = \frac{1}{2} \mathbf{g}_{ns}^{\top} \mathbf{V}_s \mathbf{g}_{ns} + \frac{1}{2} \mathbf{x}_{ns}^{\top} \boldsymbol{\Psi}_s^{-1} \mathbf{x}_{ns} - \mathbf{g}_{ns}^{\top} \mathbf{A}_s^{-\top} \boldsymbol{\Lambda}_s^{\top} \boldsymbol{\Psi}_s^{-1} \mathbf{x}_{ns} + \frac{1}{2} \mathrm{Tr}\{\boldsymbol{\Sigma}_n \mathbf{V}_s\} \tag{5.20}$$

$$+ \frac{1}{2} \log |\boldsymbol{\Psi}_s| + \log |\mathbf{A}_s| - \log p(s) + \frac{D+d}{2} \log(2\pi). \tag{5.21}$$

To maximize Φ with respect to the distributions $q_n(s)$ we equate the corresponding partial derivatives to zero and find the maximizing assignment as:

$$q_{ns} \leftarrow \frac{e^{-\mathcal{E}_{ns}}}{\sum_{s'} e^{-\mathcal{E}_{ns'}}}. \tag{5.22}$$

Similarly, to maximize Φ with respect to the distributions $q_n(\mathbf{g})$ we set:

$$\mathbf{\Sigma}_n^{-1} \leftarrow \sum_s q_{ns}\mathbf{V}_s, \qquad \mathbf{g}_n \leftarrow \mathbf{\Sigma}_n \sum_s q_{ns}\mathbf{V}_s\mathbf{m}_{ns}. \tag{5.23}$$

Given the distributions $q_n(s)$ and $q_n(\mathbf{g})$ we can maximize Φ with respect to the parameters of p. Let $\tilde{q}_{ns} = q_{ns}/\sum_m q_{ms}$, then the maximizing assignments for the mixing weights and offsets are:

$$p(s) \leftarrow \frac{1}{N}\sum_n q_{ns}, \qquad \boldsymbol{\kappa}_s \leftarrow \sum_n \tilde{q}_{ns}\mathbf{g}_n, \qquad \boldsymbol{\mu}_s \leftarrow \sum_n \tilde{q}_{ns}\mathbf{x}_n. \tag{5.24}$$

For the remaining parameters $\{\mathbf{\Lambda}_s, \mathbf{A}_s, \mathbf{\Psi}_s\}_{s=1}^k$ we can again equate the partial derivatives to zero.[4] Using the weighted correlations and covariances:

$$\mathbf{C}_s = \sum_n \tilde{q}_{ns}\mathbf{x}_{ns}\mathbf{g}_{ns}, \qquad \mathbf{G}_s = \sum_n \tilde{q}_{ns}[\mathbf{g}_{ns}\mathbf{g}_{ns}^\top + \mathbf{\Sigma}_n], \tag{5.25}$$

the equations are:

$$\mathbf{\Lambda}_s = \mathbf{C}_s\mathbf{G}_s^{-1}\mathbf{A}_s, \tag{5.26}$$

$$[\mathbf{\Psi}_s]_{ii} = \sum_n \tilde{q}_{ns}\left[\left[\mathbf{x}_{ns} - \mathbf{\Lambda}_s\mathbf{A}_s^{-1}\mathbf{g}_{ns}\right]_i^2 + \left[\mathbf{\Lambda}_s\mathbf{A}_s^{-1}\mathbf{\Sigma}_n\mathbf{A}_s^{-\top}\mathbf{\Lambda}_s^\top\right]_{ii}\right], \tag{5.27}$$

$$\mathbf{A}_s^{-1} = (\mathbf{I} + \mathbf{\Lambda}_s^\top\mathbf{\Psi}_s^{-1}\mathbf{\Lambda}_s)^{-1}(\mathbf{A}_s^\top + \mathbf{\Lambda}_s^\top\mathbf{\Psi}_s^{-1}\mathbf{C}_s)\mathbf{G}_s^{-1}. \tag{5.28}$$

These equations are coupled, and it is not immediately clear how we can find the parameters that jointly satisfy these equations. In (Roweis et al., 2002) the authors propose to use the optimality equations as assignments and iterate them until convergence, i.e. in turn setting the left-hand-side of each of the equations (5.26)–(5.28) to the righthand side of the equation. Clearly, the optimal matrices form a fixed point of these equations. Note that in this manner a 'double-loop' algorithm is obtained. First we loop over the EM steps until convergence and second, within each M-step, we iterate (5.26)–(5.28) as assignments until convergence. Below, we show that parameters that satisfy (5.26)–(5.28) can be found in closed-form, obviating the iterations within the M-steps.

[4] See e.g. (Schönemann, 1985) for some of the required matrix derivatives.

5.2.3 Closed-form update equations

In (Verbeek et al., 2002b) it was already shown that for a restricted version[5] of the probabilistic model p the parameters can be found in closed-form. Here we show that the same holds for the unrestricted model.

If we substitute (5.26) into (5.28), we obtain the equation:

$$\mathbf{A}_s^{-1} = (\mathbf{I} + \mathbf{A}_s^\top \mathbf{G}_s^{-1} \mathbf{C}_s^\top \mathbf{\Psi}_s^{-1} \mathbf{C}_s \mathbf{G}_s^{-1} \mathbf{A}_s)^{-1} (\mathbf{A}_s^\top + \mathbf{A}_s^\top \mathbf{G}_s^{-1} \mathbf{C}_s^\top \mathbf{\Psi}_s^{-1} \mathbf{C}_s) \mathbf{G}_s^{-1}. \quad (5.29)$$

This equation can be further simplified by simple algebraic manipulation to reveal the solutions for $\mathbf{A}_s$:

$$(\mathbf{I} + \mathbf{A}_s^\top \mathbf{G}_s^{-1} \mathbf{C}_s^\top \mathbf{\Psi}_s^{-1} \mathbf{C}_s \mathbf{G}_s^{-1} \mathbf{A}_s) \mathbf{A}_s^{-1} \mathbf{G}_s = \mathbf{A}_s^\top + \mathbf{A}_s^\top \mathbf{G}_s^{-1} \mathbf{C}_s^\top \mathbf{\Psi}_s^{-1} \mathbf{C}_s, \quad (5.30)$$

$$\mathbf{A}_s^{-1} \mathbf{G}_s + \mathbf{A}_s^\top \mathbf{G}_s^{-1} \mathbf{C}_s^\top \mathbf{\Psi}_s^{-1} \mathbf{C}_s = \mathbf{A}_s^\top + \mathbf{A}_s^\top \mathbf{G}_s^{-1} \mathbf{C}_s^\top \mathbf{\Psi}_s^{-1} \mathbf{C}_s, \quad (5.31)$$

$$\mathbf{A}_s^{-1} \mathbf{G}_s = \mathbf{A}_s^\top, \quad (5.32)$$

$$\mathbf{G}_s = \mathbf{A}_s \mathbf{A}_s^\top. \quad (5.33)$$

Once the optimal $\mathbf{A}_s$ is found, corresponding optimal $\mathbf{\Lambda}_s$ and $\mathbf{\Psi}_s$ are found through (5.26) and (5.27). Since $\mathbf{G}_s$ is positive definite by construction, see (5.25), the existence of a matrix $\mathbf{A}_s$ satisfying (5.33) is guaranteed (Horn and Johnson, 1985). Note that for any matrix $\mathbf{A}_s$ that satisfies (5.33) and some unitary matrix $\mathbf{U}$, i.e. with $\mathbf{U}\mathbf{U}^\top = \mathbf{I}$, the matrix $\mathbf{A}_s\mathbf{U}$ also satisfies (5.33). This invariance corresponds to an arbitrary unitary transformation of the local coordinates in the subspace of a mixture component.

We will now show that we do not need to explicitly factorize $\mathbf{G}_s$ as is suggested by (5.33). Let us make the following observations:

1. In all computations needed for the other updates and evaluation of Φ, the matrix $\mathbf{\Lambda}_s$ appears only in products $\mathbf{\Lambda}_s \mathbf{A}_s^{-1}$. For optimal $\mathbf{A}_s$ and $\mathbf{\Lambda}_s$ this product equals $\mathbf{C}_s \mathbf{G}_s^{-1}$, cf. (5.26), and does not depend on the particular factorization that is used.
2. To compute $\mathcal{E}_{ns}$ we need the logarithm of the determinant of $\mathbf{A}_s$. The equality $\log |\mathbf{A}_s| = \frac{1}{2} \log |\mathbf{G}_s|$ shows that this quantity does not depend on the particular factorization.
3. To compute $\mathbf{V}_s$ we need $A_s^{-\top} A_s^{-1}$, which equals $\mathbf{G}_s^{-1}$, again independent of which factorization is used.

Thus, although optimal matrices $\mathbf{A}_s$ are characterized by the factorization of $\mathbf{G}_s$ in (5.33), the actual factorization is not needed in the CFA parameter estimation procedure.

Evaluation of closed-form update equation. Next we consider how our closed-form update compares with the iterative scheme of (Roweis et al., 2002) in terms of computation time and the obtained parameter estimates. For our closed-form update we

[5] The covariance matrix was restricted to be of the k-subspaces type (see Section 3.1.4).

only need to compute $\mathbf{G}_s^{-1}$ and the product $\mathbf{C}_s\mathbf{G}_s^{-1}$. To perform the latter multiplication $O(Dd^2)$ computations are needed. The matrices $\mathbf{G}_s^{-1}$ and $\mathbf{C}_s\mathbf{G}_s^{-1}$, which are fixed throughout the iterations, are also needed when iterating (5.26)–(5.28). In addition, in the iterative approach, the amount of computations needed in each iteration is $O(Dd^2)$, which is needed to compute the different matrix products in the updates equations. Thus, our closed-form update equations require less computations than a single iteration of iterative approach. The factor of speed-up when using our closed-form equations rather than the iterative approach depends on the number of iterations needed by the iterative approach to reach a fixed point.

Below we present results comparing the performance when using the closed-form update equations versus the iterative approach. To start the iterations of (5.26)–(5.28) we initialized $\mathbf{A}_s$ by its value found in the previous M-step, and randomly in the first step. To determine when the iterative procedure has converged we considered the maximum relative change m in the elements a_{ij} of $\mathbf{A}_s$, i.e.

$$m = \max_{i,j} |1 - a_{ij}^{new}/a_{ij}^{old}|. \tag{5.34}$$

We assumed the iterations to have converged if $m < 10^{-4}$.

To our knowledge it is not known whether the iterative procedure is guaranteed to converge. In practice we encountered cases (with $D = 3$ and $d = 2$) where the iterations did not converge within 10^5 iterations. Since (i) the number of iterations needed to converge can be extremely large and (ii) there might be cases where the iterations do not converge at all, it makes sense to limit the number of iterations. If the iterations do not converge, we can simply keep the old parameters found at the last time the iterations converged. In our experiments we found that using a maximum of 100 iterations hardly ever yields convergence in the M-step, a maximum of 1000 yielded convergence in most cases. In our experiments we also noticed that the number of iterations needed for convergence is much larger when the estimates of the $\mathbf{g}_n$ are very poor, e.g. initialized by independent draws form a Gaussian distribution.

To quantify the speed-up of the closed-form equations in practice, and to see whether they lead to better final parameter estimates, we measured: (i) the total time needed for the algorithm to converge[6], (ii) the value of Φ that was attained after convergence and (iii) the number of EM steps required to converge.

We measured the performance when applying the algorithms to two data sets. The first one is the data set of Fig. 5.1 (we will refer to it as the 'S' data set), we initialized the latent coordinates as coordinates on the unrolled 'S'. The second data set is a set of 698 images of 64×64 pixel each of a face seen from different directions and under different lighting conditions, rendered by a computer graphics engine.[7] To speed-up the experiment, the

[6] To measure convergence of the EM algorithm we checked whether the relative change in Φ was smaller than 10^{-4}.

[7] This data set is available online at `http://isomap.stanford.edu`.

Results for the 'S' data set, $N = 1000, D = 3, d = 2, k = 10$**.**				
	# EM steps	time	time/step	Φ
closed-form	123.5 ± 49.5	6.6 ± 3.9	0.05 ± 0.01	0.9 ± 0.6
iterative	125.5 ± 33.8	77.8 ± 29.3	0.61 ± 0.10	0.2 ± 0.8
Results for the isomap face data set, $N = 698, D = 30, d = 3, k = 50$**.**				
	# EM steps	time	time/step	Φ
closed-form	78.4 ± 7.0	30.0 ± 2.8	0.382 ± 0.004	9.7 ± 0.7
iterative	76.7 ± 9.1	198.0 ± 32.7	2.596 ± 0.420	8.1 ± 1.1

Figure 5.2: Performance evaluation of closed-form update equations.

faces were first projected on the 30-dimensional principal components subspace which contains over 90% of the total data variance. The face varies in the direction from which it is seen (two degrees of freedom) and the direction of lighting (one degree of freedom). Here, we initialized the latent coordinates as the known three pose parameters.

The results are summarized in Fig. 5.2 by averages and standard deviations from ten runs. The results show that using the closed-form update equations in practice yields a significantly faster algorithm and higher values for the objective function in a comparable number of EM iterations. We conclude that the closed-form update equations are to be preferred over the iterative procedure proposed in (Roweis et al., 2002).

5.2.4 Parameter initialization

Like many other EM algorithms, the CFA algorithm described above can terminate at poor local optima of the objective function. Careful parameter initialization, especially of the $\mathbf{g}_n$, is needed to successfully use the algorithm. In (Roweis et al., 2002) the authors proposed to initialize the $\mathbf{g}_n$ by using some other unsupervised non-linear dimension reduction technique. We can then apply the CFA algorithm described above until convergence while keeping the $\mathbf{g}_n$ fixed at their initialized value. The $\mathbf{\Sigma}_n$ are also kept fixed at a small multiple of identity and the q_{ns} are initialized at random near uniform.[8] In this manner we learn a MoG on the 'complete' data, i.e. each data vector $\mathbf{x}_n$ is augmented with its estimated global coordinate $\mathbf{g}_n$. After such an initialization phase, the $\mathbf{g}_n$ and $\mathbf{\Sigma}_n$ can be updated as well, to obtain better estimates of consistent global coordinates $\mathbf{g}_n$ and their uncertainties. Note that the overall procedure is still unsupervised.

In principle any non-linear dimension reduction method can be used for the initialization. However, since we need it to avoid poor local optima in the optimization procedure of the last section, the initialization should not suffer from the same problem. The

[8] Alternatively, while keeping the $\mathbf{g}_n$ and $\mathbf{\Sigma}_n$ fixed, a greedy mixture learning approach can be used such as described in Chapter 3.

methods we consider below minimize quadratic error functions, which have only one global minimum which can be identified efficiently.

Initialization using locally linear embedding. In (Roweis et al., 2002) the authors used the locally linear embedding (LLE) algorithm for initialization of the $\mathbf{g}_n$, see Section 2.2.3. Recall that the LLE algorithm consists of two steps. In the first step, for each data point $\mathbf{x}_n$ weights w_{nm} are computed that optimally reconstruct $\mathbf{x}_n$ from a linear combination of a few data points $\mathbf{x}_m$ that are nearest to $\mathbf{x}_n$ in the data space. We use $\mathbf{W}$ to denote the matrix with $[\mathbf{W}]_{nm} = w_{nm}$ and $\mathbf{G}$ to denote the $N \times d$ matrix with $\mathbf{g}_n^\top$ on the n-th row. In the second step of LLE, we use the weights in $\mathbf{W}$ to find the matrix $\mathbf{G}$ with low dimensional coordinates that minimizes:

$$E_{LLE} = \sum_n \|\mathbf{g}_n - \sum_m w_{nm}\mathbf{g}_m\|^2 = \mathrm{Tr}\{\mathbf{G}^\top \mathbf{M}\mathbf{G}\}, \tag{5.35}$$

where $\mathbf{M} = (\mathbf{I}-\mathbf{W})^\top(\mathbf{I}-\mathbf{W})$. Since E_{LLE} is invariant to translations and rotations of the $\mathbf{g}_n$, the low dimensional coordinates are constrained to be zero mean and have identity covariance matrix. The d dimensional coordinates that minimize E_{LLE} can be recovered by computing the $(d+1)$ eigenvectors with smallest eigenvalues of the sparse $N \times N$ matrix $\mathbf{M}$.

Although the matrix $\mathbf{M}$ used by LLE is sparse, its size grows as the number of points in the data sets grows. In (Teh and Roweis, 2003), a constrained version of the LLE algorithm was proposed that avoids the linear growth of the eigenproblem with the number of data points. In this approach first a mixture of factor analyzers (MFA) is fitted on the data. The fitted model is used to constrain the solutions of the LLE algorithm as follows. The MFA gives for each data point $\mathbf{x}_n$ and mixture component s a posterior probability $q_{ns} = p(s|\mathbf{x}_n)$. Furthermore, we can project each data point $\mathbf{x}_n$ on the subspace of the s-th mixture component to obtain $\mathbf{z}_{ns} = \mathbf{\Lambda}_s(\mathbf{x}_n - \boldsymbol{\mu}_s)$. The global coordinates $\mathbf{g}_n$, which are found by LLE, are now constrained to be given by:

$$\mathbf{g}_n = \sum_s q_{ns}\mathbf{g}_{ns}, \qquad \mathbf{g}_{ns} = \boldsymbol{\kappa}_s + \mathbf{A}_s\mathbf{z}_{ns}, \tag{5.36}$$

where $\mathbf{g}_{ns}$ is a linear transformation of $\mathbf{z}_{ns}$. Given the q_{ns} and $\mathbf{z}_{ns}$, the global coordinates only depend on the linear maps $\mathbf{A}_s$ and the offsets $\boldsymbol{\kappa}_s$. To make this linear dependence explicit we write $\mathbf{G}$ as the product of matrices $\mathbf{U}$ and $\mathbf{L}$:

$$\mathbf{G} = \mathbf{U}\mathbf{L}, \qquad \mathbf{U} = \begin{pmatrix} q_{11}[1\ \mathbf{z}_{11}^\top] & \dots & q_{1k}[1\ \mathbf{z}_{1k}^\top] \\ \vdots & \ddots & \vdots \\ q_{N1}[1\ \mathbf{z}_{N1}^\top] & \dots & q_{Nk}[1\ \mathbf{z}_{Nk}^\top] \end{pmatrix}, \qquad \mathbf{L} = \begin{pmatrix} \boldsymbol{\kappa}_1^\top \\ \mathbf{A}_1^\top \\ \vdots \\ \boldsymbol{\kappa}_k^\top \\ \mathbf{A}_k^\top \end{pmatrix}. \tag{5.37}$$

If we now substitute $\mathbf{G} = \mathbf{UL}$ in E_{LLE} from (5.35), we obtain:

$$E_{LLE} = \text{Tr}\{\mathbf{G}^\top \mathbf{M}\mathbf{G}\} = \text{Tr}\{\mathbf{L}^\top \mathbf{U}^\top \mathbf{M}\mathbf{U}\mathbf{L}\}. \tag{5.38}$$

Under the constraints that the $\mathbf{g}_n$ are zero mean and have identity covariance matrix, the gradient of the error function equals zero for vectors $\mathbf{v}$ that satisfy:

$$\mathbf{U}^\top \mathbf{M}\mathbf{U}\mathbf{v} = \lambda \mathbf{U}^\top \mathbf{U}\mathbf{v}. \tag{5.39}$$

Similar to the unconstrained LLE, the optimal linear maps are recovered by computing the $(d+1)$ eigenvectors with smallest eigenvalues. The first eigenvector has eigenvalue $\lambda = 0$ and it has constant value in all the entries for the $\boldsymbol{\kappa}_s$ and zero elsewhere. This eigenvector does not satisfy the variance constraint since it leads to assigning the same coordinate to all the data points. The other eigenvectors do satisfy the constraints and the value of the error function is given by the sum of their corresponding eigenvalues.

