Clustering Based on Principal Curve

Ioan Cleju

Master Thesis
University of Joensuu
30.12.2004

Abstract

The topic of the thesis is clustering algorithms that use one-dimensional projections of the data space. Clustering is a very intensive research topic due to its numerous applications and because it is a difficult problem to solve. The high dimensionality and the lack of ordering of the data have motivated research in projection-based methods for dimensionality reduction and their application to clustering.

We make a short review of the main dimensionality reductions techniques, both linear and non-linear, with more emphasis on the principal curves. Then we investigate possible use of projections on the principal axes and principal curves for clustering algorithms. To our knowledge, this approach on clustering has not been applied before. The principal curve offers more advantages by the fact that it can incorporate non-linear correlations of data. We compare the use of the principal curve to the use of the first principal component, a widely used linear one-dimensional subspace.

Based on the algorithm developed for clustering, we introduce a new method for partitioning the space in order to create suitable subsets for the principal curve based clustering. This provides significantly better results than the methods based on curves, and the method is much more robust to the variations of the principal curve parameters.

In the last part of the thesis, we will focus on other important issues in clustering, such as determining the correct number of clusters. We study the projection of the estimated probability density function over the principal curve, and on the basis of this, we determine the number of clusters by detecting local maxima of the probability density function.

Table of Contents

1. Introduction

Clustering is a general classification procedure that divides a set of objects in different classes. Objects from the same class should be similar to each other. There is no initial indication about the classes or about their number but only the properties of the objects in the set. Most clustering applications are related to data mining, data compression and pattern matching applications used in the fields of computer science, engineering or bioinformatics. Due to the general classification purpose of clustering analysis, it is also used for data analysis in sciences like psychology, medicine, sociology or economy.

The objects are usually characterized by their feature vector, so that the objects can be considered as points in a multi-dimensional space. A clustering algorithm aims at finding groupings in the data space so that points similar to each other belong to the same group. The object classes are derived only by analyzing the data space. The classification is unsupervised. Opposed to it, there is another major classification approach known as supervised classification. In supervised classification, we are provided with a training set of data that is already classified; based on it, we have to find classifying rules and apply them for new data.

A typical clustering activity includes five steps [JD1988]:

(1) pattern representation (possibly including feature extraction),
(2) definition of a pattern proximity distance,
(3) actual clustering (grouping),
(4) data abstraction,
(5) assessment of output.

The last two steps are not necessary, and their completion depends on the actual purpose of the clustering task. Depending on the algorithm, there might be a feedback among the steps. From step (3), the algorithm might return to one of the first steps.

Pattern representation refers to the number of clusters, patterns and features. Some of them might be provided to the algorithm by assumptions or by a previous pre-processing step. Depending on this, the algorithm might include estimation of the number of clusters, feature extraction or feature selection. *Pattern proximity* usually refers to defining a distance relation that can be evaluated for each pair of patterns. The distance between two patterns actually reflects the dissimilarity between the patterns, but it is by far more used than a similarity relation.

Grouping the data is the core process of clustering. The algorithms typically optimize a clustering criterion that characterizes things like the variance of data within one cluster. In the *data abstraction* step, a simple and compact representation of each cluster is extracted. It is usually represented by finding a good representative of that particular cluster, such as centroid. In the last step, known as *cluster validity*, the optimality of the solution is studied.

Despite the long time the clustering methods have been studied, research on this topic is still intensive. The variety of data spaces implies that no generally designed algorithm can best suit all applications. For each application, a quality measure for the algorithm is stated, and usually a metric is defined on the data space. A criterion for the separation of the clusters has to be defined, and it is usually dependent on the application. The goodness of the solution can be considered only in relation to the quality measure criterion. In Figure 1.1, different shapes of clusters can be observed.

Classification is also a matter of perception: different people classify the same data set differently. Above all, even without considering the problems of finding a metric in the space, a measure of quality and the number of clusters in a data set, the clustering problem is very difficult (see Section 1.1)[SCH1975].

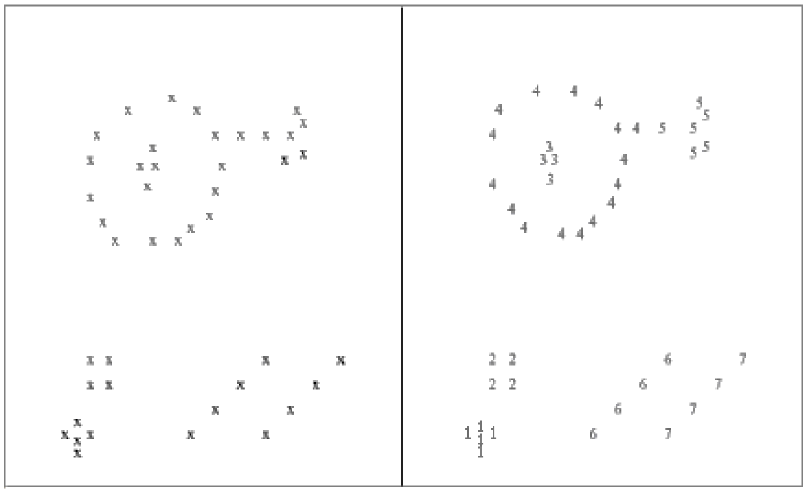

Figure 1.1. Labeling data in clusters [JMF1999].

1.1. Motivation

Clustering is a difficult problem. It is related to another widely known problem, *vector quantization* (VQ). The clustering problem tries to organize the data into homogenous groups while vector quantization tries to map the data to a reduced discrete space, minimizing the representation error. For the clustering problem, minimizing a function does not always guarantee that the classes will be correctly found because of the different properties of data sets or subsets. Moreover, the number of different classes is generally unknown.

The clustering problem is more general and thus more difficult than VQ problem. VQ itself has been proven to be NP-complete [SCH1975]. For an NP-complete problem, there are no algorithms that can solve it in polynomial time. This leads to search for algorithms that do not necessarily find the optimal solution, but have good performances regarding both the quality of solution and the time complexity.

A slightly different case is the *scalar quantization* (SQ) problem, which is a one-dimensional special case of VQ. The most important property of SQ is that there are algorithms that can solve it optimally in polynomial time. The difference to VQ is the fact that the scalar space is naturally ordered, and it can be proven (see Section 2.2) that the optimal clusters form subsequences of the data sequence. Using this supplemental information, the algorithms can be designed to find the optimal solution in polynomial time.

Unfortunately, such an order relation does not exist in the multi-dimensional vector space. However, a data-dependent ordering of vector space can be constructed artificially, and a similar algorithm to the one that solves SQ could also yield to good results for VQ and clustering. In this work, we will mainly focus on finding a good order in multi-dimensional spaces and apply them to clustering.

The natural order relations for vector space can be given by one-dimensional projections of the data. This approach has already been considered for color quantization by using the first principal component [W1992]. Good results have been obtained only for simple data sets that have the data distribution along the principal axis. Usually, the first principal component corresponds to the principal axis of the data. We will study further the usability of the first principal component, and then we propose to use the principal curve as the one-dimensional projection space. Principal curves can describe the nonlinearities of the data better than the principal axis, and it therefore can have better precision in approximating it.

We will discuss possible adjustments to the order given by the one-dimensional projections by solving the shortest Hamiltonian path within the data set. We also consider using the principal curve for estimating the number of clusters in the data set, by constructing the probability density function, projecting it on the curve and detecting its local maxima.

1.2. Problem Definition

Let us consider a set $X=\{x_1,\dots,x_N\}$ of N data objects. Each object x_i is characterized by M features $x_i^1,\dots,x_i^M$. A partition $P=\{p_1,\dots,p_N\}$ specifies for each object in X an index for the category it belongs to. Each different index of P specifies a different category. The categories are called *clusters* and a cluster c_a is defined as the set:

$$c_a = \{x_i \in X \mid p_i = a\}\,. \tag{1.1}$$

Clustering is defined as the set of clusters $C=\{c_1,\dots,c_M\}$ such that:

$$\bigcup_{i=1}^{M} c_i = X\,, \tag{1.2}$$

$$c_i \cap c_j = \phi \quad 1 \le i \le M, 1 \le j \le M, i \ne j\,. \tag{1.3}$$

In some applications, it is of interest to assign to each cluster a representative. Thus, we define the *codebook* Q as the set of the representatives for every cluster: $Q=\{q_1,\dots,q_M\}$. There exists a mapping from C to Q but not obligatory one from Q to C.

