Mixture Models for Clustering and Dimension Reduction
Verbeek, J.J.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (http://dare.uva.nl)
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 \( \{x_1, \ldots, x_N\} \) 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-
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:

\[
\begin{align*}
    d_{AB} &= \min_{i \in A, j \in B} d_{ij} \quad \text{(single link),} \\
    d_{AB} &= \max_{i \in A, j \in B} d_{ij} \quad \text{(complete link).}
\end{align*}
\]

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: \( d_{ij} = \|x_i - x_j\|^2 \). 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:

\[
\mu_i = \frac{1}{n_i} \sum_{n \in A_i} x_n.
\]
The sum-of-squares error is then given by:

\[ E_{SoS} = \sum_{i=1}^{k} \sum_{n \in A_i} \| x_n - \mu_i \|^2. \] (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:

\[
d_{A_iA_j} = \sum_{n \in (A_i \cup A_j)} \| x_n - \mu_{(A_i \cup A_j)} \|^2 - \sum_{n \in A_i} \| x_n - \mu_i \|^2 - \sum_{n \in A_j} \| x_n - \mu_j \|^2
\]

\[
= n_i \| \mu_{A_i} \|^2 + n_j \| \mu_{A_j} \|^2 - n_{(A_i \cup A_j)} \| \mu_{(A_i \cup A_j)} \|^2.
\] (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 \( d_{i,j,k} \) 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 \( \mu_i \), 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 \( \mu_i \). 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 \( \mu_i \) and minimize the error with respect to the assignment of the data to the clusters. This amounts to assigning every point \( x_n \) to the cluster \( i = \arg\min_j ||x_n - \mu_j||^2 \). 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 \( D(p_n||q_i) \) 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 \( D(p||q) \) measures how well distribution \( q \) matches distribution \( p \) and is defined as:

\[
D(p||q) = \sum_w p(w) \log \frac{p(w)}{q(w)}.
\]  

(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. (Meila 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 $x_i$. 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 $x_i$ and $x_j$, with a large value indicating great similarity. The weights $w_{ij}$ are collected in the symmetric weight matrix $W$ with an associated diagonal 'degree matrix' $D$ 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' $\text{cut}(A, B)$:

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

(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’ $\text{vol}(A)$, which is the sum of all edge weights connected to points in $A$:

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

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:

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

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 $x$, a value $y_i$:

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

We can now write\(^1\) Ncut in terms of the $y_i$, which we collect in a vector $y$ of length $N$:

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

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:

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

where $\lambda$ is the value of the relaxed Ncut criterion corresponding to $y$. The vector $y = 1$ 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.
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 $x_i$ 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 Meilä, 2003) for overviews. However, often this post-processing employs clustering algorithms susceptible to local optima, e.g. k-means. 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 $w_{ij} = \exp(-||x_i - x_j||^2/\sigma^2)$ 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 $\psi_i$ 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 $x_i$ is among the $k$-nearest neighbors of $x_j$ or vice-versa. Efficient techniques exist to find nearest neighbors among $N$ points in time $O(N \log N)$ for fixed $k$ (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 \(384 \times 288\) 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 \(x\), i.e., \(y = x^T \mathbf{w}\). 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
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 \( X \) be a \( N \times D \) matrix containing \( N \) measurements \( x_n (n = 1, \ldots, N) \), each being a vector in \( \mathbb{R}^D \) stacked as rows in \( X \). 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 \( w \in \mathbb{R}^D \) be the vector that contains the coefficients of the linear combination that gives the new feature. Let \( y = Xw \) be the vector of length \( N \) with the values of the new feature for our data. Since \( y \) will also have zero mean, we can write its variance as:

\[
\frac{1}{N} \sum_{n=1}^{N} y_n^2 = y^\top y / N = w^\top X Xw / N = w^\top Cw.
\]  

(2.14)

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

\[
L(w, \lambda) = w^\top Cw - \lambda (w^\top w - 1).
\]  

(2.15)

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

\[
\partial L / \partial w = Cw - \lambda w = 0
\]  

(2.16)

\[
\Rightarrow Cw = \lambda w.
\]  