Note that the eigenvectors $\mathbf{v}$ are of length $k(d+1)$, the number of local models times one plus the dimensionality of the local subspaces, rather than of length N as in the normal LLE algorithm. However, in order to compute the error function we *do* need to find the $N \times N$ weight matrix $\mathbf{W}$ for which we need to find for each point its nearest neighbors in the data space. Finding the nearest neighbors costs $O(N^2)$ operations in a naive implementation which computes distances for each pair of data points. As mentioned in Section 2.2.4, more efficient techniques exist.

In (Wieghardt, 2001) a similar approach was presented to integrate a fixed set of local linear models. In this approach each data point is assigned to one or more local models, and the *overlap* of two local models s and t is defined as the set of data points that is assigned to both s and t. Then, approximate distances between overlapping local models are estimated on the basis of the points in their overlap. The distances between non-overlapping local models are estimated as the length of the shortest connecting path through overlapping models. Then classical MDS (see Section 2.2.3) is applied to find global low dimensional coordinates $\boldsymbol{\kappa}_s$ for each local model. (Note the similarity with the isomap algorithm, discussed in Section 2.2.3.) Finally, for each local model s a linear transformation from local to global coordinates is found. This is done on the basis of (i) the center of gravity, in local coordinates of model s, of points in the overlap between s and each overlapping model t, and (ii) the difference $\boldsymbol{\kappa}_s - \boldsymbol{\kappa}_t$ in latent coordinates between local model s and all overlapping models t. Compared to the approach of (Teh and Roweis, 2003), this approach is more ad-hoc in the sense that here the optimization is decoupled into several steps which are not optimizing a single objective function.

Initialization without computing nearest neighbors. An alternative method, similar to the constrained LLE algorithm but for which nearest neighbors are not needed, was proposed in (Brand, 2003). As in the constrained LLE algorithm, a mixture of linear models is fitted to the data and for every data point $\mathbf{x}_n$ each mixture component s yields

a posterior probability $q_{ns} = p(s|\mathbf{x}_n)$ and a projection $\mathbf{z}_{ns}$ of $\mathbf{x}_n$ on the subspace associated with component s. Each mixture component s is assigned a linear map that can be used to map local coordinates $\mathbf{z}_{ns}$ to global coordinates, again we have: $\mathbf{g}_{ns} = \mathbf{A}_s\mathbf{z}_{ns} + \boldsymbol{\kappa}_s$.

The error function here is similar to the notion of 'agreement' in Section 5.2.2: for each data point we want the mapping of its local coordinates to the global coordinates — using components with large q_{ns}— to be as close as possible to the (unknown) global coordinate $\mathbf{g}_n$. Given the global coordinates $\mathbf{g}_n$, we can define the error function as the weighted (by the posteriors) sum over all data points $\mathbf{x}_n$ and local models s of the squared distance between the global coordinate $\mathbf{g}_n$ and the local projections $\mathbf{g}_{ns}$:

$$E_{Charting} = \sum_{n=1}^{N}\sum_{s=1}^{k} q_{ns}\|\mathbf{g}_n - \mathbf{g}_{ns}\|^2. \tag{5.40}$$

To find an estimate of the $\mathbf{g}_n$ from given linear maps and corresponding local projections $\mathbf{g}_{ns}$ we can minimize this error with respect to the $\mathbf{g}_n$. The minimizing $\mathbf{g}_n$ are given by a weighted average of the local mappings of $\mathbf{x}_n$ to global coordinates:

$$\mathbf{g}_n = \sum_{s=1}^{k} q_{ns}\mathbf{g}_{ns}. \tag{5.41}$$

We can now substitute the estimated global coordinates of (5.41) into (5.40), such that it becomes a function of the local linear maps only. This yields:

$$E_{Charting} = \frac{1}{2}\sum_{n=1}^{N}\sum_{s=1}^{k}\sum_{t=1}^{k} q_{ns}q_{nt}\|\mathbf{g}_{ns} - \mathbf{g}_{nt}\|^2. \tag{5.42}$$

This error function is quadratic in the linear maps $\mathbf{A}_s$ and offsets $\boldsymbol{\kappa}_s$. We define the block-diagonal matrix $\mathbf{D}$ with k blocks $\mathbf{D}_s$ $(s = 1, \ldots, k)$ as:

$$\mathbf{D} = \begin{pmatrix} \mathbf{D}_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{D}_k \end{pmatrix}, \qquad \mathbf{D}_s = \sum_{n=1}^{N} q_{ns}[\mathbf{z}_{ns}^\top\ 1]^\top[\mathbf{z}_{ns}^\top\ 1]. \tag{5.43}$$

With $\mathbf{U}$ and $\mathbf{L}$ as defined above, the objective can now be written as:

$$E_{Charting} = \mathrm{Tr}\{\mathbf{L}^\top(\mathbf{D} - \mathbf{U}^\top\mathbf{U})\mathbf{L}\}. \tag{5.44}$$

For the same reasons as before, we constrain the $\mathbf{g}_n$ to be zero mean and have unit covariance. The columns of $\mathbf{L}$ yielding zero derivative of $E_{Charting}$ under this constraint are characterized as the generalized eigenvectors:

$$(\mathbf{D} - \mathbf{U}^\top\mathbf{U})\mathbf{v} = \lambda\mathbf{U}^\top\mathbf{U}\mathbf{v}. \tag{5.45}$$

The value of the objective function is given by the sum of the corresponding eigenvalues λ. The smallest eigenvalue is always zero, corresponding to mapping all data into the same latent coordinate. This embedding is uninformative since it is constant, therefore we select the eigenvectors corresponding to the second up to the $(d+1)^{st}$ smallest eigenvalues to obtain the best embedding in d dimensions.

Note that the objective $E_{Charting}$ effectively only takes points into account for which the posterior q_{ns} is non-negligible for at least two mixture components. This can be seen from (5.42), where each term is multiplied by the product of two posteriors $q_{ns}q_{nt}$. Experimentally we found that often only a small percentage of the data points have a posterior distribution with entropy significantly larger than zero if a mixture of FA or PCA is fitted to high dimensional data that is distributed on or near a low dimensional manifold. Therefore, only a small number of data points determine the actual error function. Thus, this method is not very robust, since two different mixtures fitted to the same data might yield quite different sets of points that determine the actual error function and quite different results could be obtained.

Robustness for variation in the mixture fitting can be improved by using several, say C, different mixtures of local models fitted to the same data. To obtain several different mixtures that fit the data well, we can use the EM algorithm and initialize each mixture at different parameter values; the EM algorithm will most probably terminate at different mixture configurations due to the local optima in the data log-likelihood surface. Alternatively, we can train each mixture on a different random subset of, say 90%, of all available data. We can then define a new mixture, p, by averaging over the C mixtures $p_1, \ldots, p_C$; thus $p = \frac{1}{C}\sum_{i=1}^{C} p_i$. Since each individual mixture p_i has to fit all the data, most of the posteriors $p(s|\mathbf{x})$ on components in p will have an entropy significantly larger than zero. Thus in the larger mixture p almost every point will contribute to the associated objective function $E_{Charting}$ and better results are obtained in practice (Verbeek et al., 2003a).

Comparison of coordinate initialization methods. Above we have described several ways to initialize the estimates of the global coordinates of the data. We pointed out that $E_{Charting}$ can become degenerate if the mixture posteriors are near zero entropy. This degeneracy does not occur when using the constrained LLE error function. Even when all points have a posterior with zero entropy, the constrained error function (5.38) is still non-degenerate and can be used successfully. This is an important advantage of using the constrained LLE method.

However, $E_{Charting}$ has the advantage that it can be used in two settings where the LLE algorithm cannot. First, if there are missing values in the data vectors then it is not clear how to compute the nearest neighbors and the reconstruction weights. Second, it is also not clear how to compute the neighbors and weights if the data are not vectors in $\mathbb{R}^D$ but also have discrete valued variables. In both settings we can still fit a mixture model to

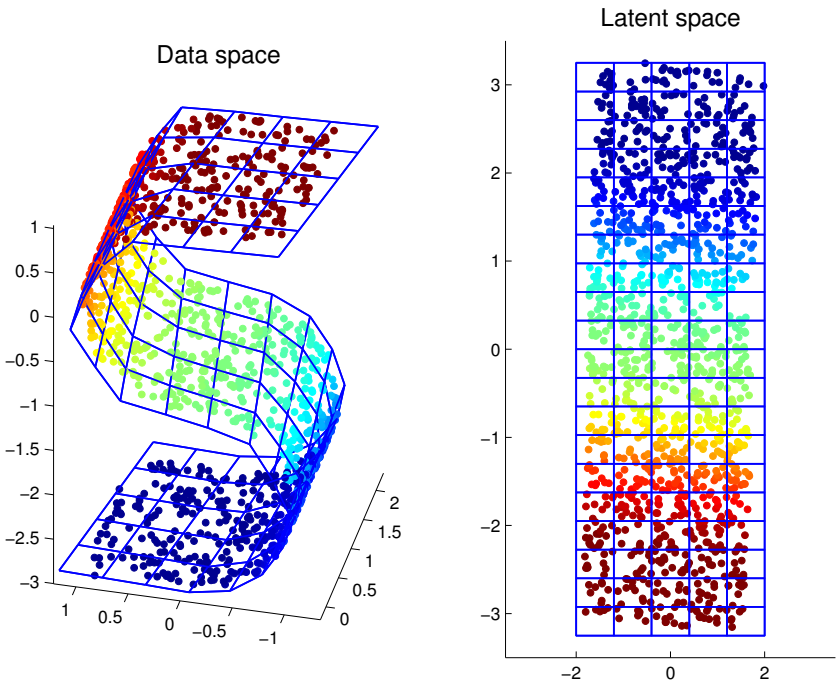


Figure 5.3: Original data in $\mathbb{R}^3$ (top left). Recovered low dimensional coordinates with grid overlay (bottom). Grid points mapped to the data space using model (top right).

the data and compute the posteriors on the mixture components. In the first setting we can also compute projections on local subspaces. In the second setting projections can only be computed from the real-valued variables. If there are no real valued variables at all, it is possible to discard the linear maps $\mathbf{A}_s$ and only optimize for the $\boldsymbol{\kappa}_s$.[9]

For the data sets used in our experiments, which are all treated as sets of vectors in $\mathbb{R}^D$ without missing values, we found that the mixture fitting often takes more time than finding the nearest neighbors. In particular when several mixtures are used to alleviate the problem of low entropy posteriors with $E_{Charting}$, the mixture fitting takes more computation than finding the nearest neighbors. Therefore, in the experiments described in the next section we used initialization based on the LLE error function.

5.2.5 Experimental results

In this section we present results obtained by applying the method described in the previous section to several data sets. First, we present several qualitative experimental results. Second, we present a quantitative comparison of the obtained results with results obtained with self-organizing maps and generative topographic mapping.

[9] In fact, in this case solving for the $\boldsymbol{\kappa}_s$ coincides with the Laplacian eigenmap algorithm, see Section 2.2.3, using the affinity matrix $\mathbf{Q}^\top\mathbf{Q}$, where $[\mathbf{Q}]_{ns} = q_{ns}$.

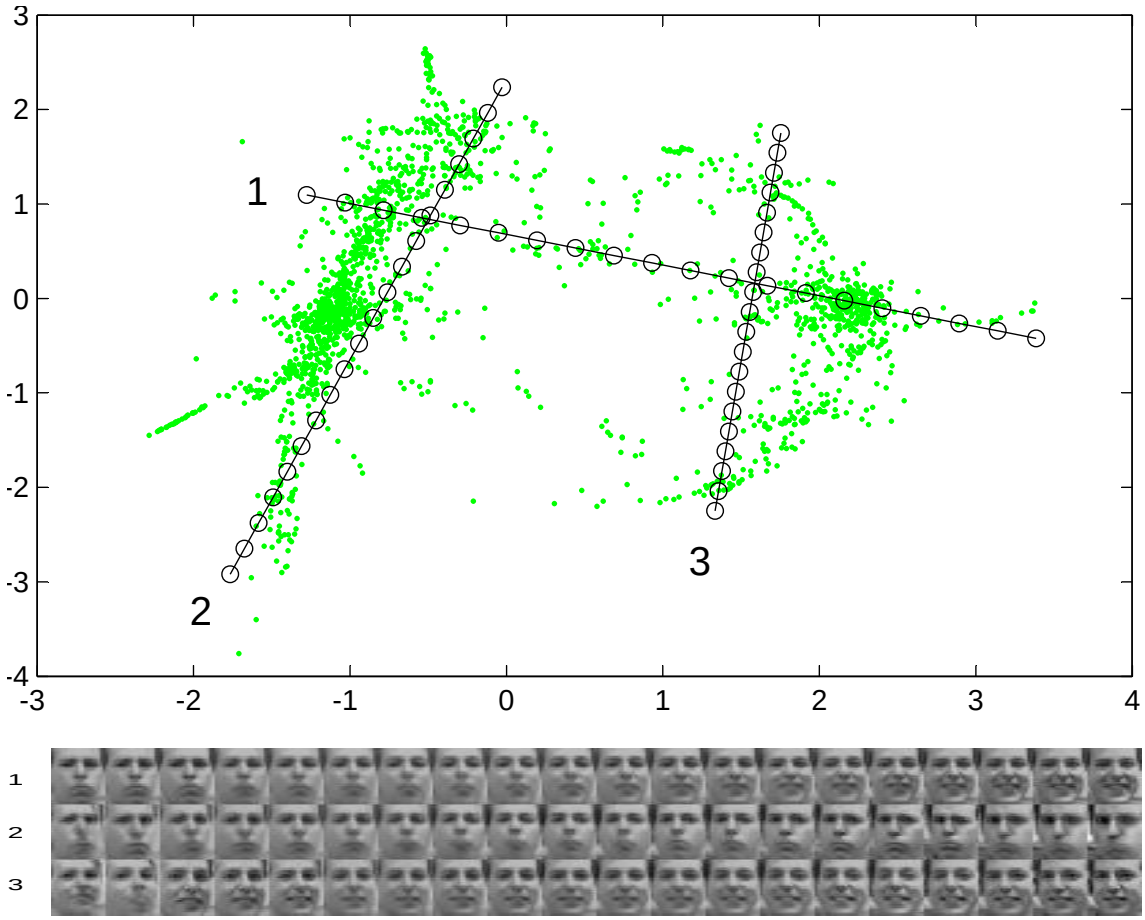


Figure 5.4: Recovered latent representation of the images (top). Images generated with model along the line segments shown in the top panel (bottom).

Qualitative results. The model presented above can be used in two directions. First, the model can be used for dimension reduction, i.e. mapping a point x in the data space to a point g in the latent space. Second, the model can be used for data reconstruction, i.e. mapping a point g in the latent space to a point x in the data space. In Fig. 5.3 we illustrate the two directions in which we can use the model using the 'S' data set of Fig. 5.1. We initialized the latent coordinates with the LLE algorithm using 10 nearest neighbors, and we used a mixture of $k = 10$ factor analyzers. In the left panel of Fig. 5.3 the original three dimensional data is depicted on which the model was trained. The right panel depicts the two-dimensional latent representation of the original data and a rectangular grid in the same space. The left panel depicts the same grid, when mapped to the data space with the trained model. The illustration shows that the non-linear subspace containing the data was indeed recovered based on the training data and that accurate non-linear mapping between the data space and latent space is possible using $k = 10$ locally valid linear mappings.

Next, we show results obtained when applying the model to a data set of images of a

	Reconstruction error E_{rec}.			Log-likelihood$/10^3$.		
	PCA: 0.685 ± 0.055			PCA: -1.61 ± 0.02		
k	CFA	GTM	SOMM	CFA	GTM	SOMM
9	0.00 ± 0.00	0.31 ± 0.01	0.42 ± 0.01	0.19 ± 0.15	-1.82 ± 0.02	-1.92 ± 0.02
16	0.00 ± 0.00	0.18 ± 0.02	0.27 ± 0.04	0.65 ± 0.08	-1.60 ± 0.03	-1.80 ± 0.06
25	0.00 ± 0.00	0.12 ± 0.01	0.18 ± 0.02	0.73 ± 0.12	-1.53 ± 0.03	-1.68 ± 0.04
36	0.00 ± 0.00	0.08 ± 0.01	0.14 ± 0.02	0.74 ± 0.09	-1.45 ± 0.04	-1.56 ± 0.05
49	0.00 ± 0.00	0.09 ± 0.02	0.11 ± 0.02	0.56 ± 0.16	-1.39 ± 0.04	-1.47 ± 0.04
64	0.00 ± 0.00	0.06 ± 0.01	0.08 ± 0.01	0.30 ± 0.20	-1.35 ± 0.04	-1.39 ± 0.03
81	0.00 ± 0.00	0.04 ± 0.01	0.07 ± 0.01	0.01 ± 0.22	-1.34 ± 0.07	-1.34 ± 0.04

Figure 5.5: Results for the 'S' data set.

face, which vary in the pose and expression of the face. This data set consists of 1965 images of 20×28 pixels each. The data set was also used in (Roweis et al., 2002) and we were able to confirm the results reported there.[10] For initialization we used the LLE algorithm with 14 nearest neighbors. The recovered latent representation of the data is illustrated in Fig. 5.4, where each dot represents the coordinates $\mathbf{g}$ of an image. We used the trained model to generate images $\mathbf{x}$ from latent coordinates $\mathbf{g}$ along three straight trajectories in the latent space. The trajectories are plotted in the figure and the corresponding generated images are shown in the bottom panel of Fig. 5.4. The latent coordinates form two clusters; the first trajectory passes through both clusters and the other two trajectories stay within a single cluster. From the generated images along the trajectories we can see that the clusters roughly correspond to images of smiling and non-smiling faces. The reconstructed images along the second and third line segment show that the variation within the clusters corresponds to the variation in gazing direction (left-right) of the faces.