The quality of a clustering is defined by the objective function f, called criterion function or clustering criterion, which gives a real number for every possible partition. The clustering criterion specifies the quality of the clustering C for the set X. The clustering problem aims to optimize the value of the clustering criterion.

In this work, we will assume the Euclidean metric and use the *mean square error* (MSE) as the clustering criterion. Considering the notations introduced, the MSE for the set X having the partition P and the codebook Q is:

$$f(X,Q) = \frac{1}{N}\sum_{i=1}^{N}\left\| x_i - q_{p_i} \right\|^2 , \tag{1.4}$$

where $\|*\|$ denotes the Euclidean norm. Notice that the partition P uniquely determines the clustering C.

For the MSE criterion, the goal is to minimize the value of the criterion. A natural choice for the codebook is to assign the centroid of the cluster as its representative code vector, because it determines the minimal error. By optimizing the MSE criterion, the clusters are necessarily *Voronoi cells*: each data point corresponds to the cluster whose centroid is closer to it. Thus, specifying one of P, C or Q uniquely determines the other two.

While the general problem of clustering considers determining the number of clusters as a part of the problem, there is also a simplified approach: *K-clustering*. The K-clustering problem refers to finding a clustering of size K so that the objective function is optimized.

Except for Section 7, in the rest of the work we will consider the K-clustering problem. The clustering criterion will be also referred to as *distortion function* or just distortion. The distortion of clustering is defined as the sum of all squared distances from data points to their corresponding code vectors. MSE is the average of the distortion considering the number of points. Since the size of a data set is a constant, minimizing the distortion is equivalent to minimizing the MSE.

1.3. Clustering Techniques

There are many approaches to clustering. The hierarchy in Figure 1.2 presents the main clustering techniques. At the top level, we can differentiate two main approaches: hierarchical and partitional. Hierarchical algorithms produce a nested series of partitions while partitional methods produce one partition [JMF1999].

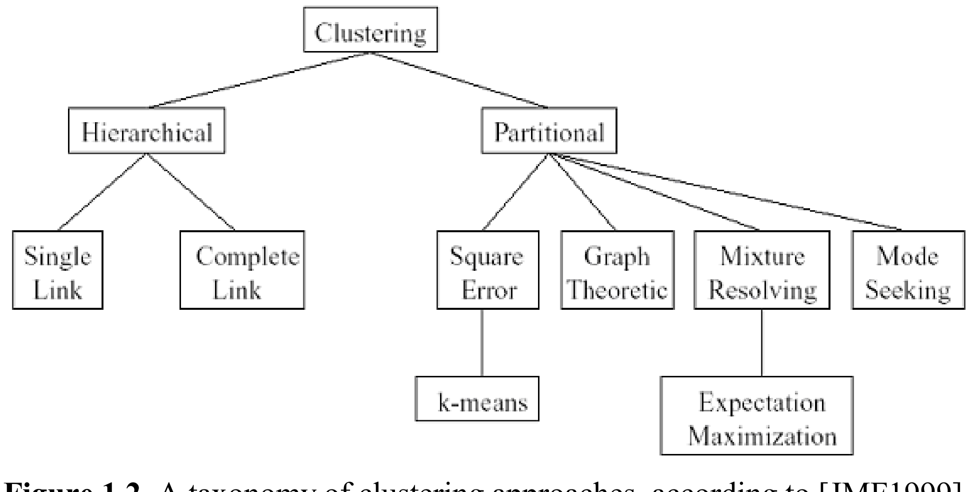

Figure 1.2. A taxonomy of clustering approaches, according to [JMF1999].

A hierarchical algorithm produces a *dendogram* (see Figure 1.3), which represents the nested groupings of patterns together with the similarity level of the groups. The dendogram is then broken at a convenient level. Typically, in the beginning each data point is assigned to its own cluster and then the clusters are merged until the whole data set will be found inside one cluster. This is known as *agglomerative clustering*. *Hierarchical divisive algorithms* construct the dendogram top-down. The thing that differs among the algorithms is the way the similarity of two clusters is computed. Most of the hierarchical algorithms are variants of *single link*, *complete link* or *minimum variance* approaches [JMF1999].

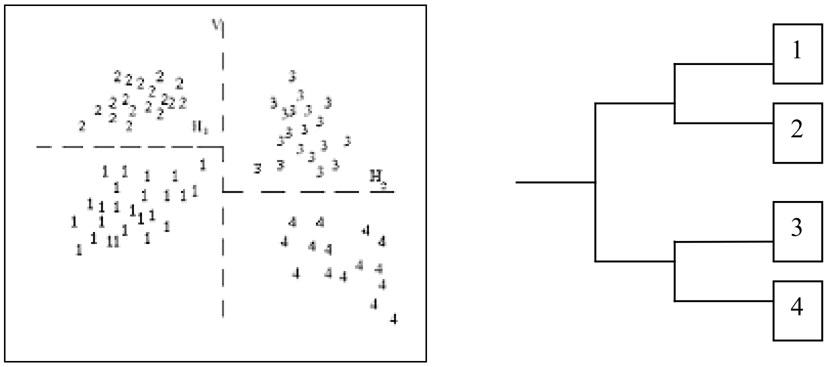

Figure 1.3. An example of hierarchical clustering [JMF1999] and the dendogram.

The most frequently used algorithms are *squared error algorithms*, e.g. *K-means* [M1967]. The algorithm finds a local minimum of the MSE; the results are highly dependent on the initialization. K-means algorithm randomly chooses *K* initialization partitions; then it iterates adjustment of codebook and repartition until codebook remains unchanged (see Figure 1.4). Repartitioning step aims at finding an optimal clustering for the codebook and codebook adjustment finds the codebook as centroids of the clusters.

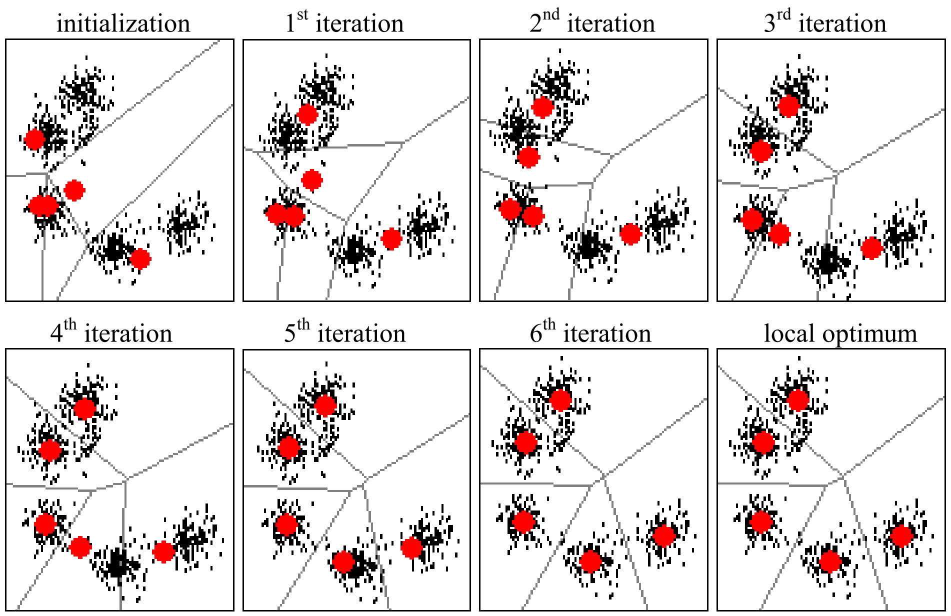

Figure 1.4. Example of K-means algorithm, for 5 clusters.

Graph theoretic approaches consider the whole data set as a graph. Edge weights are usually assigned as similarity/dissimilarity measures between data points. The most used graph theoretic approach considers the *minimal spanning tree*, and deletes the edges that are too expensive [Z1971] (see Figure 1.5). This approach is very similar to the hierarchical approach explained earlier.

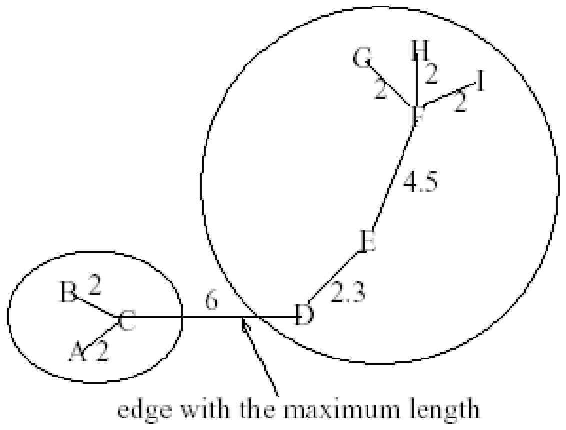

Figure 1.5. Minimum spanning tree used for clustering [JMF1999].