(2.17)

thus \( w \) is an eigenvector of \( C \). The second condition implies that the variance corresponding to some eigenvector \( w \) is given by the corresponding eigenvalue \( \lambda \). This can be seen by multiplying both sides of (2.17) on the left with \( w^\top \). 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 \( w \) 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 \( C \) 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 \( C \) can also be obtained from the eigenvectors of the data inner product matrix \( T = XX^\top \). If \( v \) is an eigenvector of \( T \) with eigenvalue \( \lambda \), then \( w = X^\top v \) is an eigenvector of \( C \) with eigenvalue \( \lambda /N \). Note that \( w \) is expressed as a linear combination of the data points, where \( x_n \) is weighted by the \( n \)-th element of \( v \). Depending on whether \( N < D \) (or vice versa) it is computationally more efficient to perform PCA based on \( T \) (or \( C \)). Methods based on \( T \) (or \( C \)) 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 \( C \) nor \( T \). 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 \( Y = XW \) be the \( N \times d \) matrix having in row \( n \) the projection of \( x_n \) on the first \( d \) principal components, with \( W \) the \( D \times d \) matrix with the corresponding \( d \) eigenvectors of \( C \) as columns. The optimal reconstruction \( X \) (in the sense of minimum squared distance between original and reconstructed points) of \( X \) is given by \( X = YW^\top = XWW^\top \).

**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 $T$ 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 $k(x_i, x_j)$ 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: $k(x_i, x_j) = \exp(- \|x_i - x_j\|^2 / (2\sigma^2))$, 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 $\phi(x)$ be the high dimensional feature vector corresponding to x and $k(\cdot, \cdot)$ the corresponding kernel function, i.e. $k(x_i, x_j) := \phi(x_i)' \phi(x_j)$. PCA computes the eigenvectors of the inner product matrix $T$ 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 $K$. Let $\mu = \frac{1}{N} \sum_i \phi(x_i)$ denote the mean of the data in the feature space, then the centered inner products $t_{ij}$ are given by:

$$\hat{k}(x_i, x_j) = (\phi(x_i) - \mu)' (\phi(x_j) - \mu)$$

$$= k(x_i, x_j) - \frac{1}{N} \sum_i (k(x_i, x_i) + k(x_j, x_i)) + \frac{1}{N^2} \sum_{i,m} k(x_i, x_m).$$

Suppose $v$ is an eigenvector of the centered inner product matrix, corresponding to a principal component on which we want to project. We write $r_i$ for the $i$-th element of $v$ and $r$ for the average of its elements. Then, as mentioned in the previous section, the principal component is given by $w = \sum_{i=1}^N r_i (\phi(x_i) - \mu)$. 
To map a new data point $x$ on the principal component in the feature space, we first subtract the estimated mean in the feature space, $\mu$, and then project it on the principal component. Thus the mapping of $x$ on the principal component is given by:

$$
\begin{align*}
(\phi(x) - \mu)^T w &= (\phi(x) - \mu)^T \sum_{i=1}^{N} v_i (\phi(x_i) - \mu) \\
&= \sum_i k(x, x_i)(v_i - \bar{v}) - \frac{1}{N} \sum_{i,j} k(x_i, x_j)(v_i - \bar{v}).
\end{align*}
$$

(2.20)

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 $\sigma^2$ 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 $f(\lambda)$ be a smooth curve of finite length in $\mathbb{R}^D$. The 'projection index' $\lambda_x \in \mathbb{R}$ of a point $x \in \mathbb{R}^D$ is the value of $\lambda$ for which $f(\lambda)$ is closest to $x$ (or the largest such $\lambda$ if multiple exist). The curve $f$ is called a principal curve with respect to $p$ if the expectation under $p$ of points with projection index $\lambda$ equals $f(\lambda)$, i.e. if $E_p[x : \lambda_x = \lambda] = f(\lambda)$. Note that for finite data the principal curve is not well defined.
since for all but finitely many values of $\lambda$ there will be no data with that projection index and thus $E_x : \lambda_x = \lambda$ 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 $f_0$ and set $i = 0$.

2. Find for each $x_i$ in the data set its projection index on the curve $\lambda_{x_i}$.

3. Set $f_{i+1}(\lambda)$ to be the average of data that has a projection index within some distance $\epsilon$ to $\lambda$. Re-parameterize the curve to unit speed.

4. If for all points $x_i$ in the data set $f_{i+1}(\lambda_{x_i}) = f_i(\lambda_{x_i})$ then the curve is consistent and the algorithm is terminated. Otherwise, set $i = i + 1$ 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 $\epsilon$ 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. 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.
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 $x \in \mathcal{X}$ and output vectors $y \in \mathcal{Y}$. A function $f: \mathcal{X} \rightarrow \mathcal{Y}$ 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 $x$ 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 $f(x)$. 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 $g_i(x) = x^T w_i + b_i$ and sigmoidal $g_i(x) = [1 + \exp(x^T w_i + b_i)]^{-1}$. Given a set of input vectors $x_i$ and corresponding desired output vectors $y_i$, the parameters of the neural network are adjusted to minimize the average of the error between $f(x_i)$ and $y_i$. 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
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 \( x \), and their corresponding network outputs \( f(x_i) \).