Quantitative comparison. Here we compare CFA with several other dimension reduction methods: PCA, self-organizing mixture models (SOMM), and generative topographic mapping (GTM). To asses the generalization performance of these methods for data not included in the training set, the methods are evaluated using a separate set of test data. We compared the log-likelihood assigned to the test data and the reconstruction error obtained for the test data. The reconstruction error was measured as follows: (i) we map each test data point $\mathbf{x}_n$ to a point $\mathbf{g}_n$ in the latent space and then (ii) we map the $\mathbf{g}_n$ in the latent space back to the data space[11], yielding $\hat{\mathbf{x}}$ and (iii) we measure the

[10] This data set is available online from `http://www.cs.toronto.edu/~roweis`.

[11] For SOMM we mapped latent coordinates back to the data space by assigning the high dimensional coordinates of the mixture component with nearest mean in the latent space. For GTM we map latent coordinates back to the data space using the obtained generalized linear model, see Section 2.2.2.

	Reconstruction error $E_{rec}/10^5$			Log-likelihood$/10^6$		
	PCA: 10.48 ± 0.06.			PCA: -2.998 ± 0.002.		
k	CFA	GTM	SOMM	CFA	GTM	SOMM
9	7.0 ± 0.5	9.3 ± 0.1	9.3 ± 0.1	-2.40 ± 0.01	-3.688 ± 0.003	-3.682 ± 0.004
16	6.2 ± 0.4	8.1 ± 0.1	8.2 ± 0.1	-2.33 ± 0.01	-3.662 ± 0.004	-3.632 ± 0.005
25	5.9 ± 0.7	7.3 ± 0.1	7.4 ± 0.1	-2.29 ± 0.01	-3.644 ± 0.005	-3.591 ± 0.005
36	5.4 ± 0.5	6.8 ± 0.1	6.7 ± 0.1	-2.26 ± 0.01	-3.634 ± 0.005	-3.553 ± 0.006
49	5.4 ± 0.6	6.4 ± 0.1	6.2 ± 0.1	-2.26 ± 0.02	-3.625 ± 0.004	-3.521 ± 0.005
64	5.6 ± 0.7	6.1 ± 0.2	5.8 ± 0.1	-2.28 ± 0.02	-3.619 ± 0.004	-3.495 ± 0.006
81	5.4 ± 0.6	6.0 ± 0.1	5.5 ± 0.1	-2.32 ± 0.02	-3.614 ± 0.004	-3.473 ± 0.007

Figure 5.6: Results for the image data.

squared Euclidean distance between the original test point and its reconstruction. Thus the reconstruction error may be defined as:

$$E_{rec} = \sum_{n=1}^{N} \|\mathbf{x}_n - \hat{\mathbf{x}}_n\|^2. \tag{5.46}$$

All models used a two dimensional latent space, and for GTM and SOMM the nodes were placed on a square rectangular grid in the latent space such that the node locations had zero mean and identity covariance. For GTM and SOMM we use Gaussian component densities with isotropic covariance matrix. For GTM we used as many basis functions as mixture components. The basis functions were of the form $\phi_i(\mathbf{g}) = \exp(-\|\mathbf{g} - \boldsymbol{\kappa}_i\|^2/2\sigma^2)$, with $\sigma^2 = 1/10$.

We performed the comparison on two data sets: the 'S' data set used before (using a training and test set of 600 points) and a data set of 2000 images of 40×40 pixels each of a face looking in different directions (using 1500 training images and 500 test images). The reported results are averages and standard deviations over 20 randomly drawn train and test sets. The results on the 'S' data set are summarized in Fig. 5.5 and those on the second data set in Fig. 5.6. In Fig. 5.7 we plotted some obtained reconstructions (using $k = 81$) for the different models. As expected from the results in Fig. 5.6, the reconstructions of the different methods appear quite similar although the ones produced with CFA look slightly better (in particular those in the second and last column).

Note the overfitting effect for CFA on both data sets. The mean reconstruction error and log-likelihood are optimal for about 36-50 components, using more components actually worsens the results although the models are more expressive. This effect is well known (Webb, 2002) and is caused by the fact that when using many components more parameters must be estimated. However, since the amount of available data is

Figure 5.7: Top row: original images. Reconstructions of images based on two-dimensional representation using CFA (second row), GTM (third row) and SOMM (fourth row).

limited, the parameters cannot be accurately estimated and as a result the model can be erroneous. The overfitting effect is not observed for SOMM and GTM. This can be explained by the fact that these models use fewer parameters than the CFA model, while using the same number of mixture components.

The obtained results lead to the following conclusions. The CFA model is able to achieve significantly higher log-likelihood on the test data than the SOMM and GTM models, while using the same number of mixture components. Compared to the log-likelihood differences, the reduction in reconstruction errors is relatively small. However, with a small number of mixture components the CFA model is able to achieve small reconstruction errors. SOM and GTM need a much larger number of mixture components in order to achieve comparable reconstruction errors since they do not use local linear mappings. The standard deviation in the results obtained with the CFA model is several times larger than in those obtained for SOM and GTM (except for the reconstruction errors on the 'S' data set). This is caused by the sensitivity to the initialization of the CFA model. It is possible that this sensitivity can be resolved by using more robust initialization techniques, e.g. it is possible to minimize the sum of LLE error function using several numbers of neighbors or to use bootstrap-like procedures as in (Ham et al., 2003). Generally, the linear PCA models perform the worst. Interestingly, for the image

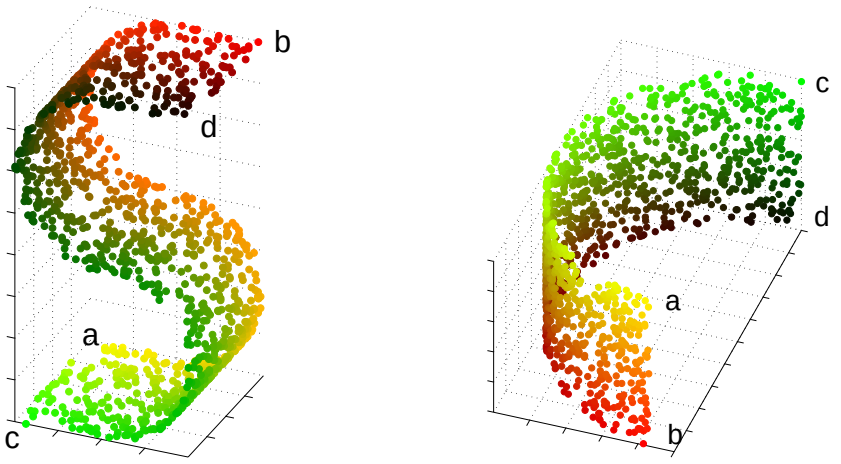


Figure 5.8: Two data sets (left and right panel), the four pairs of corresponding points have been labelled 'a'–'d'.

data the PCA model assigns higher likelihood to the test data than the GTM and SOMM models. This is due to the fact that the data is very high dimensional; if the dimensionality grows the mass of an isotropic Gaussian density contained within a ball of unit radius quickly drops[12]. The PCA model is able to concentrate most mass in the linear subspace with most data variance, and therefore yields a higher likelihood.

5.3 Learning mappings between manifolds

In the previous section we considered the CFA model for non-linear dimension reduction. In this section, we consider how the CFA model applies to the case where multiple embeddings of the same underlying low dimensional manifold are observed, each embedding lying in a different high dimensional data space. So rather than one set of high dimensional data points, we are now given two sets of high dimensional data points: $\{\mathbf{x}_n^1 \in \mathbb{R}^{D_1}\}(n = 1, \ldots, N_1)$ and $\{\mathbf{x}_n^2 \in \mathbb{R}^{D_2}\}$ $(n = 1, \ldots, N_2)$. The two sets are related through a set of correspondences: for some pairs $\mathbf{x}_n^1, \mathbf{x}_m^2$ we know they share the same low dimensional coordinate on the manifold. Fig. 5.8 illustrates this setting: the two data sets are plotted respectively in the left and right panel and the correspondences are indicated by the letters 'a'–'d'. Our goal is to predict the missing correspondences, i.e. for a point in one space without a correspondence we want to estimate the coordinates of the point in the other space that would correspond to it. In other words, we consider a prediction problem where both the inputs and outputs are intrinsically low dimensional, but specified as vectors in a high dimensional space.

[12] For a Gaussian with unit isotropic variance the mass contained in a unit ball is 70% in a one dimensional space, but only 0.02% in a 10 dimensional space (Carreira-Perpiñán, 1997).

One can also think of the data as points in the product space $\mathbb{R}^{D_1} \times \mathbb{R}^{D_2} = \mathbb{R}^{D_1+D_2}$. Corresponding pairs $\mathbf{x}_n^1, \mathbf{x}_m^2$ can be regarded as a point in the product space, the first D_1 coordinates are given by $\mathbf{x}_n^1$ and the remaining D_2 coordinates are given by $\mathbf{x}_m^2$. Points $\mathbf{x}_n^1$ for which we do not have a correspondence can be regarded as points in the product space for which the first D_1 coordinates are given by $\mathbf{x}_n^1$ and the other coordinates are not observed and similarly for points $\mathbf{x}_n^2$ without correspondence.

Note that without loss of generality we can re-order the points such that (i) all the points with a correspondence have a lower index than points without a correspondence and (ii) all corresponding points have the same index, i.e. we only have correspondences for pairs $\mathbf{x}_n^1, \mathbf{x}_m^2$ with $n = m$. The data vectors can be collected into a $N_1 \times D_1$ matrix $\mathbf{X}_1$ and a $N_2 \times D_2$ matrix $\mathbf{X}_2$ which contain the coordinates of the data points as rows. We can then partition the two matrices into a block containing points with correspondences (indexed by c) and a part for points without a correspondence (indexed by w):

$$\mathbf{X}^1 = \begin{pmatrix} \mathbf{X}_c^1 \\ \mathbf{X}_w^1 \end{pmatrix}, \qquad \mathbf{X}^2 = \begin{pmatrix} \mathbf{X}_c^2 \\ \mathbf{X}_w^2 \end{pmatrix}. \tag{5.47}$$

Viewing the data as points in the product space with missing values, the matrix with data in the product space is given by:

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_c^1 & \mathbf{X}_c^2 \\ \mathbf{X}_w^1 & ? \\ ? & \mathbf{X}_w^2 \end{pmatrix}, \tag{5.48}$$

where the question marks indicate the missing values. The goal is now to estimate the missing values. This can be done by estimating a density $p(\mathbf{x}_1, \mathbf{x}^2)$ on the product space, since such a density induces the predictive densities $p(\mathbf{x}^1|\mathbf{x}^2)$ and $p(\mathbf{x}^2|\mathbf{x}^1)$.

The method described in this section is related to the parameterized self-organizing map (PSOM) (Ritter, 1993). The PSOM also produces a mapping between two high dimensional spaces through an underlying low dimensional representation. The basic idea is to first find a low dimensional representation[13] of the set of given input-output pairs $\{(\mathbf{x}_i, \mathbf{y}_i)\}$ $(i = 1, \ldots, N)$ and then to find a vector-valued function $\mathbf{f}$ that optimally reproduces each input-output pair from its low dimensional representation. Thus if $\mathbf{g}_i$ is the vector of low-dimensional coordinates of the i-th pair, then $\mathbf{f}$ is such that for $i = 1, \ldots, N$ we have $\mathbf{f}(\mathbf{g}_i) \approx (\mathbf{x}_i, \mathbf{y}_i)$. The function $\mathbf{f}$ is taken to be a linear combination of non-linear basis functions. To predict the output for a new input $\mathbf{x}$, low dimensional coordinates $\mathbf{g}^*$ are found such that $\mathbf{g}^* = \arg\min_{\mathbf{g}} \|\mathbf{x} - \mathbf{f}_{\mathbf{x}}(\mathbf{g})\|^2$, where $\mathbf{f}_{\mathbf{x}}$ is the function $\mathbf{f}$ restricted to the part corresponding to $\mathbf{x}$. Compared to the CFA approach the PSOM has two drawbacks: (i) PSOM can not use observations without a correspondence and (ii) to map between the two spaces we have to find the low dimensional coordinates $\mathbf{g}^*$

[13] The authors suggest using self-organizing maps.

that optimally reproduce the input. This involves minimizing a non-convex error function, which may produce erroneous results if a local but non-global minimum is found. Arguably, another drawback of PSOM is that for a given input it simply produces an output where the CFA approach produces a density over possible outputs. For these reasons we do not compare the CFA approach with PSOM below.

Another approach to the correspondence problem is to fit a mixture of factor analyzers (MFA) to the (incomplete) data matrix $\mathbf{X}$. The EM algorithm can be used to estimate the parameters in the presence of missing values, see e.g. (Ghahramani and Jordan, 1994).

In Section 5.3.1 we start by considering how the CFA model and learning algorithm should be adapted to apply to this new setting. Then, in Section 5.3.2, we consider initialization techniques suitable for this setting. In Section 5.3.3 we present experimental results which are used to compare the CFA and the MFA approach.

5.3.1 The probabilistic model

The probabilistic model presented in Section 5.2, together with its learning algorithm can be adapted to apply to the current problem. Recall that in Section 5.2 we assumed the densities

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, \mathbf{I}), \tag{5.49}$$
$$p(\mathbf{x}|\mathbf{z}, s) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_s + \boldsymbol{\Lambda}_s \mathbf{z}_s, \boldsymbol{\Psi}_\mathbf{s}). \tag{5.50}$$

Furthermore, conditioned on s, we posited a deterministic relation: $\mathbf{g} = \boldsymbol{\kappa}_s + \mathbf{A}_s \mathbf{g}$, which induces the densities:

$$p(\mathbf{g}|s) = \mathcal{N}(\mathbf{g}; \boldsymbol{\kappa}_s, \mathbf{A}_s \mathbf{A}_s^\top), \tag{5.51}$$
$$p(\mathbf{x}|\mathbf{g}, s) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_s + \boldsymbol{\Lambda}_s \mathbf{A}_s^{-1}(\mathbf{g} - \boldsymbol{\kappa}_s), \boldsymbol{\Psi}_\mathbf{s}). \tag{5.52}$$

In the current setting it is more convenient to parameterize the model in a slightly different manner:

$$p(\mathbf{g}|s) = \mathcal{N}(\mathbf{g}; \boldsymbol{\kappa}_s, \boldsymbol{\Sigma}_s), \tag{5.53}$$
$$p(\mathbf{x}|\mathbf{g}, s) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_s + \boldsymbol{\Lambda}_s(\mathbf{g} - \boldsymbol{\kappa}_s), \boldsymbol{\Psi}_\mathbf{s}). \tag{5.54}$$

These densities induce the conditional densities $p(\mathbf{x}|s)$ and $p(\mathbf{g}|\mathbf{x}, s)$:

$$p(\mathbf{x}|s) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_s, \mathbf{C}_s), \tag{5.55}$$
$$\mathbf{C}_s = \boldsymbol{\Lambda}_s \boldsymbol{\Sigma}_s \boldsymbol{\Lambda}_s^\top + \boldsymbol{\Psi}_\mathbf{s}, \tag{5.56}$$

$$p(\mathbf{g}|\mathbf{x}, s) = \mathcal{N}(\mathbf{g}; \mathbf{m}_s, \mathbf{V}_s^{-1}), \tag{5.57}$$
$$\mathbf{V}_s = \boldsymbol{\Sigma}_s^{-1} + \boldsymbol{\Lambda}_s^\top \boldsymbol{\Psi}^{-1} \boldsymbol{\Lambda}_s, \tag{5.58}$$
$$\mathbf{m}_s = \boldsymbol{\kappa}_s + \mathbf{V}_s^{-1} \boldsymbol{\Lambda}_s^\top \boldsymbol{\Psi}_s^{-1}(\mathbf{x} - \boldsymbol{\mu}_s). \tag{5.59}$$

The marginal $p(\mathbf{x}|s)$ has a $D \times D$ covariance matrix of the factor analysis type and its inverse and determinant can be computed from inverses and determinants of $d \times d$ matrices and diagonal matrices:

$$|\mathbf{C}_s| = |\mathbf{\Psi_s}| \times |\mathbf{\Sigma_s}| \times |\mathbf{V}_s|, \tag{5.60}$$

$$\mathbf{C}_s^{-1} = \mathbf{\Psi}_s^{-1}(\mathbf{\Psi}_s - \mathbf{\Lambda}_s \mathbf{V}_s^{-1} \mathbf{\Lambda}_s^\top)\mathbf{\Psi}_s^{-1}. \tag{5.61}$$

Since we do not know the missing values, the objective function Φ defined in (5.17) in the previous section is modified by basing it on the observed variables only. If $\mathbf{x}_i^o$ indicates the part of $\mathbf{x}_i$ that was observed (either the D_1 first variables, the last D_2 variables, or all variables), then the modified objective function reads:

$$\Phi = \sum_{n=1}^{N} \left[\mathcal{H}(q_n(s)) + \mathcal{H}(q_n(\mathbf{g})) + \sum_{s=1}^{k} q_n(s) \int q_n(\mathbf{g}) \log p(\mathbf{x}_n^o, \mathbf{g}, s) \, \mathrm{d}\mathbf{g} \right]. \tag{5.62}$$