Mixture resolving and *mode-seeking* approaches assume that the clusters were generated according to some unknown distributions. There are methods that try to determine the parameters and the number of these distributions. Usually, it is assumed that the distributions types are known (e.g. Gaussian).

Fuzzy clustering assigns to each cluster a fuzzy set of all the patterns, each pattern having a probability to belong to one cluster, as shown in Figure 1.6. *Artificial neural networks* are popular for clustering nowadays, especially with *Kohonen's learning vector quantization* and *self-organizing maps*. *Evolutionary algorithms* have also been used: *genetic algorithms*, *evolution strategies* and *evolutionary programming*. Among them, genetic algorithms offer the best results [F2000].

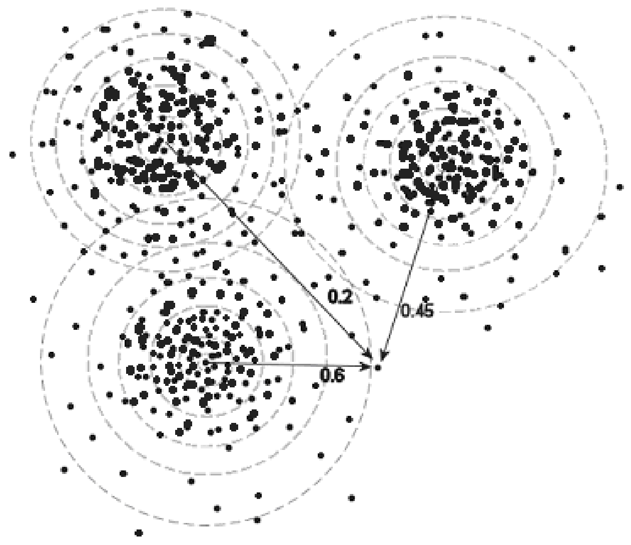

Figure 1.6. Illustrative example of fuzzy clustering.

1.4. Outline of the Thesis

The rest of the thesis is organized as follows. Section 2 defines order constrained clustering, compares it with scalar quantization, and presents an optimal approach to solve it by dynamic programming. Section 3 details the clustering algorithm based on projection on principal axis as an example of how to use one-dimensional projections for clustering.

Section 4 presents linear dimensionality reduction techniques, emphasizing principal component analysis. Section 5 continues the dimensionality reduction techniques by presenting non-linear methods, and detailing different definitions for the principal curves. Section 6 introduces the proposed algorithm for clustering based on projection on the principal curve of the data set. The basic idea is then extended from curves to tree structure in order to find a better layout of the data than by using the principal curve. The partition is done by using the minimal spanning tree.

Section 7 considers the estimation of the number of clusters based on the principal curve using the probability density function. Section 8 presents experimental results of all the studied methods. Conclusions of the thesis are given in Section 9.

2. Order Constrained Clustering

It is possible to guide the clustering process if explicit knowledge about the data domain properties is known, like specific properties of the clusters or possible correlations among the data samples. The supplemental knowledge is usually incorporated in the clustering algorithm and the new problem formulation is known as *constrained clustering*. The constraint imposed to the solution limits the solution space, thus lowering the complexity of the algorithms.

Order constrained clustering is a special case of clustering. It assumes that the data set is ordered by relations so that the data forms a sequence and the clusters are subsequences. O*ptimal order constrained clustering* refers to splitting the sequence in subsequences that optimize the clustering criterion. The advantage of this formulation is its reduced time complexity of the algorithms. While the unconstrained problem is NP-complete, the order constrained formulation can be optimally solved in polynomial time [G1980].

The imposed order constraint may have different justifications. The first one is related to the application domain. The clusters might be imposed as subsequences of the data sequence, for example some events ordered in time. One may want to segment the events keeping the time relation, so that the grouped events are consecutive. The second constraint type is related to the connection between a general order relation of the data space and the clustering criterion. An example is scalar quantization dependency to the order of real numbers. Optimal scalar quantization will always obtain clusters as subsequences of the set. Finally, the third constraint type is imposed just to reduce the complexity of the algorithm. This constraint is data dependent and will not guarantee that the optimal constrained clustering solution will be the same as the optimal clustering solution. The latter approach will be used in this work.

In this work, we will reformulate the clustering problem for multi-dimensional vector space as an order constrained clustering problem. Unlike scalar quantization, no general order in the vector space can be used for order constrained clustering in order to give optimal result for the unconstrained problem, as well. Instead, we will generate an order that is data dependent and use it as a constraint for the vector data set.

A simple way to order the set is to use the projections on the principal axis for this task. . The first principal component is a linear one-dimensional subspace that maximizes the variance of the data and can be considered as the principal axis of the data set. We will order the set based on the order on the principal axis. In order to better model the non-linear correlations in the space, we will prolong the method and use the projections on the *principal curve* [H1989]. Both methods lose accuracy of data representation in the projection phase. One way to avoid this is to consider the shortest Hamiltonian path in the data. The shortest Hamiltonian path considers minimizing the path between two data points that crosses all the data points only once. One difficulty with this approach is that this problem is also NP-complete [GJ1979]. In Figure 2.1, we present the ordering methods stated before in an intuitive way.

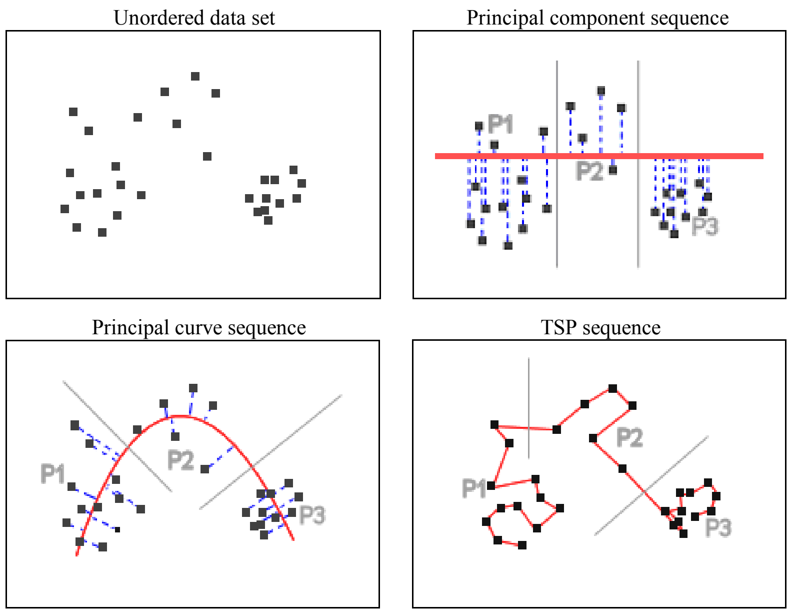

Figure 2.1. Partitions based on different orders for a two-dimensional data set: principal component, principal curve and shortest Hamiltonian path.

As the sequence is constructed by one of the methods described before, optimal splitting to subsequences is similar. It can be optimally solved by dynamic programming in polynomial time. This approach will be later detailed, after a short introduction to dynamic programming.

2.1. Dynamic Programming Technique

The dynamic programming paradigm is widely used for optimization problems. It can be applied to any problem if the solution is obtained after a sequence of decisions. At each step, the taken decision optimizes the criterion function. The value of the function from the previous step is increased by the cost caused by the current step. This can be written in a formal way as a recurrence relation:

$$F(n,q) = opt_{k \in Q}(F(n-1,k) + step(k,q)) \qquad (2.1)$$

For the dynamic programming approach to optimally solve the recurrence relation above, *the principle of optimality* must be satisfied. There are more equivalent formulations for this principle, but all of them say that *whatever the initial state is, the remaining decisions must be optimal with regard to the state following from the first decision*.

Dynamic programming is a *bottom-up* technique that explicitly solves the sub-instances of the problem and stores the partial results, progressively constructing larger subsolutions. The optimal partial results are stored in memory. Dynamic programming can be efficiently used only when the number of partial results is not prohibitive.