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 \( \mu_i \). 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 \( g_i \) in a ‘latent’ space (the space of reduced dimension). A ‘neighborhood’ function is used to measure prox-
iminity 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:

$$h_{ij} = \exp(-\lambda||g_i - g_j||^2).$$

(2.22)

where $\lambda$ controls how fast the neighborhood function decays. The training algorithm of the SOM forces clusters with great proximity $\mathbf{h}_{ij}$ to represent similar data.

The training algorithm for SOM cycles through 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 $\mu_i$ in the data space. For each data point $x_n$, 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 $\mu_i$. Then, each cluster center $\mu_i$ is moved toward the data point by an amount proportional to the neighborhood function evaluated at $j$ and the winner, i.e. $\mu_i = \mu_i + \alpha h_{ij}(x - \mu_i)$. Thus, not only the center of the best matching cluster $i'$ moves toward $x$, also the centers of clusters near $i'$ in the latent space move toward $x$. 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 $g_{i'}$. In this manner, the SOM maps the coordinates of a data point $x$ 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 $\mathcal{L}$:

$$p(x) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{N}(x; \mu_i, \sigma^2 I).$$

(2.23)

$$\mathcal{L} = \sum_{n=1}^{N} \log p(x_n).$$

(2.24)
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 \( \{x_1, \ldots, x_N\} \) 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 \( x_i \) and \( x_j \) can be expressed in terms of inner products between data points:

\[
\|x_i - x_j\|^2 = (x_i - x_j)^\top (x_i - x_j) = t_{ii} + t_{jj} - 2t_{ij}.
\]  

(2.26)

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

\[
t_{ij} = -\frac{1}{2} (d_{ij} - \bar{d}_i - \bar{d}_j + d),
\]  

(2.27)

\[
\bar{d}_i = \frac{1}{N} \sum_{j=1}^{N} d_{ij},
\]  

(2.28)

\[
\bar{d} = \frac{1}{N^2} \sum_{i,j=1}^{N} d_{ij}.
\]  

(2.29)

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

\[
T = U^\top A U.
\]  

(2.30)

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

$$T = YY^\top,$$

$$Y = UA^{1/2}. \quad (2.31)$$

i.e. $T$ is the inner product matrix of the set of vectors given by the rows of $Y$. It is not hard to show that the constant vector is an eigenvector of $T$ with eigenvalue zero. This, together with the orthogonality of the eigenvectors, implies that the columns of $Y$, 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, $T$ may have several negative eigenvalues. If $T$ 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 $A$ to zero an proceeding as before (Cox and Cox, 1994). The justification for this procedure is that the $Y$ constructed in this manner minimizes the error function associated with classical scaling (Hastie et al., 2001):

$$E_{CS} = \sum_{i,j} (t_{ij} - (y_i - \bar{y})^\top(y_j - \bar{y}))^2. \quad (2.33)$$

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

If we constrain the $y_i$ 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 $y_i$. 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:

$$E_{ls} = \sum_{i\neq j} (d_{ij} - \|y_i - y_j\|)^2. \quad (2.34)$$
The Sammon mapping (Sammon, 1969) has a similar error function, but weights each term in $E_{is}$ with a factor $d_{ij}^{-1}$. This renders the approximation of small distances more important. The error function is:

$$E_{Sammon} = \sum_{i \neq j} \frac{1}{d_{ij}} (d_{ij} - \| y_i - y_j \|)^2 .$$

(2.35)

The derivative of $E_{Sammon}$ with respect to the coordinates $y_i$ is given by:

$$\frac{\partial E_{Sammon}}{\partial y_i} = -4 \sum_j (y_j - y_i) \left[ \frac{1}{d_{ij}} \frac{1}{\| y_i - y_j \|} \right] .$$

(2.36)

Also several approaches exist that minimize the Sammon mapping error function but constrain the $x_i$ 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 focusing 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 $P$. The $i$-th row of the $N \times N$ transition matrix $P$ contains a
2.2 Dimension reduction

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

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

(2.37)

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

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 $\sigma_i^2$ have to be set and control how fast $p_{ij}$ decays as a function of $d_{ij}$. The authors propose to set the parameters $\sigma_i^2$ 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 $\sigma_i^2$ 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.\footnote{The value of $\sigma^2$ such that $p_i$ has a certain entropy can not be found analytically. However, the entropy is a monotone decreasing function of $\sigma^2$, and a binary search can be performed to find the correct value.}