To write Φ in terms of the model parameters we use the superscripts 1 and 2 to indicate the sub-matrices and sub-vectors related to the first and second data space. Thus $\mathbf{x}_i^1$ indicates the first D_1 coordinates of vector $\mathbf{x}_1$ and similarly $\mathbf{\Lambda}_s^2$ denotes the sub-matrix given by the last D_2 rows of $\mathbf{\Lambda}_s$. We write Φ as a sum of entropy terms $\mathcal{S}_{ns}$ and energy terms $\mathcal{E}_{ns}$, using the abbreviations $\mathbf{x}_{ns}^1 = \mathbf{x}_n^1 - \boldsymbol{\mu}_s^1$, and similarly for $\mathbf{x}_{ns}^2$, and $\mathbf{g}_{ns} = \mathbf{g}_n - \boldsymbol{\kappa}_s$:

$$\Phi = \sum_{n=1}^{N} \sum_{s=1}^{k} q_{ns}[\mathcal{S}_{ns} - \mathcal{E}_{ns}], \tag{5.63}$$

$$\mathcal{S}_{ns} = \frac{1}{2} \log |\mathbf{\Sigma}_n| - \log q_{ns} + \frac{d}{2} \log(2\pi), \tag{5.64}$$

$$\mathcal{E}_{ns} = -\log p(s) + \frac{1}{2} \log |\mathbf{\Sigma}_s| + \frac{1}{2} \mathrm{Tr}\{\mathbf{\Sigma}_s^{-1}(\mathbf{\Sigma}_n + \mathbf{g}_{ns}\mathbf{g}_{ns}^\top)\} \tag{5.65}$$

$$+ \frac{1}{2}(\mathbf{x}_{ns}^1 - \mathbf{\Lambda}_s^1 \mathbf{g}_{ns})^\top \mathbf{\Psi}_s^{1-1}(\mathbf{x}_{ns}^1 - \mathbf{\Lambda}_s^1 \mathbf{g}_{ns}) + \frac{1}{2} \mathrm{Tr}\{\mathbf{\Sigma}_n \mathbf{\Lambda}_s^{1\top} \mathbf{\Psi}_s^{1-1} \mathbf{\Lambda}_s^1\} + \frac{1}{2} |\mathbf{\Psi}_s^1| \tag{5.66}$$

$$+ \frac{1}{2}(\mathbf{x}_{ns}^2 - \mathbf{\Lambda}_s^2 \mathbf{g}_{ns})^\top \mathbf{\Psi}_s^{2-1}(\mathbf{x}_{ns}^2 - \mathbf{\Lambda}_s^2 \mathbf{g}_{ns}) + \frac{1}{2} \mathrm{Tr}\{\mathbf{\Sigma}_n \mathbf{\Lambda}_s^{2\top} \mathbf{\Psi}_s^{2-1} \mathbf{\Lambda}_s^2\} + \frac{1}{2} |\mathbf{\Psi}_s^2|. \tag{5.67}$$

The second (respectively third) line of the expression for $\mathcal{E}_{ns}$ should not be added for points of which the first D_1 (respectively last D_2) coordinates are not observed.

Next we consider the parameter update equations of the EM-like algorithm. For $\mathbf{\Sigma}_n$ and $\mathbf{g}_n$ the updates become:

$$\mathbf{\Sigma}_n^{-1} \leftarrow \sum_{s=1}^{k} q_{ns} \left[\mathbf{\Sigma}_s^{-1} + \mathbf{\Lambda}_s^{1\top} \mathbf{\Psi}_s^{1-1} \mathbf{\Lambda}_s^1 + \mathbf{\Lambda}_s^{2\top} \mathbf{\Psi}_s^{2-1} \mathbf{\Lambda}_s^2 \right], \tag{5.68}$$

$$\mathbf{g}_n \leftarrow \mathbf{\Sigma}_n \sum_{s=1}^{k} q_{ns} \left[\mathbf{\Sigma}_s^{-1} \boldsymbol{\kappa}_s + \mathbf{\Lambda}_s^{1\top} \mathbf{\Psi}_s^{1-1}(\mathbf{x}_{ns}^1 + \mathbf{\Lambda}_s^1 \boldsymbol{\kappa}_s) + \mathbf{\Lambda}_s^{2\top} \mathbf{\Psi}_s^{2-1}(\mathbf{x}_{ns}^2 + \mathbf{\Lambda}_s^2 \boldsymbol{\kappa}_s) \right]. \tag{5.69}$$

Note that for the updates related to a particular data point, only the terms related to the observed coordinates should be added. The update equations for $q_{ns}, p(s)$ and $\boldsymbol{\kappa}_s$ remain as before, thus using $\tilde{q}_{ns} = q_{ns} / \sum_m q_{ms}$, the updates are:

$$q_{ns} \leftarrow \frac{e^{-\mathcal{E}_{ns}}}{\sum_{s'} e^{-\mathcal{E}_{ns'}}}, \qquad p(s) \leftarrow \frac{1}{N} \sum_n q_{ns}, \qquad \boldsymbol{\kappa}_s \leftarrow \sum_n \tilde{q}_{ns} \mathbf{g}_n. \tag{5.70}$$

The other updates are given below:

$$\boldsymbol{\Sigma}_s = \sum_n \tilde{q}_{ns} \left[\boldsymbol{\Sigma}_n + \mathbf{g}_{ns} \mathbf{g}_{ns}^\top \right], \tag{5.71}$$

$$\boldsymbol{\Lambda}_s^1 = \left[\sum_n q_{ns} \mathbf{x}_{ns}^1 \mathbf{g}_{ns}^\top - \sum_n q_{ns} \mathbf{x}_{ns}^1 \times \frac{\sum_n q_{ns} \mathbf{g}_{ns}^\top}{\sum_n q_{ns}} \right] \tag{5.72}$$

$$\times \left[\sum_n q_{ns} \left[\boldsymbol{\Sigma}_n + \mathbf{g}_{ns} \mathbf{g}_{ns}^\top \right] - \sum_n q_{ns} \mathbf{g}_{ns} \times \frac{\sum_n q_{ns} \mathbf{g}_{ns}^\top}{\sum_n q_{ns}} \right]^{-1}, \tag{5.73}$$

$$\boldsymbol{\mu}_s^1 = \frac{1}{\sum_n q_{ns}} \sum_n q_{ns} \left[\mathbf{x}_n^1 - \boldsymbol{\Lambda}_s^1 \mathbf{g}_{ns} \right], \tag{5.74}$$

$$\left[\boldsymbol{\Psi}_s^1 \right]_i = \frac{1}{\sum_n q_{ns}} \sum_n q_{ns} \left[\mathbf{x}_{ns}^1 - \boldsymbol{\Lambda}_s^1 \mathbf{g}_{ns} \right]_i^2 + \left[\boldsymbol{\Lambda}_s^1 \boldsymbol{\Sigma}_n {\boldsymbol{\Lambda}_s^1}^\top \right]_{ii} \tag{5.75}$$

In the last three updates the sums over data points run only over points for which the first D_1 coordinates are observed. The updates for $\boldsymbol{\mu}_s^2, \boldsymbol{\Lambda}_s^2$ and $[\boldsymbol{\Psi}_s^2]_i$ are analogous.

5.3.2 Parameter initialization

As before, proper initialization of the $\mathbf{g}_n$ is crucial in order to find good parameter estimates with our EM-like algorithm. Below, we consider how the initialization methods of the previous section apply to the current setting.

Parameter initialization without computing nearest neighbors. The parameter initialization method based on $E_{Charting}$, presented in Section 5.2.4, can be extended to the current setting as follows. Recall that the method based on $E_{Charting}$ consists of two steps. First, a mixture of linear models is fitted to the data to yield for each data point and mixture component a responsibility q_{ns} and a projection $\mathbf{z}_{ns}$ of $\mathbf{x}_n$ on the subspace of component s. Second, the quadratic error function $E_{Charting}$ (which is based on the q_{ns} and $\mathbf{z}_{ns}$) is minimized to find optimal linear maps that project the coordinates in the component subspaces to global low dimensional coordinates. The global low dimensional coordinates $\mathbf{g}_n$ for a point $\mathbf{x}_n$ are given by the weighted average of the projections

$\mathbf{g}_{ns}$ with the different components:

$$\mathbf{g}_n = \sum_{s=1}^{k} q_{ns}\mathbf{g}_{ns} = \sum_{s=1}^{k} q_{ns}[\boldsymbol{\kappa}_s + \mathbf{A}_s\mathbf{z}_{ns}]. \tag{5.76}$$

In the current setting where we have two sets of points, we can proceed as before for each point set separately. However, in order to take into account the correspondence between pairs $\mathbf{x}_n^1$ and $\mathbf{x}_n^2$, we want the latent coordinates $\mathbf{g}_n^1$ and $\mathbf{g}_n^2$ computed from $\mathbf{x}_n^1$ and $\mathbf{x}_n^2$ to be as similar as possible. Thus, we can use an error function that adds $E_{Charting}$ based on each point set separately and for each correspondence a term that penalizes the squared distance between $\mathbf{g}_n^1$ and $\mathbf{g}_n^2$:

$$E = E_{Charting}^1 + E_{Charting}^2 + \sum_n \|\mathbf{g}_n^1 - \mathbf{g}_n^2\|^2, \tag{5.77}$$

where the last sum runs over corresponding pairs $\mathbf{x}_n^1$, $\mathbf{x}_n^2$. This error function is again quadratic in the linear maps assigned to the mixture components and can be minimized with the same techniques that were used in the previous section.

LLE based parameter initialization. In the current setting we can not directly apply the LLE based parameter initialization used before because the missing values do not allow us to compute nearest neighbors and reconstruction weights in the usual manner.

In (Ham et al., 2003) an approach was proposed to extend the LLE algorithm to the current setting. The idea is to perform LLE on both sets of points $\mathbf{X}^1$ and $\mathbf{X}^2$ separately, but to constrain the latent coordinates of corresponding points to be identical. Note that for each set $\mathbf{X}^1$ and $\mathbf{X}^2$ we can compute the nearest neighbors and reconstruction weights as in the first step of the standard LLE algorithm. Recall that the standard LLE error function is:

$$E_{LLE} = \sum_{n=1}^{N} \|\mathbf{g}_n - \sum_{m=1}^{N} w_{nm}\mathbf{g}_m\|^2 = \mathrm{Tr}\{\mathbf{G}^\top\mathbf{M}\mathbf{G}\}. \tag{5.78}$$

Let $E_{LLE}^1 = \mathrm{Tr}\{\mathbf{G}^{1\top}\mathbf{M}^1\mathbf{G}^1\}$ be the LLE error function based on $\mathbf{X}^1$ and similarly for E_{LLE}^2. Our goal is to minimize $E_{LLE}^1 + E_{LLE}^2$ subject to the constraint that corresponding points have the same latent coordinates, and the usual constraint that the latent coordinates are zero mean and have identity covariance. If we partition the matrix $\mathbf{G}^1$ with latent coordinates for $\mathbf{X}^1$ into blocks $\mathbf{G}_c^1$ and $\mathbf{G}_w^1$ as:

$$\mathbf{G}^1 = \begin{bmatrix} \mathbf{G}_c^1 \\ \mathbf{G}_w^1 \end{bmatrix}, \tag{5.79}$$

and similarly for $\mathbf{G}^2$, then the correspondence constraint is that $\mathbf{G}_c^1 = \mathbf{G}_c^2$. We can use a similar partition of the matrices $\mathbf{M}^1$ and $\mathbf{M}^2$:

$$\mathbf{M}^1 = \begin{bmatrix} \mathbf{M}_{cc}^1 & \mathbf{M}_{cw}^1 \\ \mathbf{M}_{wc}^1 & \mathbf{M}_{ww}^1 \end{bmatrix}, \qquad \mathbf{M}^2 = \begin{bmatrix} \mathbf{M}_{cc}^2 & \mathbf{M}_{cw}^2 \\ \mathbf{M}_{wc}^2 & \mathbf{M}_{ww}^2 \end{bmatrix}. \tag{5.80}$$

It is not difficult to show that the constrained minimization problem is solved by computing the $(d+1)$ eigenvectors of:

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_{cc}^1 + \mathbf{M}_{cc}^2 & \mathbf{M}_{cw}^1 & \mathbf{M}_{cw}^2 \\ \mathbf{M}_{wc}^1 & \mathbf{M}_{ww}^1 & \mathbf{0} \\ \mathbf{M}_{wc}^2 & \mathbf{0} & \mathbf{M}_{ww}^2 \end{bmatrix}, \tag{5.81}$$

with smallest eigenvalues. As before, the eigenvector with smallest eigenvalue (zero) contains a degenerate solution and is discarded. The final solution is then given by letting the other d eigenvectors be the columns of:

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_c \\ \mathbf{G}_w^1 \\ \mathbf{G}_w^2 \end{bmatrix}. \tag{5.82}$$

Comparison of initialization methods. In the previous section we saw that the error function $E_{Charting}$ becomes degenerate when the entropy in the posterior probabilities tends to zero. The same degeneracy appears in the current setting. In the current setting, the LLE based error function can also become degenerate. However, this only happens if the dimensionality of the latent space is greater than, or equal to, the number of correspondences (which is unlikely since usually this dimensionality is very small).

Another difference is that in the LLE based method a *constraint* is used to account for the correspondences, while in the method based on $E_{Charting}$ the penalty term that encodes the correspondences is *added* to the separate error function of each set of points. As a result, if there are only very few correspondences, the penalty term is relatively unimportant as compared to the other error functions and some weighting factor has to be introduced to increase the influence of the penalty term. The value of the weighting factor introduces a free parameter, and it is not clear how to set it automatically.

In our experiments below, we have used the LLE based initialization method because it can be used when only very few correspondences are given and does not involve weighting of the different errors.

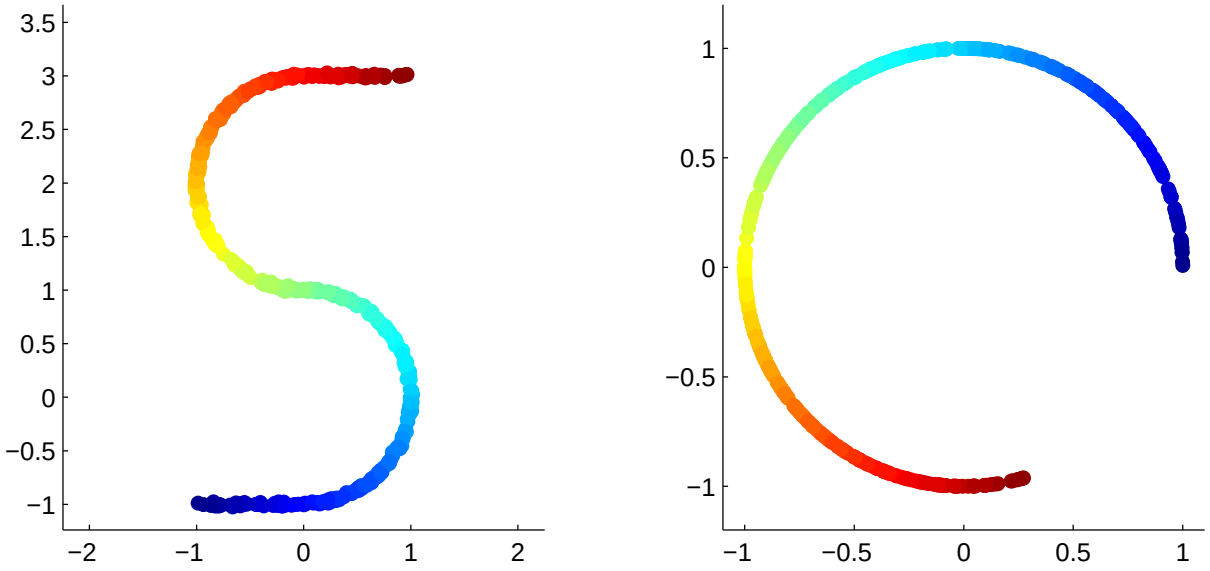


Figure 5.9: Data sets $\mathbf{X^1}$ (left panel) and $\mathbf{X^2}$ (right panel). If both curves are stretched out on a line between zero and one, then corresponding points have the same position. Corresponding points are filled identically in the figures.

5.3.3 Experimental results

In this section we experimentally compare CFA and MFA models to predict high dimensional correspondences. We investigate how the performance of the approaches depends on the number of given correspondences and the number of mixture components. First, we compare prediction accuracy on two synthetic data sets to gain insight into the differences between the two approaches. Then we compare them using a data set consisting of gray scale images of two objects to determine if the differences observed with the synthetic data sets are also apparent in more realistic data sets.

The MFA density model was described in Section 5.2.1, and we used the EM algorithm for mixtures of factor analyzers (Ghahramani and Hinton, 1996) to find parameters that correspond to a (local) maximum of the data log-likelihood.

Synthetic data sets. In the first experiment we consider two data sets $\mathbf{X^1}$ and $\mathbf{X}^2$ that are both distributed along a curve embedded in $\mathbb{R}^2$, illustrated in Fig. 5.9. Both sets contain 1240 points, in each experiment we used a training set of 200 points from each set and a second set of 200 corresponding points to assess the quality of the fitted models. The results reported below are averages over 20 experiments.