2.2. Optimal Scalar Quantization

Scalar quantization is the special case of clustering on one-dimensional space that considers MSE as the optimization criterion. In this case, it can be easily proven that optimal clusters must be sub-sequences of the initial sequence: the MSE criterion implies the Voronoi cells as clusters and the concave shape of the clusters. In the one-dimensional space, the concavity condition determines that the clusters be subsequences of the sequenced data set. This fact can be observed in Figure 2.2.

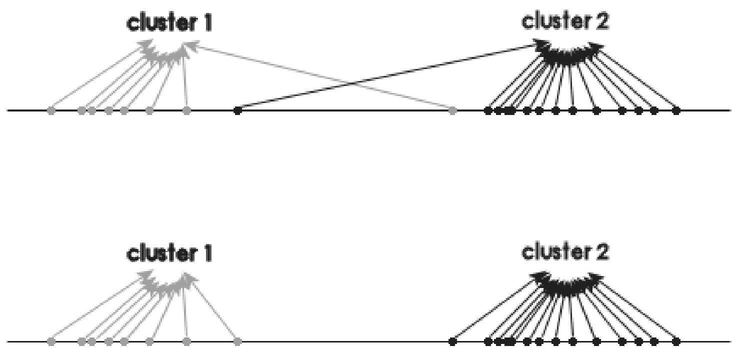

Figure 2.2. Optimal scalar quantization always obtains compact clusters.

The quantization problem can be reformulated as the *K-links shortest path problem*. Let us consider an oriented weighted graph $G = (V, E)$, $V = (v_1,\ldots,v_n)$ that has a sequence among the vertexes, and its edges correspond to:

$$e_{ij} = v_i v_j \in E \Leftrightarrow order(v_i) \le order(v_j) . \qquad (2.2)$$

In other words, there is always a path from a node to all the other nodes that appear later in the sequence. The K-links shortest path problem consists of finding the path from the v_1 to v_n consisting of K links that minimizes the total weight.

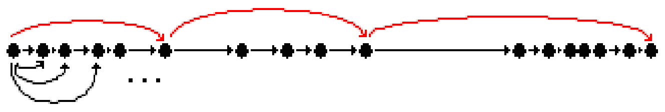

Figure 2.3. K-links shortest path problem for $K = 3$; the edges that form the solution are shown above the sequence.

In order to solve scalar quantization problem by the K-links shortest path, the weights of the graph are computed accordingly. Each point in the data set has a corresponding node in the graph. The order relation among the real data points is used to order the nodes, as well. The weight of an edge e_{ij} that connects the nodes v_i and v_j is equal to the distortion of a cluster that contains all the nodes between v_i and v_j. By minimizing the K-links path, the algorithm finds the optimal quantizer having a codebook with dimension K.

The actual algorithm for the K-links shortest path problem is detailed in Figure 2.4. The desired number of clusters is K and the size of the data set is N. The algorithm uses the tables D[K,N] for storing the partial costs of the paths and TB[K,N] to trace back the optimal solution.

```
PROCEDURE DP_KLSP

Parameters: $G$, $K$
Output: $Q = \{q_1,\ldots,q_K\}$
Locals: D[$K$,$N$], TB[$K$,$N$]

BEGIN
   FOR i := 1 TO N
       D[1,i] = Weight(1,i);
   END
   FOR k := 2 TO $K$
       FOR i := k+1 TO $N$-$K$+k
           D[k,i] := min { j = k : i / D[k-1,j] + Weight (j+1,i)};
           TB[k,i] := arg min { j = k : i / D[k-1,j] + Weight (j+1,i)};
       END
   END
   limit := $N$;
   FOR k:=$K$ TO 2
       $q_k$ := $edge_{TB[k,limit],limit}$;
       limit := TB[k,limit];
   END
   $q_1$ := $edge_{1,limit}$;
END-PROCEDURE
```

Figure 2.4. Pseudo-code of the K-links shortest path algorithm.

Usually, the construction of the dynamic programming matrix takes $O(KN^2)$ time. However, for the scalar quantization it has been shown [W1991, SJ1993] that the complexity of $O(KN)$ can be achieved.

2.3. Optimal Order Constrained Clustering

The K-links shortest path algorithm can be used to any ordered data set, not necessarily to one-dimensional sets. Once the data set has been sequenced, the order constrained clustering algorithm can be formulated using the K-links shortest path. The partial distortions are computed in the vector space. It is very important that we use the one-dimensional space only to generate the sequence. The distortion is computed on the original multi-dimensional space. This is actually natural, because we aim to optimize the MSE criterion on the vector-space and not on the projection space. Depending on the projection used, the adjacency of the points in the vector space might present important differences to the adjacency in the projection space. Thus, the partial distortions computed on the projection space do not probably reflect the correspondent partial distortions on the vector space.

The algorithm for optimal order constrained clustering is presented in Figure 2.5. The input for the algorithm consists in the data set *X* and the size of the codebook *K*, and the output is the generated codebook.

```
PROCEDURE Order_Constrained_Cluster

Parameters: X, K
Output: Q = {q_1,…,q_K}

BEGIN
    ORDER_DATASET (X);
    Q := DP_CLUSTER (X, K);
END-PROCEDURE
```

Figure 2.5. Pseudo-code of order constrained clustering algorithm.

Order constrained clustering has been studied [G1980], and polynomial time algorithms have been given to solve it optimally. The algorithm is based on the following recursion:

$$t(1,i) = s(1,i)(1 \le i \le N) \tag{2.3}$$

$$t(k,j) = \min_{k-1 \le i \le j-1}\left[t(k-1,i) + s(i+1,j)\right] : (k \le j \le N; 2 \le k \le N), \tag{2.4}$$

where $s(i_1,i_2)$ is the minimum distortion considering one cluster between i_1 and i_2, and *t* is the dynamic programming matrix; $t(k,j)$ keeps the total minimum distortion between 1 and *j*, using *k* clusters. Assuming that there are *K* clusters, $t(K,N)$ contains the optimal distortion value for the whole set. This formulation is equivalent to the K-links shortest path formulation for order constrained clustering.

Let us consider the ordered data set *X* of size *N*. The algorithm in Figure 2.6 will find the optimal codebook of size *K* in presence of the order constraint. The algorithm uses the tables D[*K*,*N*] for storing the partial distortions (the matrix *t* from the recursive formulation) and TB[*K*,*N*] to trace back the optimal solution. The algorithm is practically the same as the one that solves the K-links shortest path, and the difference comes from the fact that the weights are now computed on the basis of distortion, instead of being contained in the graph. Notice that the implementation of the for-loops of the algorithm can be optimized [FKSC2000]; we give the form in Figure 2.6 because it is easier to be understood.

One should notice that the optimal order constrained vector quantization does not give the optimal result for the vector quantization problem. The optimality of the result strongly depends only on the quality of the order.

```
PROCEDURE DP_Cluster

Parameters: $X$, $K$, $N$=|$X$|
Output: Q = {$q_1$,…,$q_K$}
Locals: D[$K$,$N$], TB[$K$,$N$]

BEGIN
    FOR i := 1 TO $N$
        D[1,i] = Distortion (1,i);
    END
    FOR k := 2 TO $K$
        FOR i := k+1 TO $N$-$K$+k
            D[k,i] := min { j = k:i / D[k-1,j] + Distortion (j+1,i)};
            TB[k,i] := arg min { j = k:i / D[k-1,j] + Distortion (j+1,i)};
        END
    END
    limit := $N$;
    FOR k:=$K$ TO 2
        $q_k$ := Centroid (TB[k,limit],limit);
        limit := TB[k,limit];
    END
    $q_1$ := Centroid (1, limit);
END-PROCEDURE
```

Figure 2.6. Pseudo-code of DP-based order constrained clustering algorithm.

3. Principal Axis Based Clustering

In this section, we study optimal order constrained clustering using the principal axis to sequence the data set. The order will be given by the order of the projections. The first principal component is the one-dimensional direction of the data set that maximizes the variance of the data and can be considered as the principal axis of the data set. This is the one-dimensional linear projection that is assumed to contain the most information. Details about how the first principal component is calculated will be given in Section 4.

The algorithm consists of three main steps (see Figure 3.1):

(1) computation of the first principal component as the principal axis,
(2) projection of data set on the principal axis and ordering the set, and
(3) optimal clustering for constrained data set.