The goal is to find an embedding $y_1, \ldots, y_N$ in few dimensions that gives rise to similar stochastic neighbors. With an embedding of the data we associate a second transition matrix $Q$, which is based on squared Euclidean distances between the embedding coordinates: $q_{ii}=0$ and for $i \neq j$:

$$q_{ij} = \frac{\exp(-\| y_i - y_j \|^2)}{\sum_{j' \neq i} \exp(-\| y_i - y_{j'} \|^2)}.$$  

(2.38)

Thus $q_{ij}$ is large if and only if $y_i$ and $y_j$ are relatively nearby. We want the transition matrices $P$ and $Q$ 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 $P$ and the corresponding rows of $Q$:

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

(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:

$$\frac{\partial E_{SNF}}{\partial y_i} = -2 \sum_j (y_j - y_i)(p_{ij} + p_{ji}) - (q_{ij} + q_{ji}).$$  

(2.40)
The $j$-th summand in the gradient can be interpreted as a force between $y_i$ and $y_j$. This force is repelling if the transitions between $i$ and $j$ are more likely according to $Q$ than according to $P$, 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 $Q$ than according to $P$.

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 $||y_i - y_j||^2$ 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 $||y_i - y_j||^2$ 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 $||y_i - y_j||^2$ 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 $x_i$ ($i = 1, \ldots, N$) a ‘neighborhood graph’ with $N$ vertices is constructed; data point $x_i$ corresponds to vertex $i$. Each node $i$ is connected by undirected edges to the $k$ vertices that correspond to the $k$ nearest neighbors of $x_i$ in the data space. Alternatively, one can connect all pairs with $||x_i - x_j|| < \epsilon$. 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 $i$ and $j$ is said to have a length that is equal to the Euclidean distance between $x_i$ and $x_j$. 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-
2.2 Dimension reduction

...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 $\epsilon$ 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 $x$, 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 $x$, weights $w_{ij}$ are found that optimally reconstruct $x$, from its nearest neighbors. The weights are constrained to sum to one, which makes the optimal weights invariant to translations of $x$, and its neighbors. Thus, for each point $x$, we minimize the error:

$$e_i = \| x_i - \sum_{j \in N(i)} w_{ij} x_j \|^2.$$

(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: $e_i' = e_i + \sum_{j \in N(i)} w_{ij}^2$.

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 $x$, 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 $y_i$ that aims to minimize the reconstruction error of all the $y_i$ from $y_j$ with $j \in \mathcal{N}(i)$ 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 $y_i$:

$$E_{LLE} = \sum_i ||y_i - \sum_{j \in \mathcal{N}(i)} w_{ij}y_j||^2 = \text{Tr}(Y^\top (I - W)^\top (I - W)Y) \geq 0. \quad (2.42)$$

where $y_i^\top$ is the $i$-th row of $Y$ and the $(i, j)$-th entry of $W$ 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 $y_i$ for the same reasons as $\epsilon$. To obtain a unique solution, the $y_i$ 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 $y_i$ are zero. To obtain non-degenerate solutions with variance in the $y_i$, 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 $(I - W)^\top (I - W)$. 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 $Y$.

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 $x_i$ to all points that have distance smaller than $\epsilon$ to $x_i$. 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 $\exp(-\|x_i - x_j\|^2 / \sigma^2)$ 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 $y_i$ ($i = 1, \ldots, N$) that minimize the sum of pairwise squared distances between the embedded points, weighted by their edge
2.2 Dimension reduction

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

$$E_{LEM} = \frac{1}{2} \sum_{i,j} w_{ij} \| y_i - y_j \|^2 = \text{Tr}\{Y^T(D - W)Y\}. \quad (2.43)$$

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: $Y^T D Y = I$. The optimal embedding coordinates in $d$ dimensions are then found through the $(d+1)$ eigenvectors $v$ with smallest eigenvalues $\lambda$ given by:

$$(D - W)v = \lambda Dv. \quad (2.44)$$

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 $Y$ 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 $W_{LLE}$ to denote the LLE weight matrix, and let:

$$W = W_{LLE} + W_{LLE}^T - W_{LLE}^T W_{LLE}. \quad (2.45)$$

This matrix $W$ 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 $\epsilon$ of a point $x$, grows as a function of $\epsilon$, 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 $\mathbb{R}^{d-1}$, 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 $f(x)$ is convex if for any $x_1$ and $x_2$ and $0 \leq \lambda \leq 1$ it holds that:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2).$$  \hspace{1cm} (2.46)

If a function $f$ is convex, then all local minima of $f$ are also global minima. Thus, to find a minimizer $x$ of a convex function $f(x)$, 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 $Mv$ of vectors $v$ with a $N \times N$ matrix $M$. 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.
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).