We trained CFA and MFA models using a latent dimensionality of $d = 1$, while varying the percentage c of training data for which correspondences are available (ranging from $c = 5\%$ up to $c = 55\%$ in 5% intervals, i.e. using from 10 up to 110 correspondences)

k	$c = \frac{1}{20}$		$\frac{3}{20}$		$\frac{5}{20}$		$\frac{7}{20}$		$\frac{9}{20}$		$\frac{11}{20}$	
1	**0.64**	**0.83**	0.59	0.60	**0.59**	**0.62**	**0.57**	**0.58**	**0.53**	**0.54**	**0.51**	**0.52**
2	**0.22**	**0.69**	0.20	0.26	0.18	0.21	0.17	0.18	**0.16**	**0.17**	0.16	0.17
3	**0.09**	**0.88**	**0.10**	**0.20**	**0.08**	**0.11**	0.08	0.11	0.07	0.07	0.07	0.07
4	**0.04**	**0.93**	**0.05**	**0.19**	0.04	0.05	0.04	0.05	0.04	0.05	0.04	0.03
5	**0.02**	**1.05**	**0.02**	**0.09**	0.02	0.08	0.03	0.10	0.02	0.03	0.02	0.02
6	**0.04**	**1.03**	**0.01**	**0.34**	**0.01**	**0.05**	0.03	0.06	0.01	0.03	0.01	0.01
7	**0.03**	**0.79**	**0.01**	**0.19**	**0.01**	**0.06**	**0.01**	**0.06**	0.01	0.05	0.01	0.02
8	**0.09**	**1.03**	**0.01**	**0.21**	**0.01**	**0.16**	**0.01**	**0.06**	**0.00**	**0.06**	**0.00**	**0.02**
9	**0.26**	**1.01**	**0.01**	**0.31**	**0.00**	**0.15**	**0.00**	**0.09**	**0.01**	**0.14**	0.01	0.01
10	0.49	0.99	**0.01**	**0.25**	**0.01**	**0.14**	**0.00**	**0.08**	**0.00**	**0.02**	**0.00**	**0.02**

Figure 5.10: Results (CFA left and MFA right in each column) for the first synthetic data set.

and the number k of mixture components (ranging from $k = 1$ to $k = 10$). To asses the quality of the fitted models we used the test data, for which all 4 coordinates are known. For the test data we predicted the last two coordinates given the first two coordinates with the trained models, and vice versa.

When using an MFA, each mixture component s is a Gaussian density $p(\mathbf{x}|s)$ on the product space which induces a conditional Gaussian on the coordinates in the second space given the coordinates in the first space: $p(\mathbf{x}^2|\mathbf{x}^1, s)$. Combining the different components we obtain the conditional mixture density $p(\mathbf{x}^2|\mathbf{x}^1) = \sum_{s=1}^{k} p(\mathbf{x}^2|\mathbf{x}^1, s)p(s|\mathbf{x}^1)$. Hence, the expected value of $\mathbf{x}^2$ under this distribution, which we denote by $\hat{\mathbf{x}}^2$, is the sum of the means of $p(\mathbf{x}^2|\mathbf{x}^1, s)$ $(s = 1, \ldots, k)$ where each mean is weighted by the corresponding posterior probability $p(s|\mathbf{x}^1)$. The same analysis holds for the CFA models and the predictions on $\mathbf{x}^1$ given $\mathbf{x}^2$.

As an error measure of the models we used the squared difference between the predicted coordinates $\hat{\mathbf{x}}^1$ (or $\hat{\mathbf{x}}^2$) and the true coordinates of $\mathbf{x}^1$ (or $\mathbf{x}^2$), averaged over all test data points and over the coordinates. Thus for all $N = 200$ test points we measure:

$$E_{rec} = \frac{1}{ND_1} \sum_{n=1}^{N} \|\hat{\mathbf{x}}_n^1 - \mathbf{x}_n^1\| + \frac{1}{ND_2} \sum_{n=1}^{N} \|\hat{\mathbf{x}}_n^2 - \mathbf{x}_n^2\|. \tag{5.83}$$

In Fig. 5.10 we tabulated, for different numbers of correspondences and mixture components, the error E_{rec} obtained with CFA and MFA. Results where the CFA error is significantly[14] smaller are printed in bold.

Results of the same experiment using the data set shown in Fig. 5.8 are tabulated in Fig. 5.11. In these experiments the latent dimensionality was $d = 2$. Of the total 1240

[14] To determine significance we used a t-test with 19 degrees of freedom and $p = 0.05$.

k	$c = \frac{1}{20}$		$\frac{3}{20}$		$\frac{5}{20}$		$\frac{7}{20}$		$\frac{9}{20}$		$\frac{11}{20}$	
1	0.19	0.19	**0.19**	**0.19**	0.19	0.19	0.19	0.19	**0.19**	**0.19**	**0.19**	**0.19**
2	**0.27**	**0.74**	**0.18**	**0.34**	0.16	0.18	**0.16**	**0.18**	0.16	0.17	0.16	0.16
3	**0.34**	**0.98**	0.21	0.33	**0.13**	**0.19**	0.13	0.18	**0.13**	**0.15**	**0.13**	**0.15**
4	**0.34**	**1.26**	**0.14**	**0.57**	**0.12**	**0.25**	**0.12**	**0.16**	**0.11**	**0.17**	**0.10**	**0.14**
5	**0.22**	**1.13**	**0.15**	**0.51**	**0.08**	**0.30**	**0.09**	**0.20**	**0.10**	**0.18**	**0.08**	**0.16**
6	**0.25**	**1.45**	**0.13**	**0.71**	**0.08**	**0.37**	**0.08**	**0.23**	**0.07**	**0.18**	**0.06**	**0.16**
7	**0.25**	**1.40**	**0.09**	**0.67**	**0.09**	**0.40**	**0.09**	**0.23**	**0.06**	**0.20**	**0.05**	**0.14**
8	**0.23**	**1.28**	**0.09**	**0.67**	**0.06**	**0.42**	**0.06**	**0.26**	**0.05**	**0.19**	**0.05**	**0.13**
9	**0.20**	**1.30**	**0.10**	**0.63**	**0.05**	**0.40**	**0.05**	**0.26**	**0.05**	**0.17**	**0.05**	**0.14**
10	**0.18**	**1.20**	**0.08**	**0.62**	**0.06**	**0.43**	**0.05**	**0.26**	**0.04**	**0.19**	**0.04**	**0.15**

Figure 5.11: Results (CFA left and MFA right in each column) for the second synthetic data set.

data points in each set, 600 random points were used to fit the models and another 600 random pairs of corresponding points were used to asses the error of the model. Again, significantly smaller errors are printed bold.

The results for both data sets show a similar pattern. If many correspondences are available, CFA and MFA give comparable results. If only a few correspondences are given CFA performs significantly better than MFA. This difference is more pronounced as the number of mixture components becomes larger. The effect is observed more clearly in Fig. 5.12 where we plotted the average error differences against the number of correspondences divided by the number of mixture components.

The explanation of the effect is that if the average number of correspondences per mixture component becomes very low, then in an MFA model there can be mixture components s which have (almost) no responsibility for correspondences (i.e. observations without missing values). In such cases it is not possible to determine the dependencies between coordinates in the two different spaces, since for each data point with non-negligible responsibility for the component s either only the first D_1 coordinates are observed or only the last D_2 coordinates. In comparison, the CFA model can exploit the incomplete observations to determine these dependencies; this is possible through the dependencies between the incomplete observations and the estimated global low dimensional coordinates on the manifold. To illustrate the difference in predictions using a few correspondences and relatively many components, we plotted the obtained predictions on the test set in Fig. 5.13. These predictions were obtained on the first synthetic data set, using $k = 10$ mixture components and four correspondences ($c = \frac{2}{100}$).

Mapping between views of different objects. In the second experiment we also compare the reconstruction errors using MFA and CFA models, but we use a more realistic

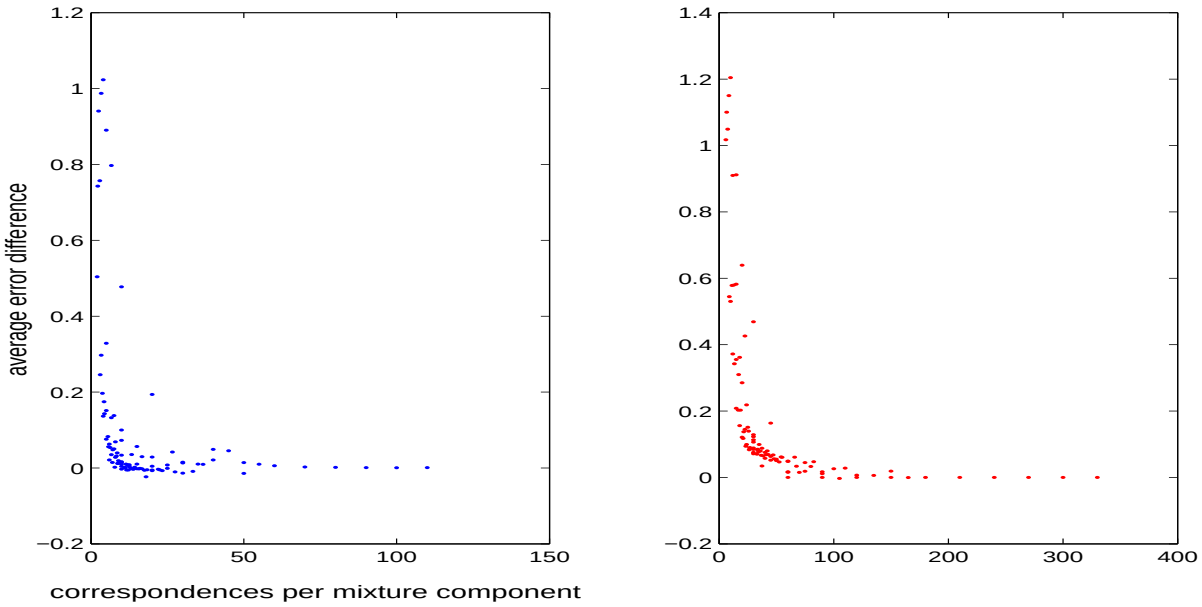


Figure 5.12: Average error of MFA minus average error of CFA on test set, plotted as function of the number of correspondences per mixture component. Left and right panel show results for respectively the first and the second synthetic data set.

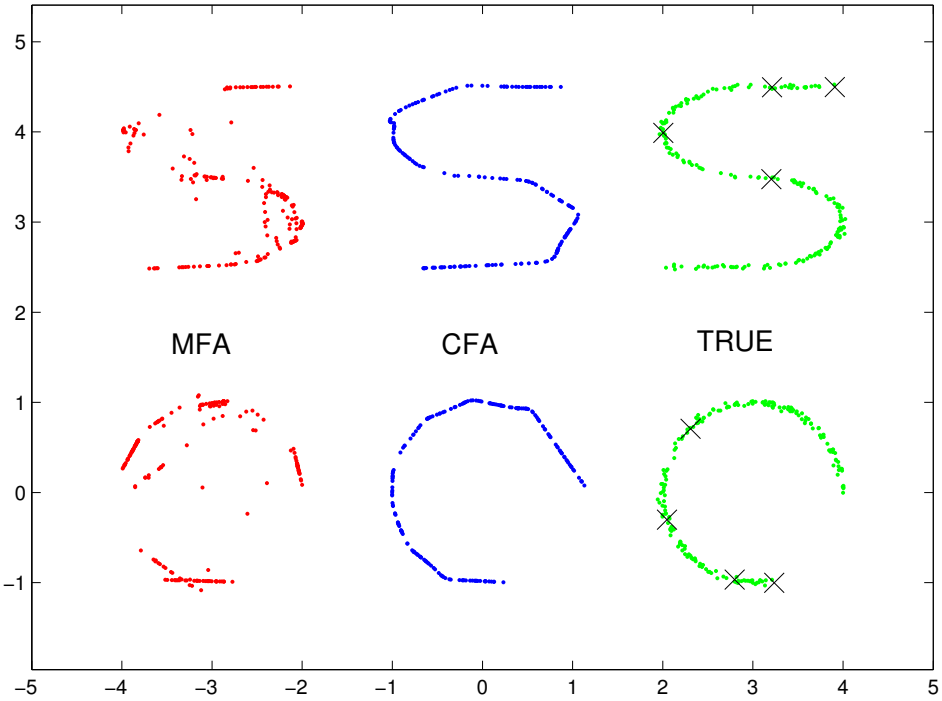


Figure 5.13: Predictions on the test set with MFA and CFA and the true coordinates plotted as dots. The correspondences are plotted with an $\times$ symbol.

Figure 5.14: Examples of the images of the two toy puppets. The top row contains images of the cat figure and the bottom row images of the dwarf figure. Corresponding views are displayed above each other.

data set of much higher dimensionality. The data consists of 2500 gray-scale images of 64×64 pixels each of two toy puppets viewed from different directions. In Fig. 5.14 some corresponding views of the two puppets are depicted. The images, originally used in (Peters et al., 2002), were provided by G. Peters who recorded them at the Institute for Neural Computation of the Ruhr-Universität-Bochum, Germany. Images of the objects were recorded while moving the camera over the *viewing hemisphere*, see the left panel of Fig. 5.15. The viewing hemisphere was sampled at $3.6°$ intervals in longitude (yielding 100 steps to complete the circle) and at $4.0°$ intervals in latitude (yielding 25 steps to go from the equator of the hemisphere to the pole). Note that the images recorded near the top of the hemisphere differ considerably, since these are different rotations of the top view of the object.

There are only two degrees of freedom in each set of images since the images are determined by position of the camera, which is in turn determined by its longitude and latitude on the hemisphere. Since the longitude is a periodic degree of freedom, the images can be embedded on the surface of a cylinder in a Euclidean space. In principle the images can also be embedded, while preserving nearest neighbor relations, in a two dimensional space by embedding images with equal latitude on concentric circles with a radius that is monotonically increasing with the latitude. However, such a two dimensional embedding in not returned by the LLE based algorithm since it is not directly aiming at the preservation of nearest neighbor relations. Three dimensions are required to recover a cylinder-like embedding of the images in which only similar images are embedded nearby. Therefore, we used a three dimensional latent space for the CFA models, and to obtain comparable results we also do this for the MFA models. In Fig. 5.16 the two dimensional and three dimensional latent representation recovered by the LLE based initialization method when using all 2500 images of each set and 125 correspondences is illustrated. The recovered coordinates indeed trace-out a cylinder-like shape in the three dimensional embedding. The two dimensional embedding corre-

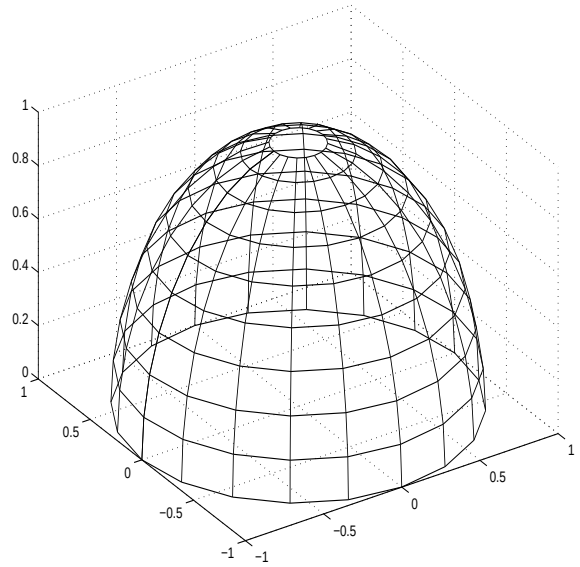


Figure 5.15: The viewing hemisphere; the object was fixed in the center of the sphere and images were recorded when the camera was placed a different locations on the hemisphere (crossings of the lines on the surface) and directed toward the object.

sponds roughly to a projection of the cylinder along is axis.

In order to speed-up experimentation, the images were first projected to the 100 dimensional linear PCA subspace of the $64 \times 64 = 4096$ dimensional space spanned by the pixel values. Over 90% of the original variance in the data was contained in this subspace with approximately 40 times fewer dimensions. Since the discarded dimensions contain only a small fraction of the data variance, projecting the original data to the PCA subspace is expected to have little effect on the obtained results. Below we also show results obtained when using 1000 dimensional PCA projection. We trained CFA and MFA models with different numbers of mixture components $k = 10, 20, \ldots, 100$ and with different percentages of the data for which correspondences were given $c = 1\%, 2\%, 5\%, 10\%, 20\%, 50\%$. Of each object 2000 images were used for training and 500 to asses the reconstruction error E_{rec} from (5.83).

The obtained errors, averaged over six random selections of training and test data, are tabulated in Fig. 5.17. In Fig. 5.18 we plotted averages of the error of CFA models and of MFA models using $k = 40$ components against the percentage of correspondences that was used. It can be observed that already with relatively few correspondences, CFA models make accurate predictions. Many more correspondences are needed to obtain similar errors with MFA models.

To compare MFA and CFA qualitatively, we plotted the predicted correspondence for some of the test examples in Fig. 5.19. For reference, we also included reconstructions of the images obtained from a three dimensional linear PCA projection, which is the op-

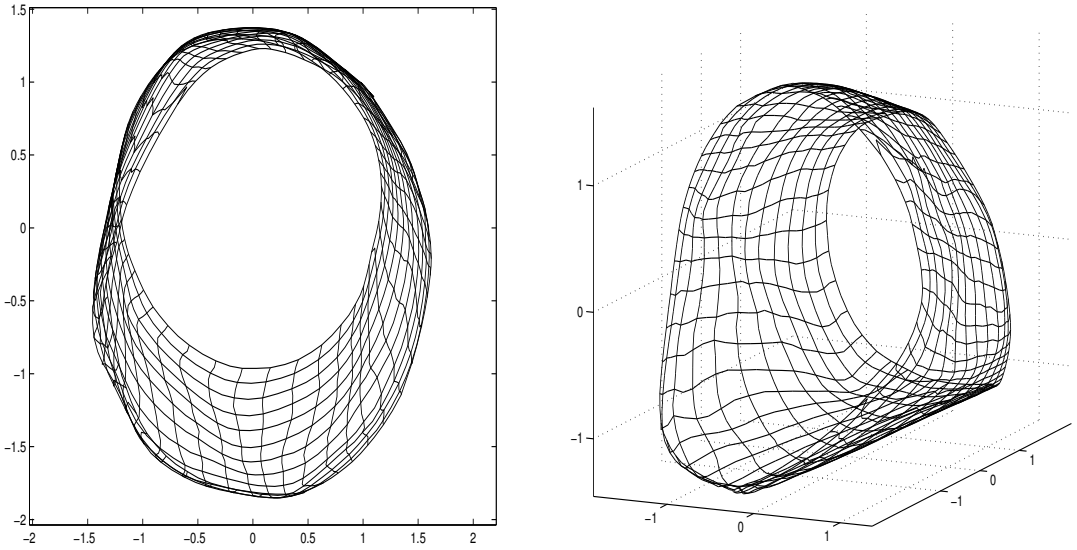


Figure 5.16: Two dimensional (left panel) and three dimensional (right panel) embedding of 625 of the 2500 images. Images recorded from equal latitude and longitude are connected by lines.

timal linear reconstruction of the images from a three dimensional representation. The depicted results were obtained by training models with $k = 65$ mixture components on 2000 images of each object with 100 images in correspondence ($c = 5/100$). Before fitting the models, the images in each set were projected on the first 1000 principal components derived from the images in that set, preserving over 99.6% of the variance in each set. The average errors (per image and per dimension) were 0.127 for CFA and 0.732 for MFA. Clearly, the CFA model —which exploited the manifold structure of the data— yields superior predictions as compared to those obtained with the MFA model.