We will refer to this as *principal axis based clustering* (*PAC*) algorithm. Keeping the same notations as in the previous sections, the pseudo-code of the algorithm is given in Figure 3.1, and the process of the algorithm is illustrated in Figure 3.2.

```
PROCEDURE PAC

Parameters: X, K
Output: Q = {q_1,…,q_K}
Locals: Principal_Axis

BEGIN

    Principal_Axis := Make_Principal_Component (X);
    Order_Data_Set (X, N, Principal_Axis);
    Q := DP_Cluster (X, K, N);

END-PROCEDURE
```

Figure 3.1: Pseudo-code of the principal axis based clustering (PAC).

The quality of the clustering is determined by the ability of the principal axis to fit the data. It has been observed [W1992] that the colors in a real image are distributed along one direction. Indeed this is true; the channels of colors are linearly correlated so that the data set is concentrated along the first diagonal. This special type of distribution of the data set makes it natural to approximate the real data distribution by the principal axis.

The author observes that the splitting strategy used by many algorithms (e.g. median cut) could be optimized for the vector sets that reveal this property. Some hierarchical methods partition the data space cutting it in two sets, then recursively cutting the new sets. For color spaces the first cuts are normal to the principal axis. The proposed solution [W1992] to optimize these cuts by considering a dynamic programming procedure that would optimally split the data space in more sets, with the partition (hyper) planes normal to the principal axis. This quantizer obtained by optimally splitting the principal component is named *optimal principal quantizer* [W1992].

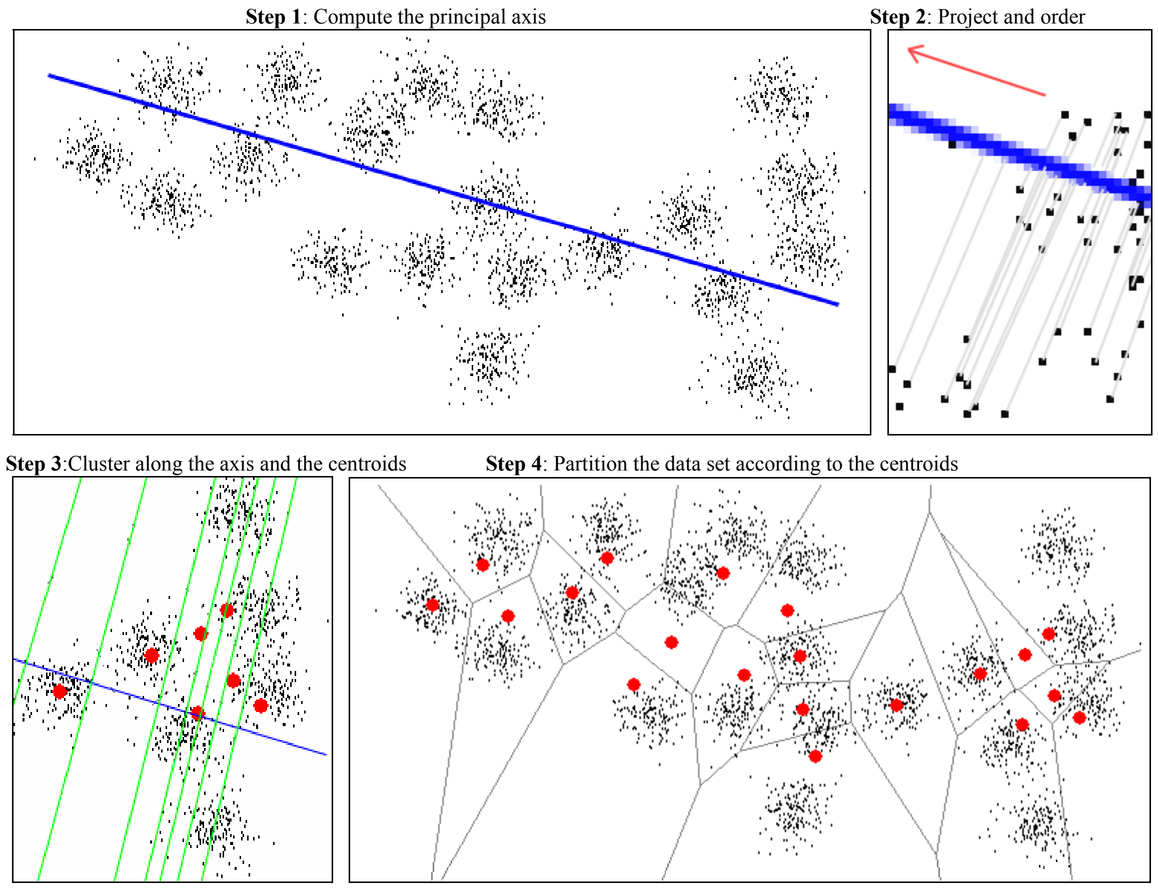
Step 1: Compute the principal axis
Step 2: Project and order
Step 3:Cluster along the axis and the centroids
Step 4: Partition the data set according to the centroids

Figure 3.2. Clustering example using the principal axis.

We have used this method for clustering of general data sets. The results are good if the data sets are simple. This is actually an expected result because too complicated data cannot be modeled by the principal axis. The method performs well as long as the projections of the clusters do not overlap too much.

Along the principal axis, the dispersion of the clusters is good, but along the other dimensions the dispersion is too small. The reason is that the clusters that appear close to each other on the principal axis can be actually quite far from each other. Thus, data elements that are adjacent in the order relation are close on the principal axis, but they might be far on other linear dimensions, see Figure 3.3.

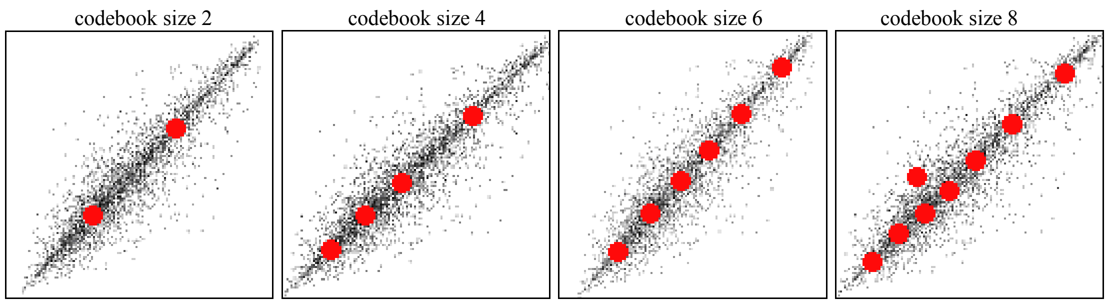

Figure 3.3. Real image data set: the codebook is along the principal axis.

3.1. Tuning by K-means

Next, we consider some possible improvements to the method presented, keeping the main approach unchanged. K-means tuning is a general method used to improve clustering results by finding a local minimum via local changes.

As the results of principal axis clustering are not very good as such, K-means tuning becomes almost compulsory. The only difference to the core algorithm is the addition of K-means step in the end. The pseudo-code is shown in Figure 3.4.

```
PROCEDURE PAC_Kmeas

Parameters: X, K
Output: Q = {q_1,…,q_K}
Locals: Principal_Axis

BEGIN

    Principal_Axis := Make_Principal_Component (X);
    Order_Dataset (X, N, Principal_Axis);
    Q := DP_Cluster (X, K, N);
    Q := Kmeans (X, Q);

END-PROCEDURE
```

Figure 3.4. Clustering on the principal axis, followed by K-means.

As it is seen in the examples from Figure 3.5, the improvements are considerable. If the data varies significantly more along the principal axis direction than along other directions, principal axis based clustering can be successfully used as a method of initialization for K-means. The method fails when the data presents variations in more orthogonal dimensions.