5.4 Conclusions

In this chapter we considered the CFA approach that combines several linear models to form a non-linear models. The amount of computation required to perform the EM-steps is $O(NDkd + Nd^3))$. Compared to GTM and SOMM, the advantage of this method is that the low dimensional representation is not restricted to a discrete set of points (the nodes in SOMM and GTM) but offers a continuous low dimensional representation. Furthermore, the conditional density $p(\mathbf{g}|\mathbf{x})$ on latent coordinates given a data vector (and vice versa $p(\mathbf{x}|\mathbf{g})$) is Gaussian mixture density whose parameters and expectation are readily computed.

In Section 5.2.3 we presented an improvement in the parameter estimation scheme proposed in (Roweis and Saul, 2000) that replaces an iterative procedure in the M-step of

k	$c = \frac{2}{100}$		$c = \frac{5}{100}$		$c = \frac{10}{100}$		$c = \frac{20}{100}$		$c = \frac{50}{100}$	
1	**4.87**	**6.47**	**4.64**	**6.47**	**4.21**	**6.42**	**4.04**	**6.42**	**3.89**	**6.32**
10	**2.31**	**8.16**	**1.99**	**5.25**	**1.78**	**3.74**	**1.54**	**2.51**	**1.47**	**1.74**
20	**1.44**	**7.21**	**1.18**	**4.89**	**1.08**	**4.29**	**0.93**	**2.55**	0.84	1.02
30	**1.15**	**7.14**	**0.96**	**4.71**	**0.84**	**3.35**	**0.73**	**2.40**	**0.64**	**1.07**
40	**1.51**	**7.06**	**0.74**	**4.20**	**0.68**	**2.71**	**0.66**	**2.26**	**0.64**	**0.86**
50	**2.32**	**7.94**	**0.73**	**4.30**	**0.54**	**3.27**	**0.50**	**2.34**	**0.44**	**0.72**
60	**1.49**	**7.36**	**0.55**	**4.55**	**0.59**	**2.95**	**0.46**	**1.78**	**0.36**	**0.89**
70	16.59	7.84	**0.58**	**5.75**	**0.45**	**3.70**	**0.51**	**1.98**	**0.38**	**0.94**

Figure 5.17: CFA and MFA errors for the image data set divided over the number of test points (500) and the number of dimensions (100).

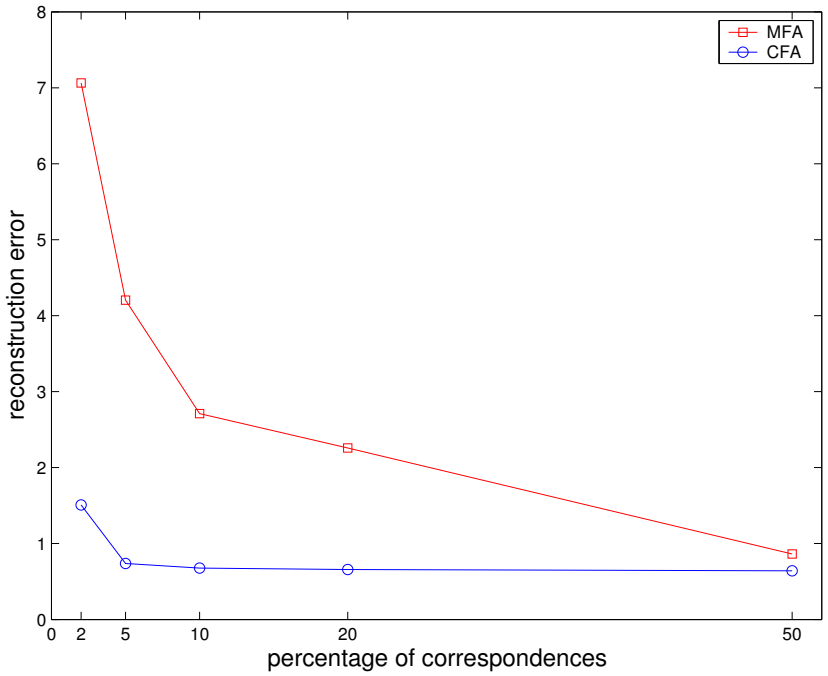


Figure 5.18: The reconstruction errors obtained with CFA and MFA models using $k = 40$ components, averaged six random train and test sets.

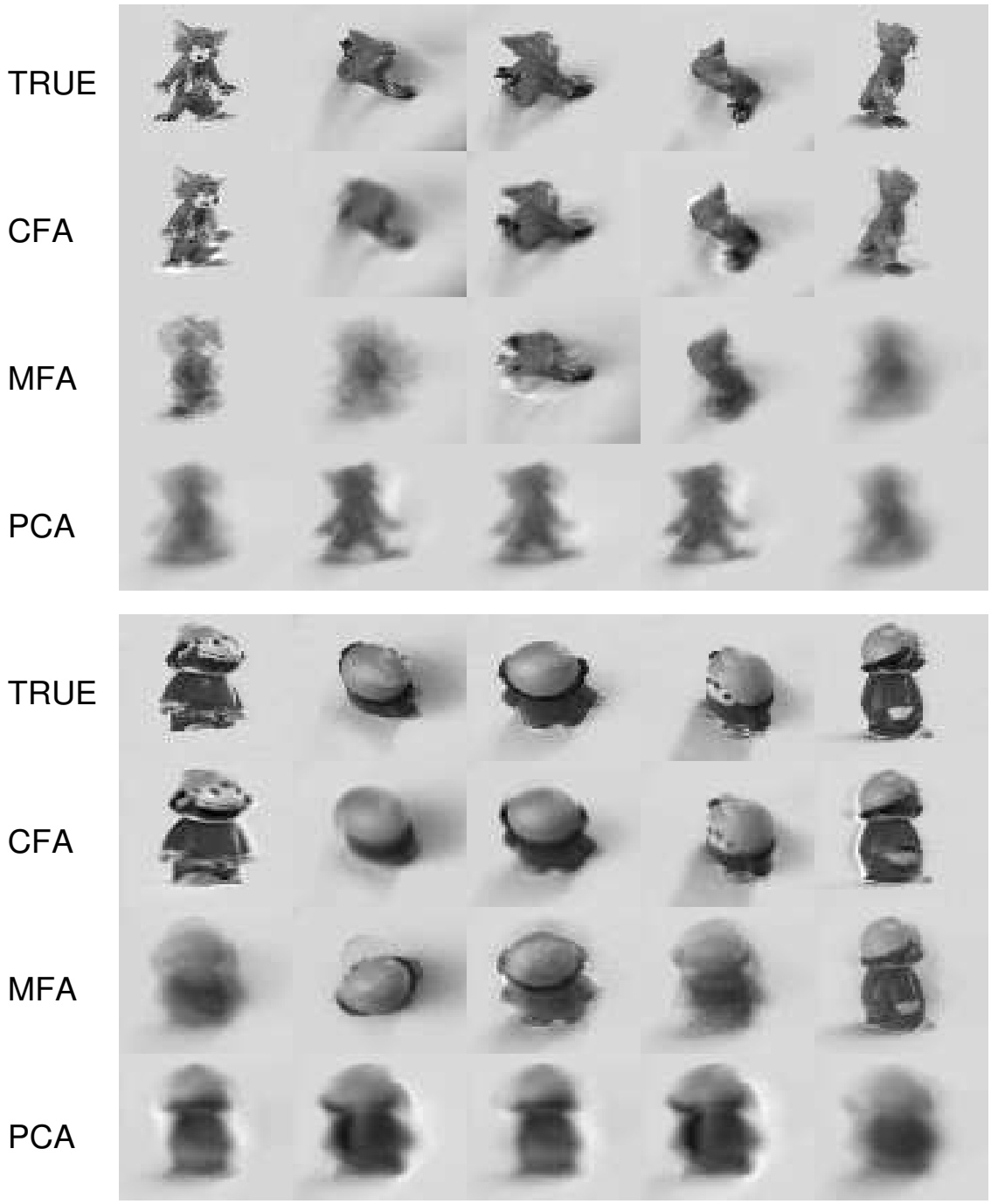


Figure 5.19: Top panel shows (predicted) corresponding view of the cat given a view of the dwarf and bottom panel shows (predicted) corresponding view of the dwarf given a view of the cat. Within each panel, from top top bottom: true correspondence, reconstruction with CFA, reconstruction with MFA and 3D PCA projection.

the EM-like algorithm with a closed-form solution. The closed-form update leads to a considerable speed-up in the algorithm and improved parameter estimates. The experimental results presented in Section 5.2.5 indicated that the CFA approach compares favorably to SOMM and GTM in the sense that fewer mixture components are needed to obtain similar reconstruction errors (errors in predicting the high dimensional points from their low dimensional representation).

In Section 5.3 we applied the CFA approach to a setting where the goal is to find a model that enables us to predict between two high-dimensional variables. In this setting, the training data consists of two sets of high dimensional observations each in a different space. Only for some of the observations the corresponding point in the other space is given. We compared the CFA approach with a mixture of factor analyzers approach, and experimentally found that when only a few correspondences are given the CFA approach leads to significantly better prediction accuracy, both for synthetic and natural data. The difference in performance is explained by the fact that CFA successfully uses the manifold structure of the data to determine the dependencies between the high dimensional variables, even if very few correspondences are given. It is straightforward to generalize this approach to settings where more than two sets of points are given.

6

CONCLUSION AND DISCUSSION

Clustering and dimension reduction methods can be used to find a more compact representation of high dimensional data. In this compact representation, the data are described on the basis of cluster membership and/or by a low dimensional vector obtained by projecting the data on a low dimensional manifold. These methods are useful tools for data visualization, and pre-processing data in classification or regression applications in order to avoid the curse of dimensionality.

In this thesis we studied probabilistic mixture models, which provide a versatile framework for clustering and dimension reduction. By using different component distributions, the framework can be applied to many different types of data. Furthermore, parameter estimation, missing values in the data, and model selection can all be treated within a single and formally sound framework. We presented several contributions which we hope increase the practical applicability of mixture models for clustering and dimension reduction problems. Below we summarize the conclusions drawn in the previous chapters and outline directions for further research.

6.1 Summary of conclusions

Many, if not all, mixture model based clustering and dimension reduction techniques suffer from the problem that the objective function that is optimized exhibits several local optima. Standard parameter estimation techniques are hindered by these local optima, since they may return parameter estimates which are locally optimal but far from globally optimal. In Section 3.2 we presented a greedy parameter estimation scheme, that iteratively (i) adds components to the mixture and (ii) re-estimates the parameters of the mixture obtained so far with the EM algorithm. In Section 3.3 we presented a similar greedy scheme for k-means clustering. Our experimental results, obtained using synthetic and natural data, show that the greedy approach yields equal or better clusterings (in terms of the objective function) than alternative techniques. An additional benefit of the greedy approach is that a sequence of mixture models with an increasing

number of components is generated, which is useful when the number of clusters has to be determined as well.

In Section 3.4 we presented a constrained EM algorithm to accelerate parameter estimation for Gaussian mixtures from large data sets. Other existing acceleration techniques allow a limited freedom in setting a trade-off between accuracy and speed-up of each EM step. For very large speed-up and small accuracy the algorithms are not guaranteed to converge to (in a particular sense) locally optimal parameter estimates. In contrast, our algorithm converges to (locally) optimal parameter values for any trade-off between accuracy and speed, and also allows other speed-up techniques (e.g. based on geometric reasoning) to be plugged-in.

In Chapter 4 we presented a self-organizing map approach based on mixture models; the parameters are estimated by a constrained EM algorithm. The advantage of our mixture model based approach over Kohonen's original self-organizing map is that the learning algorithm is guaranteed to converge and can be interpreted as maximizing an objective function. Moreover, the objective function can be augmented such that data with missing values can also be used for parameter estimation. Since self-organization is achieved by a simple modification of the standard EM algorithm for mixture models, our approach is readily applied to any mixture model for which the standard EM algorithm is available. A priori domain knowledge can be used to select an appropriate class of component densities. Therefore, it is relatively easy to apply this method to data which is not given as a set of vectors of real numbers.

In Chapter 5 we considered the coordinated factor analysis (CFA) model which uses a mixture of linear Gaussian latent variable models for non-linear dimension reduction. We presented an improvement of the original parameter estimation algorithm that is faster and leads to more accurate parameter estimates. We experimentally compared the performance of this method with generative topographic mapping and the self-organizing map approach of Chapter 4. The experimental results show that the CFA model needs about half the number of mixture components needed by the other approaches to reconstruct the data from the latent representation with a given accuracy. This is due to the fact that the CFA approach use a continuous latent representation, where the other approaches use a discrete latent representation.

In Section 5.3 we applied the CFA model to predict high dimensional correspondences. In this setting two data sets are given, each sampled from a different high dimensional embedding of the same low dimensional manifold. In addition, some correspondences are given: for some data points in the first set it is known that they share the same low dimensional coordinate on the manifold as a point in the second set. The goal is to predict, for points without a given correspondence, the coordinates of the corresponding point in the other set. This problem can be regarded as a missing value problem, which can be solved without constructing a global low dimensional latent representation, e.g. using a mixture of factor analyzers (MFA) which uses several local linear low dimensional representations to predict the correspondences. With our experiments we

demonstrated that if only few correspondences are given, then the CFA models yield more accurate predictions than the MFA models. This is explained by the fact that the MFA models lack a global low dimensional representation as used by the CFA models.

6.2 Discussion of directions for further research

Combining spectral methods and parametric models. A drawback of mixture models is the fact that parameter estimation techniques may return estimates that are locally, but not globally optimal. In principle, sampling based global optimization methods can be used, like simulated annealing, but these require an impractically long sequence of samples. In Section 3.2 and Section 3.3 we presented algorithms that avoid the local optima better than alternative algorithms, but in general there is no efficient algorithm available which is *guaranteed* to identify the globally optimal parameters.

In recent years there has been considerable interest in spectral methods for clustering and dimension reduction. Spectral methods deliver a clustering or low dimensional representation of the given data, but not a parametric model that can be used to map new data to a cluster index or low dimensional coordinate. The main attraction of these methods is that they minimize a quadratic function with a single global minimum that can be efficiently identified. Most of these methods involve a small number of parameters which have to be set, such as the number of neighbors k in nearest neighbor based methods. Since they are few in number, global optimization of these parameters is easier than finding a parametric model with many parameters of the high dimensional data.

To generalize the clustering or low dimensional representation produced by spectral methods to new data, all training data has to be retained and often nearest neighbors of the new data have to be found in the original data. Both storing the original data and finding nearest neighbors is undesirable when dealing with large data sets. In Chapter 5 we initialized the estimates of hidden variables in the CFA model with the solutions of spectral methods. Further research is needed to explore further possibilities to combine the benefits of spectral methods and methods that deliver a parametric model that can be applied to new data. For example, spectral methods could be used to define a distribution over plausible clusterings or latent coordinates. Parametric model learning can then be biased toward such plausible solutions rather than just be initialized by the solution of the spectral method.

Robust dimension reduction. In Chapter 5 we used mixtures of linear Gaussian latent variable models for non-linear dimension reduction. Gaussian densities are attractive from a computational point of view, and are also preferred as a default density in the absence of prior knowledge which suggests another, more appropriate, class of densities. However, the short tails of the Gaussian density make' parameter estimation relatively

sensitive to outliers in the data. Outliers are data points which are very different from all other data points and that would be very unlikely under a distribution with parameters estimated from the other data points. Outliers can be caused by noise in measurement systems. Mixtures of linear latent variable models based on t-distributions have been proposed (de Ridder and Franc, 2003) to obtain more robustness against outliers. It would be interesting to consider how such mixtures of t-distributions could be used to form global non-linear models as in the CFA model.

Semi-supervised learning. As discussed in the introduction, unsupervised clustering and dimension reduction techniques can be used to find compact data representations to overcome the curse of dimensionality in classification and regression problems. Data processing then proceeds in two steps. First, a suitable compact data representation is determined using unsupervised examples. Second, the supervised data is analyzed in the new representation to find an appropriate classification or regression function. Recently *semi-supervised* learning[1] approaches have been introduced that integrate these two steps by directly learning a classification or regression function in the high dimensional space from both supervised and unsupervised data. Loosely speaking, these methods avoid the curse of dimensionality by using unsupervised data to reduce the set of possible functions. The power of the semi-supervised approach lies in the fact that *both* the supervised and unsupervised data are used to reduce the set of functions.

Various semi-supervised learning approaches have been proposed (Baluja, 1998; Blum and Mitchell, 1998; Nigam et al., 2000; Szummer and Jaakkola, 2002; Zhu et al., 2003). Some of these use nearest neighbor graphs, to encode the smoothness assumption the class label (or regression variable) tends to be the same or similar for nearby points in the high dimensional data space. These methods can be implemented efficiently, have few parameters that have to be estimated and perform very well in practice. However, to evaluate the predictive density all training data needs to be accessed and thus stored. Nearest neighbor based methods can also be used to define a *distribution* over the class labels of both the supervised and unsupervised data. By conditioning on the known class labels of the supervised data we obtain a distribution over class labels of the unsupervised data. It is an interesting possibility to define such a conditional distribution on the class labels of the unsupervised data, and then to learn a parametric model from supervised and unsupervised data that —in expectation with respect to the uncertain class labels of the unsupervised data— optimally predicts the class labels.

[1] Unfortunately, the term semi-supervised learning is sometimes used to refer to reinforcement learning, e.g. in (Arbib, 1995), which is quite different from the setup considered here.

BIBLIOGRAPHY

Alsabti, K., Ranka, S., and Singh, V. (1998). An efficient k-means clustering algorithm. In *Proceedings of the First Workshop High Performance Data Mining*. Pages: 75

Andrieu, C., de Freitas, N., Doucet, A., and Jordan, M. I. (2003). An introduction to MCMC for machine learning. *Machine Learning*, 50(1):5–43. Pages: 48

Anouar, F., Badran, F., and Thiria, S. (1998). Probabilistic self-organizing map and radial basis function networks. *Neurocomputing*, 20(1-3):83–96. Pages: 98

Arbib, M. A., editor (1995). *The handbook of brain theory and neural networks*. MIT Press, Cambridge, MA, USA. Pages: 148

Bach, F. R. and Jordan, M. I. (2002). Kernel independent component analysis. *Journal of Machine Learning Research*, 3:1–48. Pages: 23

Bach, F. R. and Jordan, M. I. (2004). Learning spectral clustering. In S. Thrun and L. K. Saul and B. Schölkopf, editor, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, Cambridge, MA, USA. Pages: 17

Balasubramanian, M., Schwartz, E. L., Tenenbaum, J. B., de Silva, V., and Langford, J. C. (2002). The isomap algorithm and topological stability. *Science*, 295(5552):7a. Pages: 35, 38

Baldi, P. and Hornik, K. (1989). Neural networks and principal component analysis: learning from examples without local minima. *Neural Networks*, 2(1):53–58. Pages: 27

Baluja, S. (1998). Probabilistic modeling for face orientation discrimination: learning from labeled and unlabeled data. In M. J. Kearns S. A. Solla and D. A. Cohn, editor, *Advances in Neural Information Processing Systems*, volume 11. MIT Press, Cambridge, MA, USA. Pages: 148

Banddyopadhyay, S., Maulik, U., and Pakhira, M. K. (2001). Clustering using simulated annealing with probabilistic redistribution. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(2):269–285. Pages: 67

Beal, M. and Ghahramani, Z. (2003). The variational Bayesian EM algorithm for incomplete data: with application to scoring graphical model structures. In Bernardo, J., Bayarri, M., Berger, J., Dawid, A., Heckerman, D., Smith, A., and West, M., editors, *Bayesian Statistics*, volume 7, New-York, NY, USA. Oxford University Press. Pages: 48, 61

Belkin, M. and Niyogi, P. (2002). Laplacian eigenmaps and spectral techniques for embedding and clustering. In T. G. Dietterich and S. Becker and Z. Ghahramani, editor,

Advances in Neural Information Processing Systems, volume 14, pages 585–591. MIT Press, Cambridge, MA, USA. Pages: 36, 107

Belkin, M. and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396. Pages: 36

Bellman, R. (1961). *Adaptive control processes: a guided tour*. Princeton University Press. Pages: 3

Bengio, Y., Paiement, J., Vincent, P., Delalleau, O., Roux, N. L., and Ouimet, M. (2004). Out-of-sample extensions for LLE, isomap, MDS, eigenmaps, and spectral clustering. In S. Thrun and L. K. Saul and B. Schölkopf, editor, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, Cambridge, MA, USA. Pages: 107

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517. Pages: 69, 78

Bentley, J. L. (1980). Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229. Pages: 16

Bernstein, M., de Silva, V., Langford, J., and Tenenbaum, J. (2000). Graph approximations to geodesics on embedded manifolds. Technical report, Department of Psychology, Stanford University. Pages: 34

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford University Press, New-York, NY, USA. Pages: 17, 21, 26, 52

Bishop, C. M., Svensén, M., and Williams, C. K. I. (1998a). Developments of the generative topographic mapping. *Neurocomputing*, 21:203–224. Pages: 97

Bishop, C. M., Svensén, M., and Williams, C. K. I. (1998b). GTM: the generative topographic mapping. *Neural Computation*, 10:215–234. Pages: 28, 97

Bishop, C. M. and Tipping, M. E. (1998). A hierarchical latent variable model for data visualization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):281–293. Pages: 7

Blake, C. and Merz, C. (1998). UCI repository of machine learning databases. Pages: 69, 102

Blei, D., Griffiths, T. L., Jordan, M. I., and Tenenbaum, J. B. (2004). Hierarchical topic models and the nested Chinese restaurant process. In S. Thrun and L. K. Saul and B. Schölkopf, editor, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, Cambridge, MA, USA. Pages: 8

Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In Bartlett, P. L. and Mansour, Y., editors, *Proceedings of the Annual Conference on Computational Learning Theory*, volume 11, pages 92–100. ACM Press, New-York, NY, USA. Pages: 148

Böhning, D. (1995). A review of reliable maximum likelihood algorithms for semiparametric mixture models. *Journal of Statistical Planning and Inference*, 47(1-2):5–28. Pages: 59

Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59:291–294. Pages: 27

Brand, M. (1999). Structure learning in conditional probability models via an entropic

prior and parameter extinction. *Neural Computation*, 11(5):1155–1182. Pages: 60

Brand, M. (2003). Charting a manifold. In S. Becker and S. Thrun and K. Obermayer, editor, *Advances in Neural Information Processing Systems*, volume 15, pages 961–968. MIT Press, Cambridge, MA, USA. Pages: 37, 38, 119

Brand, M. (2004). Minimax embeddings. In S. Thrun and L. K. Saul and B. Schölkopf, editor, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, Cambridge, MA, USA. Pages: 37

Brand, M. and Huang, K. (2003). A unifying theorem for spectral embedding and clustering. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, volume 9. Pages: 15

Brassard, G. and Bratley, P. (1996). *Fundamentals of algorithmics*. Prentice Hall. Pages: 34

Carreira-Perpiñán, M. Á. (1997). A review of dimension reduction techniques. Technical Report CS-96-09, Dept. of Computer Science, University of Sheffield. Pages: 40, 127

Celeux, G., Forbes, F., and Payrard, N. (2003). EM procedures using mean field-like approximations for Markov model-based image segmentation. *Pattern Recognition*, 36:131–144. Pages: 47

Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799. Pages: 11

Chung, F. (1997). *Spectral graph theory*. Number 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society. Pages: 13

Cover, T. and Thomas, J. (1991). *Elements of Information Theory*. Wiley, New-York, NY, USA. Pages: 44, 65

Cox, T. and Cox, M. (1994). *Multidimensional scaling*. Number 59 in Monographs on statistics and applied probability. Chapman & Hall. Pages: 30, 31

Dasgupta, S. (1999). Learning mixtures of Gaussians. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, volume 40, pages 634–644. IEEE Computer Society Press, Los Alamitos, CA, USA. Pages: 62

de Ridder, D. (2001). *Adaptive methods of image processing*. PhD thesis, Delft University of Technology, Delft, The Netherlands. Pages: 51

de Ridder, D. and Duin, R. P. W. (1997). Sammon's mapping using neural networks: a comparison. *Pattern Recognition Letters*, 18(11-13):1307–1316. Pages: 32

de Ridder, D. and Duin, R. P. W. (2002). Locally linear embedding for classification. Technical Report PH-2002-01, Pattern Recognition Group, Dept. of Imaging Science and Technology, Delft University of Technology. Pages: 38

de Ridder, D. and Franc, V. (2003). Robust subspace mixture models using t-distributions. In Harvey, R. and Bangham, A., editors, *Proceedings of the 2003 British Machine Vision Conference*, pages 319–328. British Machine Vision Association, Norwich, UK. Pages: 148

Delicado, P. and Huerta, M. (2003). Principal curves of oriented points: theoretical and computational improvements. *Computational Statistics*, 18(2):293–315. Pages: 25

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series*

B (Methodological), 39(1):1–38. Pages: 12, 44, 89

DerSimonian, R. (1986). Maximum likelihood estimation of a mixing distribution. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 35:302–309. Pages: 59

Dhillon, I. S., Mallela, S., and Kumar, R. (2002). Enhanced word clustering for hierarchical text classification. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 8, pages 191–200. ACM Press, New-York, NY, USA. Pages: 12

Donoho, D. L. and Grimes, C. (2003). Hessian eigenmaps: locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences of the USA*, 100(10):5591–5596. Pages: 36

Erwin, E., Obermayer, K., and Schulten, K. J. (1992). Self-organizing maps: ordering, convergence properties and energy functions. *Biological Cybernetics*, 67(1):47–55. Pages: 88, 95

Faloutsos, C. and Lin, K.-I. (1995). FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the ACM SIGMOD international conference on Management of data*, pages 163–174. ACM Press, New-York, NY, USA. Pages: 22

Figueiredo, M. A. T. and Jain, A. K. (2002). Unsupervised learning of finite mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):381–396. Pages: 60

Fodor, I. K. (2002). A survey of dimension reduction techniques. Technical Report UCRL-ID-148494, Lawrence Livermore National Laboratory, Center for Applied Scientific Computing. Pages: 40

Fred, A. and Jain, A. K. (2002). Data clustering using evidence accumulation. In Kasturi R., Laurendeau D., S. C., editor, *Proceedings of the International Conference on Pattern Recognition*, volume 16, pages 276–280. IEEE Computer Society Press, Los Alamitos, CA, USA. Pages: 11

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139. Pages: 19

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):712–741. Pages: 61

Gersho, A. and Gray, R. M. (1992). *Vector quantization and signal compression*, volume 159 of *Series in Engineering and Computer Science*. Kluwer, Boston, MA, USA. Pages: 11

Ghahramani, Z. and Beal, M. J. (2000). Variational inference for Bayesian mixtures of factor analysers. In S. A. Solla and T. K. Leen and K.-R. Müller, editor, *Advances in Neural Information Processing Systems*, volume 12, pages 449–455. MIT Press, Cambridge, MA, USA. Pages: 61

Ghahramani, Z. and Hinton, G. E. (1996). The EM algorithm for mixtures of factor analyzers . Technical Report CRG-TR-96-1, University of Toronto, Canada. Pages:

50, 134

Ghahramani, Z. and Jordan, M. I. (1994). Supervised learning from incomplete data via an EM approach. In J. D. Cowan and G. Tesauro and J. Alspector, editor, *Advances in Neural Information Processing Systems*, volume 6, pages 120–127. Morgan Kaufmann, San Mateo, CA, USA. Pages: 92, 129

Girolami, M. (2001). The topographic organisation and visualisation of binary data using mutivariate-Bernoulli latent variable models. *IEEE Transactions on Neural Networks*, 12(6):1367–1374. Pages: 97

Golub, G. H. and Van Loan, C. F. (1996). *Matrix computations*. Johns Hopkins University Press. Pages: 22

Graepel, T., Burger, M., and Obermayer, K. (1998). Self-organizing maps: generalizations and new optimization techniques. *Neurocomputing*, 21(1-3):173–190. Pages: 95

Green, P. (1995). Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82:711–732. Pages: 61

Ham, J. H., Lee, D. D., and Saul, L. K. (2003). Learning high dimensional correspondences from low dimensional manifolds. In *Proceedings of the ICML workshop on the continuum from labeled to unlabeled data in machine learning and data mining*, pages 34–41. Pages: 126, 132

Hansen, P., Ngai, E., Cheung, B. K., and Mladenović, N. (2002). Analysis of global k-means, an incremental heuristic for minimum sum-of-squares clustering. Technical report, HEC Montreal, Group for Research in Decision Analysis. Pages: 68

Hastie, T., Friedman, J., and Tibshirani, R. (2001). *The elements of statistical learning*. Springer Series in Statistics. Spinger-Verlag, New-York, NY, USA. Pages: 17, 31, 40

Hastie, T. and Stuetzle, W. (1989). Principal curves. *Journal of the American Statistical Association*, 84(406):502–516. Pages: 24, 25

He, X. and Niyogi, P. (2004). Locality preserving projections. In S. Thrun and L. K. Saul and B. Schölkopf, editor, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, Cambridge, MA, USA. Pages: 37

Heskes, T. (2001). Self-organizing maps, vector quantization, and mixture modeling. *IEEE Transactions on Neural Networks*, 12:1299–1305. Pages: 95, 96

Hinton, G. E., Dayan, P., and Revow, M. (1997). Modelling the manifolds of images of handwritten digits. *IEEE Transactions on Neural Networks*, 8(1):65–74. Pages: 108

Hinton, G. E. and Roweis, S. T. (2003). Stochastic neighbor embedding. In S. Becker and S. Thrun and K. Obermayer, editor, *Advances in Neural Information Processing Systems*, volume 15, pages 833–840. MIT Press, Cambridge, MA, USA. Pages: 32

Horn, R. A. and Johnson, C. R. (1985). *Matrix analysis*. Cambridge University Press, Cambridge, UK. Pages: 16, 30, 115

Jacobs, R., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3:79–87. Pages: 112

Jain, A. K. and Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice-Hall, Inc. Pages: 17

Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: a review. *ACM*

Computing Surveys, 31(3):264–323. Pages: 7, 17, 67

Johnson, S. C. (1967). Hierarchical clustering schemes. *Psychometrika*, 32:241–254. Pages: 8

Jolliffe, I. T. (1986). *Principal component analysis*. Springer Series in Statistics. Spinger-Verlag, New-York, NY, USA. Pages: 21, 50

Kaban, A. and Girolami, M. (2001). A combined latent class and trait model for the analysis and visualization of discrete data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:859–872. Pages: 97

Kambhatla, N. and Leen, T. K. (1994). Fast non-linear dimension reduction. In J. D. Cowan and G. Tesauro and J. Alspector, editor, *Advances in Neural Information Processing Systems*, volume 6. Morgan Kaufmann, San Mateo, CA, USA. Pages: 51, 108

Kanungo, T., Mount, D. M., Netanyahu, N., Piatko, C., Silverman, R., and Wu, A. Y. (2002). An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:881–892. Pages: 75

Karger, D. R. and Ruhl, M. (2002). Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 741–750. ACM Press, New-York, NY, USA. Pages: 16

Kégl, B. (2003). Intrinsic dimension estimation using packing numbers. In S. Becker and S. Thrun and K. Obermayer, editor, *Advances in Neural Information Processing Systems*, volume 15, pages 681–688. MIT Press, Cambridge, MA, USA. Pages: 38

Kégl, B., Krzyzak, A., Linder, T., and Zeger, K. (2000). Learning and design of principal curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(3):281–297. Pages: 25

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680. Pages: 61

Kohonen, T. (2001). *Self-Organizing Maps*. Springer Series in Information Sciences. Spinger-Verlag, New-York, NY, USA. Pages: 27, 85, 88

Kohonen, T., Kaski, S., and Lappalainen, H. (1997). Self-organized formation of various invariant-faeture filters in the adaptive-subspace SOM. *Neural Computation*, 9(6):1321–1344. Pages: 93

Kohonen, T. and Somervuo, P. (2002). How to make large self-organizing maps for nonvectorial data. *Neural Networks*, 15(8-9):945–952. Pages: 87

Kostiainen, T. and Lampinen, J. (2002). On the generative probability density model in the Self-Organizing Map. *Neurocomputing*, 48(1-4):217–228. Pages: 98

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *Journal of the American Institute of Chemical Engineers*, 37(2):233–243. Pages: 27

Krishna, K. and Murty, M. (1999). Genetic k-means algorithm. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 29(3):433–439. Pages: 67

Laaksonen, J., Koskela, K., Laakso, S., and Oja, E. (2001). Self-organizing maps as a relevance feedback technique in content-based image retrieval. *Pattern Analysis*

and Applications, 4(2-3):140–152. Pages: 86

Lee, J. M. (2003). *Introduction to smooth manifolds*, volume 218 of *Graduate Texts in Mathematics*. Spinger-Verlag, New-York, NY, USA. Pages: 38

Leung, Y., Zhang, J.-S., and Xu, Z.-B. (2000). Clustering by scale-space filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1396–1410. Pages: 11

Li, J. Q. (1999). *Estimation of Mixture Models*. PhD thesis, Department of Statistics, Yale University. Pages: 56

Li, J. Q. and Barron, A. R. (2000). Mixture density estimation. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, Cambridge, MA, USA. Pages: 53, 55, 56

Likas, A., Vlassis, N., and Verbeek, J. J. (2003). The global k-means clustering algorithm. *Pattern Recognition*, 36(2). Pages: 66

Lindsay, B. G. (1983). The geometry of mixture likelihoods: a general theory. *The Annals of Statistics*, 11(1):86–94. Pages: 49, 59

McLachlan, G. J. and Peel, D. (2000). *Finite Mixture Models*. Wiley, New-York, NY, USA. Pages: 42

Meilă, M. and Shi, J. (2001). A random walks view of spectral segmentation. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, volume 8. Pages: 13

Mika, S., Rätsch, G., and Müller, K.-R. (2001). A mathematical programming approach to the kernel fisher algorithm. In T. K. Leen and T. G. Dietterich and V. Tresp, editor, *Advances in Neural Information Processing Systems*, volume 13, pages 591–597. MIT Press, Cambridge, MA, USA. Pages: 23

Milligan, G. W. and Cooper, M. C. (1985). An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50:159–179. Pages: 67

Minka, T. P. (2001). Automatic choice of dimensionality for PCA. In T. K. Leen and T. G. Dietterich and V. Tresp, editor, *Advances in Neural Information Processing Systems*, volume 13, pages 598–604. MIT Press, Cambridge, MA, USA. Pages: 22

Moore, A. W. (1999). Very fast EM-based mixture model clustering using multiresolution kd-trees. In M. J. Kearns S. A. Solla and D. A. Cohn, editor, *Advances in Neural Information Processing Systems*, volume 11, pages 543–549. MIT Press, Cambridge, MA, USA. Pages: 75, 78, 79, 83

Moore, A. W. and Pelleg, D. (1999). Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 5, pages 277–281. Pages: 75

Neal, R. M. and Hinton, G. E. (1998). A view of the EM algorithm that justifies incremental, sparse, and other variants. In Jordan, M., editor, *Learning in Graphical Models*, volume 89 of *NATO Science Series D: Behavioural and Social Sciences*, pages 355–368. Kluwer, Boston, MA, USA. Pages: 13, 44, 89

Negri, S. and Belanche, L. (2001). Heterogeneous Kohonen networks. In *Connectionist Models of Neurons, Learning Processes and Artificial Intelligence : 6th International Work-Conference on Artificial and Natural Neural Networks*, pages 243–252,

New-York, NY, USA. Spinger-Verlag. Pages: 102

Ng, A. Y., Jordan, M. I., and Weiss, Y. (2002). Spectral clustering: analysis and an algorithm. In T. G. Dietterich and S. Becker and Z. Ghahramani, editor, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, Cambridge, MA, USA. Pages: 13, 15