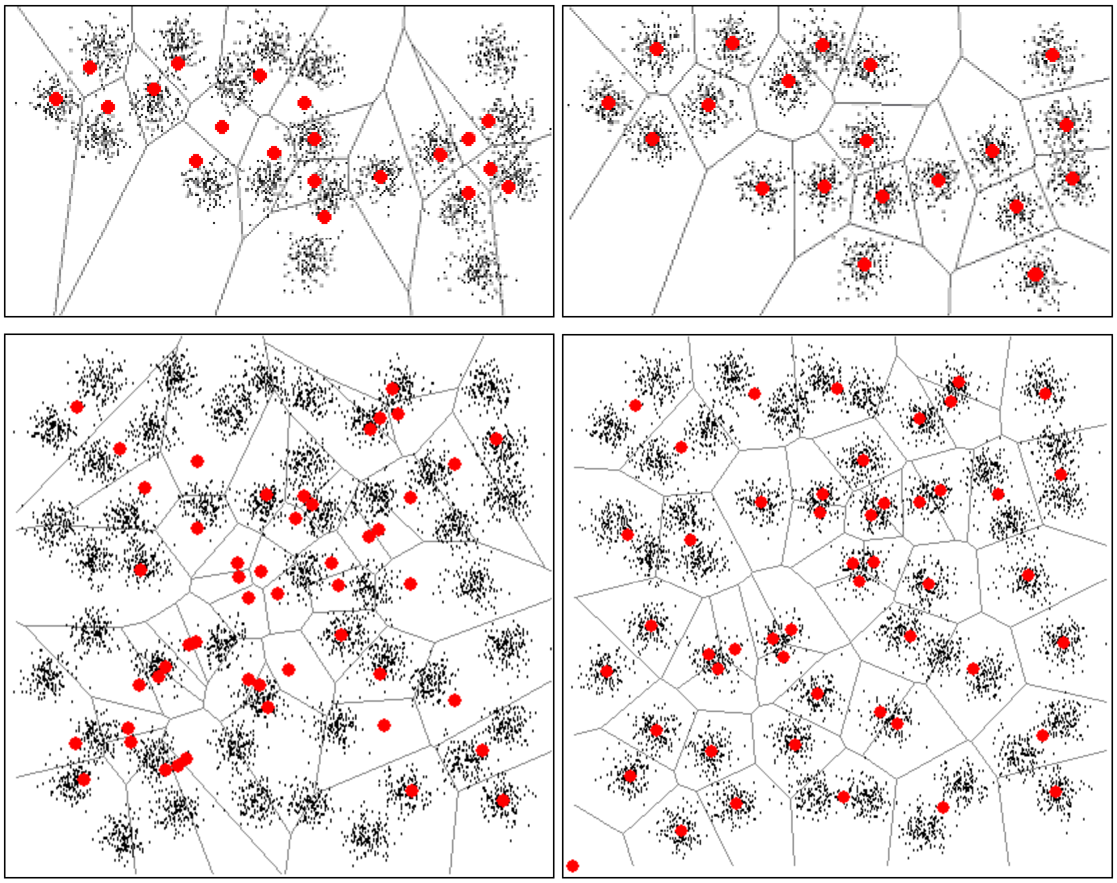

Figure 3.5. Results of the K-means initialized by principal axis clustering; initial results of principal axis based clustering (on left), and iterated by K-means (on right).

3.2. Revising the Order by Hamiltonian Shortest Path

We also consider a new idea of revising the sequence that was generated using the projections on the principal axis. Since the dynamic programming guarantees the optimality of the result according to the sequence, the quality of the result highly depends on the order in the sequence.

The quality of the order depends on how well the spatial relationship of the data points is preserved during the projection from higher dimensional space. Unfortunately, the projection on the principal axis can introduce great distortion in some cases: the adjacency of points on the projection line does not necessarily correspond to the adjacency of data points in the original space. Therefore, we refine the sequence order to overcome this problem.

We aim at shortening the length of the path from the first element to the last one according to the order by considering the *Traveling Salesman Problem* (TSP). However, this is also known to be an NP-complete problem, so we can only approximate the shortest path. We will shorten the length of the path sequence by local optimizations and use the new order of the path further on the clustering procedure.

We consider the sequence given by projections as the first approximation of the TSP and then we improve it. We consider short subsequences for which we can easily find the shortest sequence. The algorithm is iterative, and at each step we increase the subsequence size. As the subsequences become larger, we do not reorder individual data points, because the running time would become prohibitive. Instead, we reduce the precision and move the location for whole groups of data points instead of individual data points.

The difference to the basic clustering algorithm along the principal axis is the addition of the new step to modify the order, see Figure 3.6. The algorithm will be also referred to as PCA_TSP. The results show significant improvements in terms of minimizing the MSE, and the difference can be visually seen as well in Figure 3.7. However, even though the modified algorithm provides a much better order of the data points, the global settlement does not improve when the clusters are fine-tuned by K-means. In fact, the results are usually even worse. The reason is that clustering along the principal axis assures a good dispersion of clusters along this direction. This advantage will probably disappear as the dimensionality of the data set increases, or the variance of data along directions normal to the principal axis becomes higher.

If the clusters are clearly separable, the TSP path sequence might contain each cluster as subsequence. This does not happen if the clusters are overlapping; in this case, it is possible that the path would go through the same cluster more times. Therefore, it seems that the best order should be flexible enough to capture the global layout of the clusters but not too detailed, in order to prevent the path jumping between clusters, or going through the same cluster several times.

```
PROCEDURE PAC_TSP

Parameters: X, K
Output: Q = {q_1,…,q_K}
Locals: Principal_Axis

BEGIN

    Principal_Axis := Make_Principal_Component(X);
    Order_Dataset (X, N, Principal_Axis);
    Tune_Order_TSP (X);
    Q := DP_Cluster (X, K, N);

END-PROCEDURE
```

Figure 3.6. TSP tuned clustering on principal axis (PCA_TSP).

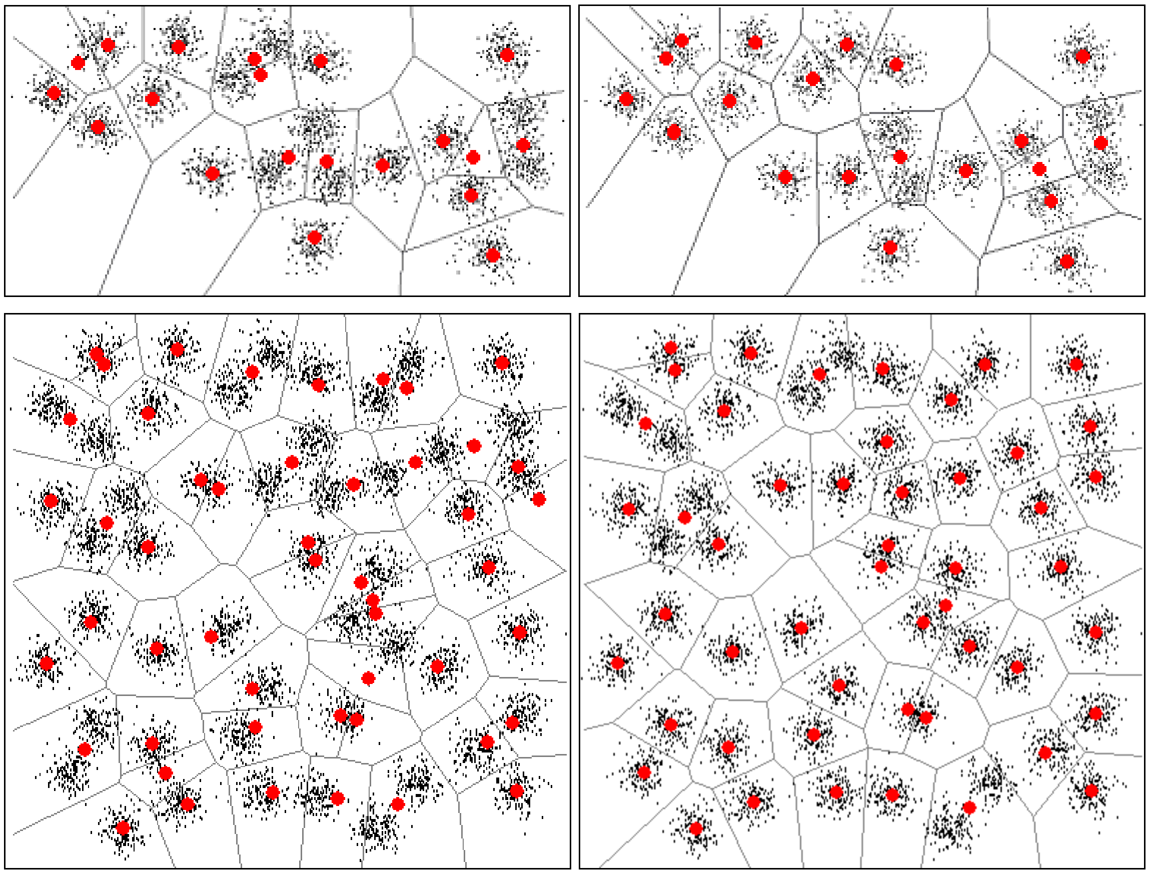
Figure 3.7. Original results of PCA_TSP (on left), and tuned by the K-means (on right).