Nigam, K., Mccallum, A., Thrun, S., and Mitchell, T. (2000). Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134. Pages: 148

Oja, E. (1982). A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15:267–273. Pages: 22

Oja, E. (1991). Data compression, feature extraction, and autoassociation in feedforward neural networks. In Kohonen, T., Mäkisara, M., Simula, O., and Kangas, J., editors, *Proceedings of the International Conference on Artificial Neural Networks*, volume 1, pages 737–745, Amsterdam, The Netherlands. North-Holland. Pages: 27

Parzen, E. (1962). On the estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33:1064–1076. Pages: 43

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, 6(2):559–572. Pages: 21

Pelleg, D. and Moore, A. W. (2000). X-means: extending k-means with efficient estimation of the number of clusters. In *Proceedings of the International Conference on Machine Learning*, volume 17, pages 727–734. Morgan Kaufmann, San Mateo, CA, USA. Pages: 11

Pena, J. M., Lozano, J. A., and Larranaga, P. (1999). An empirical comparison of four initialization methods for the k-means algorithm. *Pattern Recognition Letters*, 20:1027–1040. Pages: 66

Peters, G., Zitova, B., and von der Malsburg, C. (2002). How to measure the pose robustness of object views. *Image and Vision Computing*, 20(4):249–256. Pages: 138

Porta, J. M., Verbeek, J. J., and Kröse, B. J. A. (2004). Active appearance-based robot localization using stereo vision. *Autonomous Robots*. to appear. Pages: 19

Rasmussen, C. E. (2000). The infinite Gaussian mixture model. In S. A. Solla and T. K. Leen and K.-R. Müller, editor, *Advances in Neural Information Processing Systems*, volume 12, pages 554–560. MIT Press, Cambridge, MA, USA. Pages: 11

Richardson, S. and Green, P. J. (1997). On Bayesian analysis of mixtures with an unknown number of components. *Journal of the Royal Statistical Society. Series B (Methodological)*, 59(4):731–792. Pages: 61

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, UK. Pages: 17, 27, 69

Rissanen, J. (1989). *Stochastic complexity in statistical inquiry*. World Scientific, Singapore. Pages: 47

Ritter, H. (1993). Parametrized self-organizing maps. In Gielen, S. and Kappen, B., editors, *Proceedings of the International Conference on Artificial Neural Networks*, volume 3, pages 568–577. Spinger-Verlag, New-York, NY, USA. Pages: 128

Rockafellar, R. T. (1993). Lagrange multipliers and optimality. *SIAM Review*, 35(2):183–238. Pages: 39

Rose, K. (1998). Deterministic annealing for clustering, compression, classification, regression and related optimization problems. *IEEE Transactions on Information Theory*, 86(11):2210–2239. Pages: 61, 96

Rosipal, R. and Trejo, L. J. (2001). Kernel partial least squares regression in reproducing kernel Hilbert space. *Journal of Machine Learning Research*, 2:97–123. Pages: 23

Roweis, S. T. (1998). EM Algorithms for PCA and SPCA. In M. I. Jordan and M. J. Kearns and S. A. Solla, editor, *Advances in Neural Information Processing Systems*, volume 10, pages 626–632. MIT Press, Cambridge, MA, USA. Pages: 22, 50, 52

Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326. Pages: 35, 107, 140

Roweis, S. T., Saul, L. K., and Hinton, G. E. (2002). Global coordination of local linear models. In T. G. Dietterich and S. Becker and Z. Ghahramani, editor, *Advances in Neural Information Processing Systems*, volume 14, pages 889–896. MIT Press, Cambridge, MA, USA. Pages: 114, 115, 117, 118, 124

Rubner, Y., Tomasi, C., and Guibas, L. J. (1998). A metric for distributions with applications to image databases. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 6, pages 59–66, Bombay, India. Pages: 19

Sammon, J. W. (1969). A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409. Pages: 32

Saul, L. K. and Pereira, F. (1997). Aggregate and mixed-order Markov models for statistical language processing. In Cardie, C. and Weischedel, R., editors, *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 81–89. Association for Computational Linguistics, Somerset, NJ, USA. Pages: 12

Saul, L. K. and Roweis, S. T. (2003). Think globally, fit locally: unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4:119–155. Pages: 38

Schölkopf, B., Smola, A., and Müller, K. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319. Pages: 22, 23, 107

Schölkopf, B. and Smola, A. J. (2002). *Learning with kernels*. MIT Press, Cambridge, MA, USA. Pages: 23

Schönemann, P. H. (1985). On the formal differentiation of traces and determinants. *Multivariate Behavioral Research*, 20:113–139. Pages: 114

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464. Pages: 48

Scott, G. L. and Longuet-Higgins, H. C. (1990). Feature grouping by 'relocalisation' of eigenvectors of the proximity matrix. In *Proceedings of the 1990 British Machine Vision Conference*, pages 103–108. British Machine Vision Association, Norwich, UK. Pages: 13

Seeger, M. (2000). Bayesian model selection for support vector machines, Gaussian processes and other kernel classifiers. In S. A. Solla and T. K. Leen and K.-R. Müller,

editor, *Advances in Neural Information Processing Systems*, volume 12, pages 603–609. MIT Press, Cambridge, MA, USA. Pages: 24, 38

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905. Pages: 13, 14

Smola, A. J. and Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning. In *Proceedings of the International Conference on Machine Learning*, volume 17, pages 911–918. Morgan Kaufmann, San Mateo, CA, USA. Pages: 57

Sproull, R. F. (1991). Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589. Pages: 69

Szummer, M. and Jaakkola, T. (2002). Partially labeled classification with Markov random walks. In T. G. Dietterich and S. Becker and Z. Ghahramani, editor, *Advances in Neural Information Processing Systems*, volume 14, pages 945–952. MIT Press, Cambridge, MA, USA. Pages: 148

Teh, Y. W. and Roweis, S. T. (2003). Automatic alignment of local representations. In S. Becker and S. Thrun and K. Obermayer, editor, *Advances in Neural Information Processing Systems*, volume 15, pages 841–848. MIT Press, Cambridge, MA, USA. Pages: 118, 119

Tenenbaum, J., de Silva, V., and Langford, J. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323. Pages: 34, 38, 107

Tibshirani, R. (1992). Principal curves revisited. *Statistics and Computing*, 2:183–190. Pages: 25

Tibshirani, R., Hastie, T., Narasimhan, B., and Chu, G. (2002). Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proceedings of the National Academy of Sciences of the USA*, 99(10):6567–6572. Pages: 18

Tipping, M. E. and Bishop, C. M. (1999). Mixtures of probabilistic principal component analysers. *Neural Computation*, 11(2):443–482. Pages: 22, 50, 108

Ueda, N., Nakano, R., Ghahramani, Z., and Hinton, G. E. (2000). SMEM algorithm for mixture models. *Neural Computation*, 12:2109–2128. Pages: 60

Utsugi, A. (1998). Density estimation by mixture models with smoothing priors. *Neural Computation*, 10(8):2115–2135. Pages: 97

Vapnik, V. (1995). *The nature of statistical learning theory*. Statistics for Engineering and Information Science Series. Spinger-Verlag, New-York, NY, USA. Pages: 23, 47

Verbeek, J. (2000). An information theoretic approach to finding word groups for text classification. Master's thesis, University of Amsterdam, Institute for Logic, Language and Computation, Amsterdam, The Netherlands. Pages: 48

Verbeek, J. J., Roweis, S. T., and Vlassis, N. (2004a). Non-linear CCA and PCA by alignment of local models. In S. Thrun and L. K. Saul and B. Schölkopf, editor, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, Cambridge, MA, USA. Pages: 107

Verbeek, J. J., Vlassis, N., and Kröse, B. J. A. (2001). A soft k-segments algorithm for principal curves. In Dorffner, G., Bischof, H., and Hornik, K., editors, *Proceedings of the International Conference on Artificial Neural Networks*, volume 11, pages

450–456, New-York, NY, USA. Spinger-Verlag. Pages: 25

Verbeek, J. J., Vlassis, N., and Kröse, B. J. A. (2002a). A k-segments algorithm for finding principal curves. *Pattern Recognition Letters*, 23(8):1009–1017. Pages: 25, 108

Verbeek, J. J., Vlassis, N., and Kröse, B. J. A. (2002b). Coordinating principal component analyzers. In Dorronsoro, J. R., editor, *Proceedings of the International Conference on Artificial Neural Networks*, volume 12, pages 914–919. Spinger-Verlag, New-York, NY, USA. Pages: 51, 107, 115

Verbeek, J. J., Vlassis, N., and Kröse, B. J. A. (2003a). Non-linear feature extraction by the coordination of mixture models. In *Proceedings of the Annual Conference of the Advanced School for Computing and Imaging*, volume 8. Pages: 107, 121

Verbeek, J. J., Vlassis, N., and Kröse, B. J. A. (2003b). Self-Organization by Optimizing Free-Energy. In Verleysen, M., editor, *Proceedings of the European Symposium on Artificial Neural Networks*, volume 11. D-side, Evere, Belgium. Pages: 85

Verbeek, J. J., Vlassis, N., and Kröse, B. J. A. (2003c). Efficient greedy learning of Gaussian mixture models. *Neural Computation*, 15(2):469–485. Pages: 53

Verbeek, J. J., Vlassis, N., and Kröse, B. J. A. (2004b). Self-organizing mixture models. *Neurocomputing*. to appear. Pages: 85

Verbeek, J. J., Vlassis, N., and Nunnink, J. R. J. (2003d). A variational EM algorithm for large-scale mixture modeling. In *Proceedings of the Annual Conference of the Advanced School for Computing and Imaging*, volume 8. Pages: 76

Verma, D. and Meilă, M. (2003). A comparison of spectral clustering algorithms. Technical Report CSE 03-05-01, University of Washinton. Pages: 15, 17

Verveer, P. J. and Duin, R. P. W. (1995). An evaluation of intrinsic dimensionality estimators. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):81–86. Pages: 38

Viola, P. and Jones, M. J. (2004). Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154. Pages: 19

Vlassis, N. and Likas, A. (2002). A greedy EM algorithm for Gaussian mixture learning. *Neural Processing Letters*, 15(1):77–87. Pages: 56

Vlassis, N., Motomura, Y., and Kröse, B. J. A. (2002). Supervised dimension reduction of intrinsically low-dimensional data. *Neural Computation*, 14(1):191–215. Pages: 19

Wand, M. P. (1994). Fast computation of multivariate kernel estimators. *Journal of Computational and Graphical Statistics*, 3(4):433–445. Pages: 56

Wang, J., Lee, J., and Zhang, C. (2003). Kernel trick embedded gaussian mixture model. In Gavaldà, R., Jantke, K. P., and Takimoto, E., editors, *Proceedings of the International Conference on Algorithmic Learning Theory*, volume 14, pages 159–174. Spinger-Verlag, New-York, NY, USA. Pages: 17

Ward, J. H. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244. Pages: 9

Webb, A. R. (1995). Multidimensional scaling by iterative majorization using radial basis functions. *Pattern Recognition*, 28(5):753–759. Pages: 32

Webb, A. R. (2002). *Statistical pattern recognition*. Wiley, New-York, NY, USA. Pages:

10, 17, 48, 75, 125

Weiss, Y. (1999). Segmentation using eigenvectors: a unifying view. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 7, pages 975–982. IEEE Computer Society Press, Los Alamitos, CA, USA. Pages: 13

Wieghardt, J. (2001). *Learning the topology of views: from images to objects*. PhD thesis, Ruhr-Universität-Bochum, Bochum, Germany. Pages: 119

Wu, C. F. (1978). Some algorithmic aspects of the theory of optimal designs. *The Annals of Statistics*, 6:1286–1301. Pages: 59

Young, G. and Householder, A. S. (1938). Discussion of a set of points in terms of their mutual distances. *Psychometrika*, 3:19–22. Pages: 30

Zha, H., He, X., Ding, C. H. Q., Gu, M., and Simon, H. D. (2002). Spectral relaxation for k-means clustering. In T. G. Dietterich and S. Becker and Z. Ghahramani, editor, *Advances in Neural Information Processing Systems*, volume 14, pages 1057–1064. MIT Press, Cambridge, MA, USA. Pages: 67

Zhao, Y. and Karypis, G. (2002). Evaluation of hierarchical clustering algorithms for document datasets. In *Proceedings of the international conference on Information and knowledge management*, volume 11, pages 515–524. ACM Press, New-York, NY, USA. Pages: 8

Zhu, X., Ghahramani, Z., and Lafferty, J. (2003). Semi-supervised learning using Gaussian fields and harmonic functions. In Fawcett, T. and Mishra, N., editors, *Proceedings of the International Conference on Machine Learning*, volume 20, pages 912–919. AAAI Press, Menlo Park, CA, USA. Pages: 148

SUMMARY

Many current information processing systems have to process huge amounts of data. Often both the number of measurements as well as the number of variables in each measurement —the dimensionality of the data— is very large. Such high dimensional data is acquired in various forms: images containing thousands of pixels, documents that are represented by frequency counts of several thousands of words in a dictionary, or sound represented by measuring the energy in hundreds of frequency bands. Because so many variables are measured, a wide variety of tasks can be performed on the basis of these sorts of data. For example, images can be used to recognize hand-written digits and characters, but also to recognize faces of different people.

Information processing systems are used to perform several types of tasks, such as classification, regression and data visualization. In classification, the goal is to predict the class of new objects on the basis of a set of supervised examples. For example, in a digit recognition task the supervised examples are images of digits together with a label indicating which digit is depicted in the image. The supervised examples are used to find a function that maps new images to a prediction of the class label. Regression is similar to classification, but the goal is here to predict a continuous number rather than a discrete class label. For example, a challenging regression application would be to predict the age of a person on the basis of an image of his face. In data visualization the goal is to produce an insightful graphical display of data. For example, the results of image database queries can be presented using a two dimensional visualization, such that similar images are displayed near to each other.

In many applications where high dimensional data is used the diversity in the data considered is often limited. For example, only a limited variety of images is processed by a system that recognizes people from an image of their face. Images depicting digits or cars are not processed by such a system, or perhaps only to conclude that it does not depict a face. In general, the high dimensional data that is processed for specific tasks can be described in a more compact manner. Often, the data can either be divided into several groups or clusters, or the data can be represented using fewer numbers: the dimensionality can be reduced.

It turns out that it is not only possible to find such compact data representations, but that it is a prerequisite to successfully learn classification and regression functions from

high dimensional examples. Also for visualization of high dimensional data a more compact representation is needed, since the number of variables that can be graphically displayed is inherently limited. In this thesis we present the results of our research on methods for clustering and dimension reduction. Most of the methods we consider are based on the estimation of probabilistic mixture densities: densities that are a weighted average of several simple component densities. A wide variety of complex density functions can be obtained by combining simple component densities in a mixture.

Layout of this thesis. In Chapter 1 we give a general introduction and motivate the need for clustering and dimension reduction methods. We continue in Chapter 2 with a review of different types of existing clustering and dimension reduction methods.

In Chapter 3 we introduce mixture densities and the expectation-maximization (EM) algorithm to estimate their parameters. Although the EM algorithm has many attractive properties, it is not guaranteed to return optimal parameter estimates. We present greedy EM parameter estimation algorithms which start with a one-component mixture and then iteratively add a component to the mixture and re-estimate the parameters of the current mixture. Experimentally, we demonstrate that our algorithms avoid many of the sub-optimal estimates returned by the EM algorithm. Finally, we present an approach to accelerate mixture densities estimation from many data points. We apply this approach to both the standard EM algorithm and our greedy EM algorithm.

In Chapter 4 we present a non-linear dimension reduction method that uses a constrained EM algorithm for parameter estimation. Our approach is similar to Kohonen's self-organizing map, but in contrast to the self-organizing map, our parameter estimation algorithm is guaranteed to converge and optimizes a well-defined objective function. In addition, our method allows data with missing values to be used for parameter estimation and it is readily applied to data that is not specified by real numbers but for example by discrete variables. We present the results of several experiments to demonstrate our method and to compare it with Kohonen's self-organizing map.

In Chapter 5 we consider an approach for non-linear dimension reduction which is based on a combination of clustering and linear dimension reduction. This approach forms one global non-linear low dimensional data representation by combining multiple, locally valid, linear low dimensional representations. We derive an improvement of the original parameter estimation algorithm, which requires less computation and leads to better parameter estimates. We experimentally compare this approach to several other dimension reduction methods. We also apply this approach to a setting where high dimensional 'outputs' have to be predicted from high dimensional 'inputs'. Experimentally, we show that the considered non-linear approach leads to better predictions than a similar approach which also combines several local linear representations, but does not combine them into one global non-linear representation.

In Chapter 6 we summarize our conclusions and discuss directions for further research.

CURRICULUM VITAE

Jakob Jozef Verbeek was born in Hoevelaken, on December 21, 1975. In 1988 he entered the Hooghelandt College in Amersfoort, where he received his VWO diploma in 1994.

In 1994 he started his studies on artificial intelligence at the University of Amsterdam (UvA). In 1998 he worked on his final project at the Centrum voor Wiskunde en Informatica in Amsterdam (national research institute for mathematics and computer science) under the supervision of Prof. P. Vitanyi, dr. P. Grunwald and dr. R. de Wolf. After completing his masters thesis called "Overfitting using the MDL principle", he was rewarded his Master of Science title cum laude in 1998. From 1998 until 2000 he was enrolled in the Master of Logic programm, for which he also received a cum laude masters degree after finishing his final project "An information theoretic approach to finding word groups for text classification" under the supervision of Prof. M. van Lambalgen.

From 2000 until 2004 he worked as a Ph.D. student in the Intelligent Autonomous Systems (IAS) group at the Informatics Institute of the UvA on the STW funded project "Tools for non-linear data analysis" in cooperation with the Delft University of Technology. In 2003 he visited the Machine Learning group of the Department of Computer Science of the University of Toronto for three months to work with Prof. Sam Roweis. Currently, he is working as a postdoctoral researcher in the IAS group.

He refereed papers for several international journals: Engineering Applications of Artificial Intelligence, IEEE Transactions on Pattern Analysis and Machine Intelligence, International Journal of Computer Vision and Image Understanding, International Journal of Pattern Recognition and Artificial Intelligence, Journal of Artificial Intelligence Research, Journal of Machine Learning Research, Neural Networks, and Pattern Recognition Letters.