3.3. Complexity of the Method

The algorithm contains several steps: principal axis computation, ordering of the projected points, clustering in the one-dimensional space, and possible fine-tuning by the K-means. As the first principal component is considered the principal axis, it is determined by the power method [D1997], which will be detailed in Section 4. The complexity of the method is linear, but the convergence speed depends on the distribution of the data. For simplicity, we approximate it here by O(N).

The next step consists of computing the projections and forming the sequence. The projection part takes linear time and the sorting of the projections takes O($N\log N$) time. The most time consuming part of the algorithm is the dynamic programming, having the complexity of O(KN^2). This also gives the overall complexity of the algorithm as O(KN^2).

If fast matrix search [W1991] is applied, the complexity of the dynamic programming part will be reduced to O(KN). In this case, the complexity of the algorithm will be O(NK+$N\log N$)). Depending on the values of K and N, the bottleneck of the algorithm is either the sorting part executed while sequencing or the dynamic programming.

4. Linear Dimensionality Reduction

In the previous section, we have shown the clustering method based on the projections on the principal axis, and we have considered the first principal component as the principal axis. The same basic algorithm can be applied using the projections on any one-dimensional subspace. We will present in more details the principal component analysis and introduce other linear dimension reduction techniques.

The goal of dimensionality reduction is a better and more compact characterization of the data, discarding unimportant features. The variability of the data features can usually be modeled by smaller number of variables than it originally has. It can be stated that the data has less *degrees of freedom* than dimensions [G2002].

The dimensionality reduction also helps in the *curse of dimensionality* [B1961]. The curse of dimensionality refers to the exponential growth of the hyper-volume as a function of the dimensionality [B1961]. When applied to hyper-dimensional spaces, clustering algorithms suffer from the fact that the size of the data set is too small compared to the volume of the space. For this purpose, dimensionality reduction is very helpful, because practically the number of training data samples remains the same but the volume of the space reduces.

Formally, having an N-dimensional random variable $x = (x_1,\dots,x_N)$, dimensionality reduction tries to find a new representation for it: $y = (y_1,\dots,yP)$, $P<N$, according to a criterion. The criterion is the one that defines in what sense the properties of the original N-dimensional space will modify in the new P-dimensional space. For the purpose of this work, the new representation has one dimension, so $P = 1$.

4.1. Principal Component Analysis

Principal Component Analysis (PCA) [JW1992] is the most used method for dimensionality reduction. It is a linear transformation that keeps the dimensionality of the space. However, the new axes are ordered related to the variance of the data on them. Usually, more information is contained in the features that have bigger variance. In this way, we can assume that the first components (axes) contain the largest amount of information.

For the random variable $x = (x_1,\dots,xN)^T$ let us consider $m_x = E\{x\}$ as its mean value. The covariance matrix is defined as $C_x = \{(x-m_x)(x-m_x)^T\}$. The elements c_{ij} of the covariance matrix C_x correspond to the covariance between variables x_i and x_j, and particularly c_{ii} is the variance of variable x_i. The covariance between two random variables a and b is

$$\operatorname{cov}(a,b) = \frac{\sum_{i=1}^{n}(a_i - \overline{a})(b_i - \overline{b})}{N-1}, \tag{4.1}$$

where N is the number of variables. The covariance matrix is always symmetrical, by definition. If two variables are uncorrelated, the value of their correlation is zero.

The aim of PCA is to create such a transformation in the space so that the new variables are uncorrelated. For the covariance matrix (always a symmetric matrix), an orthogonal basis can be created by finding the corresponding eigenvalues and eigenvectors. Mathematically, the equation that needs to be solved is

$$C_x e_i = \lambda_i e_i, i = 1,\dots,N, \tag{4.2}$$

where e_i are the eigenvectors and λ_i are the corresponding eigenvalues. Let us assume, for simplicity, that the values λ_i of the eigenvalues are distinct. The eigenvalues and eigenvectors can be found by solving the equation

$$\|C_x - \lambda_i I\| = 0, \tag{4.3}$$

where I is the identity matrix, and ||*|| is the determinant operator. In practice, it is preferable to use numerical methods to solve this equation as it will be explained in Section 4.2. The eigenvectors are ordered according to the values of the eigenvalues, and the created orthogonal basis has the largest variations of data on the first direction.

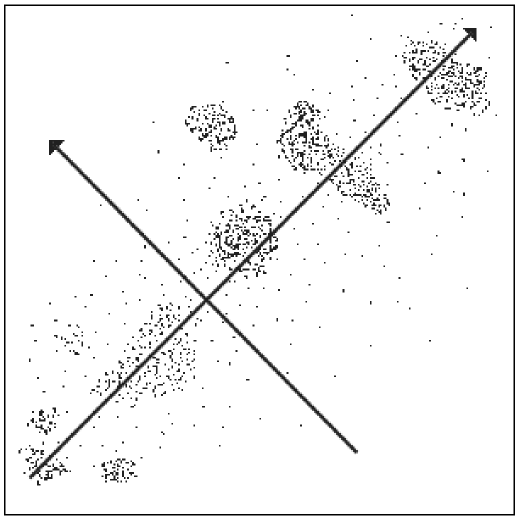

Figure 4.1. The axes corresponding to the eigenvectors.

Zero covariance does not determine the independence of the variables, so although PCA obtains variables with zero covariance, the variables are not necessarily independent.

4.2. Computing Principal Components

For clustering, we only need the first principal component. It corresponds to the largest eigenvalue, and the variance of the data along this direction is maximal. Eigenvalues are the solutions (in λ) or the equation:

$$\det(C \cdot x - \lambda \cdot I) = 0, \tag{4.4}$$

where I is the identity matrix, and C is the covariance matrix. The equation above is equivalent to solving an D-degree polynomial, where D is the number of dimensions in the space. Each value of λ has a corresponding eigenvector. If we determine the values of λ, then the eigenvectors are trivial to obtain.

Generally, the solutions for the equation above are not determined by solving the corresponding polynomial because a small error on a coefficient from the polynomial determines a large error in the solution. Instead, numerical methods are preferred.

There are other methods for solving the equation (4.4), as well. Since for our proposed clustering method we only need the first component, we have chosen the simplest method: the *power method* [D1997]. The algorithm converges only if the largest eigenvalue is strictly larger than the next one:

$$|\lambda_D| > |\lambda_{D-1}|, \tag{4.5}$$

where λ_D and λ_{D-1} are the first two eigenvalues. The procedure is based on the recursion:

$$x^{(k+1)} = C \cdot x^{(k)} / \left| C \cdot x^{(k)} \right|, \tag{4.6}$$

which successively approximates the eigenvector corresponding to the largest eigenvalue. In each iteration, the inequality

$$\left\| x^{(k+1)} - \lambda \cdot x^{(k)} \right\| < \varepsilon \tag{4.7}$$

is tested. If ε is smaller than an imposed maximum value, the eigenvector is found as $x^{(k)}$.

The pseudo-code for the power method is shown in Figure 4.2. The parameters for the method are the matrix *C*, the tolerance error ε and the maximum number of iterations *It_Max*. If the solution is not found within the maximum number of iterations allowed, then the algorithm could not converge quickly enough. Notice that other implementations of principal component are possible [D1997].

```
PROCEDURE Make_Principal_Component

Parameters: C, ε
Output: v, λ
Locals: n := dimension(C), $x^{(k)}$, λ

BEGIN
    choose $x^{(0)}$ in $R^n$, $|x^{(0)}| = 1$;
    FOR k :=0 TO It_Max
        $x^{(k+1)}=C\,x^{(k)}$;
        $f(\lambda) = \min_\lambda \{ f(\lambda) / f(\lambda) = |x^{(k+1)} - \lambda x^{(k)}| \}$ ;
        IF f(λ) < ε
            $v = x^{(k+1)}$;
            $\lambda := \arg\min_\lambda \{ f(\lambda) / f(\lambda) = |x^{(k+1)} - \lambda x^{(k)}| \}$ ;
            return;
        END
    END
    $v = x^{(k+1)}$;
    $\lambda := \arg\min_\lambda \{ f(\lambda) / f(\lambda) = |x^{(k+1)} - \lambda x^{(k)}| \}$ ;
    return ;

END-PROCEDURE
```

Figure 4.2. First principal component computation by power method.

4.3. Other Linear Methods

Principal component analysis is not the only linear method used for dimensionality reduction. For our algorithm, we have used the first principal component as the one-dimensional subspace to map the data. Next, we will describe other linear methods that are currently used in cluster analysis and dimensionality reduction.

4.3.1. Factor Analysis

Factor Analysis (FA) is related to PCA in a sense that both of them are intended for second-order (Gaussian) models. It assumes the following model for data [H1999]:

$$x = As + h\,, \tag{4.8}$$

where x is the D-dimensional observed random variable, s is the unobserved vector of latent variables, A is a constant $D \times M$ matrix and h is the noise. All the variables in s and h are considered Gaussian. It is also assumed that the dimensionality of s is lower that the dimensionality of x, and this is the basis for dimensionality reduction.

Most of the methods for solving the equation (4.8) are derived from PCA transformation. However, there is one important difference: the equation does not define the factors uniquely, but upon a rotation. Therefore, it is conventional to search the rotation of the factors for interesting properties [H1999].

4.3.2. Projection Pursuit

Projection pursuit [F1987, FT1974] is another linear transformation method, which can incorporate information higher than second order. It can be used for non-Gaussian data sets as well. Projection pursuit is defined to be *the search for interesting structured projections*.

Given a *projection index* that defines the interestingness of a direction, this method looks for directions that optimize the index. In case the projection index is defined by maximizing the second-order variance, then the same result as PCA is obtained. The Gaussian and uniform distributions are considered the least interesting, so other distribution functions will be searched for.

This method specifies two things [RMH2004]:

(1) an index I(α) that measures the interestingness of the projection α,
(2) an algorithm that varies the projection direction to find optimal projections.

A common high order projection index is the negative Shannon entropy. It is defined:

$$Q(x) = \int f(x)\log(f(x))dx\,, \tag{4.9}$$

where x is a random variable and $f(x)$ is its probability density function (pdf). Since the Gaussian distribution minimizes this measure, a projection pursuit algorithm will try to maximize it.

4.3.3. Independent Component Analysis

As opposed to the PCA method which searches for orthogonal subspaces to maximize the variance, *Independent Component Analysis* (ICA) looks for linear *independent* projections, but not necessary orthogonal. Considering N variables $x_1,\dots,x_N$, they are mutually independent (in statistical sense) if their density function can be factorized:

$$f(x_1,\dots,x_n) = f(x_1)\cdot\ldots\cdot f(x_n) \tag{4.10}$$

Statistical independence must be clearly distinguished by null correlation (see PCA), which is defined by:

$$E(x_i, x_j) = E(x_i)\cdot E(x_j),\ i \neq j \tag{4.11}$$

The property of statistical independence is stronger that non correlation, and for any function g_1, g_2, if x_i, x_j are independent [P1991]:

$$E(g_1(x_i)g_2(x_j)) = E(g_1(x_i))E(g_2(x_j)) \tag{4.12}$$

There are at least three definitions to linear ICA [H1999], and here is the most general definition [H1999]: ICA of the random vector x consists of finding a linear transformation $s = Wx$ so that the components s_i are as independent as possible, in the sense of maximizing some function $F(s_1,\dots,s_n)$ that measures the statistical independence.

This definition is general but vague, while the other definitions include assumptions of the data. In this work, we introduce some basic notions and the reader can find more detailed information in [H1999].

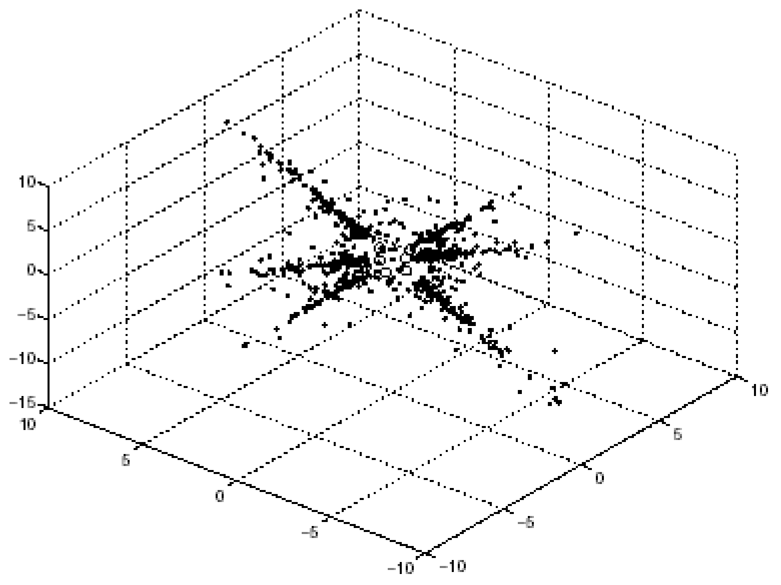

Figure 4.3. Data suitable for ICA learning [TGPL2004].

5. Nonlinear Dimensionality Reduction

Although studied and used intensively in various applications, the linear projection fails when the correlations of the data is highly non-linear. In this case, non-linear methods can better capture the model in the data. However, it is more difficult to find an appropriate projection because the danger of over-fitting the model, which have happened also when we have used the Hamiltonian shortest path in Section 3.2. In literature, non-linear methods are therefore less frequently applied than the linear ones.

The methods that have been presented for linear dimensionality reduction in Section 4 have also non-linear variants, e.g. *non-linear PCA*, and *non-linear ICA*, but methods based on other non-linear transformations exist as well. We will focus especially on methods known as *principal curves*.

5.1. Principal Curves

The principal curve has been introduced as a straightforward generalization of principal components [H1989]. The initial approach on principal curves defines them as smooth one-dimensional curves that pass through the "middle" of multi-dimensional data set in R^D [H1989] (see Figure 5.1). We will refer to this curve as HS principal curve. The HS curve aims at modeling the data that was generated according to the model:

$$x_i = f(\lambda_i) + e_i, \tag{5.1}$$

where λ_i is a *M*-dimensional vector, *f* is a vector of *D* functions and e_i is the noise. As the principal curve is one-dimensional, *M* is equal to 1.

The intuitive definition of the principal curve can be formally stated: Suppose *X* is an *D*-dimensional random vector with continuous probability density *h(x)*, *G* the class of differentiable one-dimensional curves in R^D parameterized by λ and λ belongs to Λ_f for each *f* in *G*. For each λ and for each *f* in *G*, the projection index is defined as:

$$\lambda_f : R^D \to \Lambda_f, \lambda_f(x) = \max_\lambda \{\lambda : |x - f(\lambda)| = \inf_\mu (x - f(\mu))\} \tag{5.2}$$

Principal curves are defined to be those curves of *G* that are *self-consistent*. A curve is self-consistent if:

$$E(X / \lambda_f(X) = \lambda) = f(\lambda), \tag{5.3}$$

for any λ in Λ_f. An informal definition of self-consistency property is that each point on the curve is the average of all points that project over it. The HS curve is defined so that it is not allowed to intersect with itself.

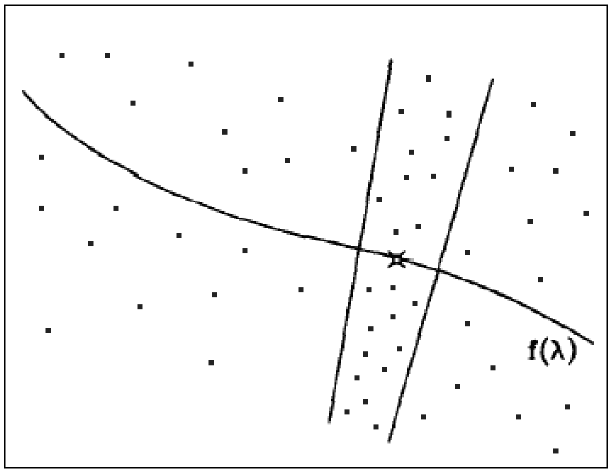

Figure 5.1. Each point on the HS principal curve is the average of the points that project there [H1989].

Next, more approaches will be considered for principal curves. An application for modeling the outlines of ice floes in satellite images that uses principal curves was developed in [BR1992] (see Figure 5.2). Another definition was given for the principal curves, as the curves are closed; the algorithm from HS curve is modified so that, for a fixed level of smoothness, the bias is reduced. This is achieved by estimation of error residuals in the expectation step of the algorithm. This curve will be referred to as the BR curve.

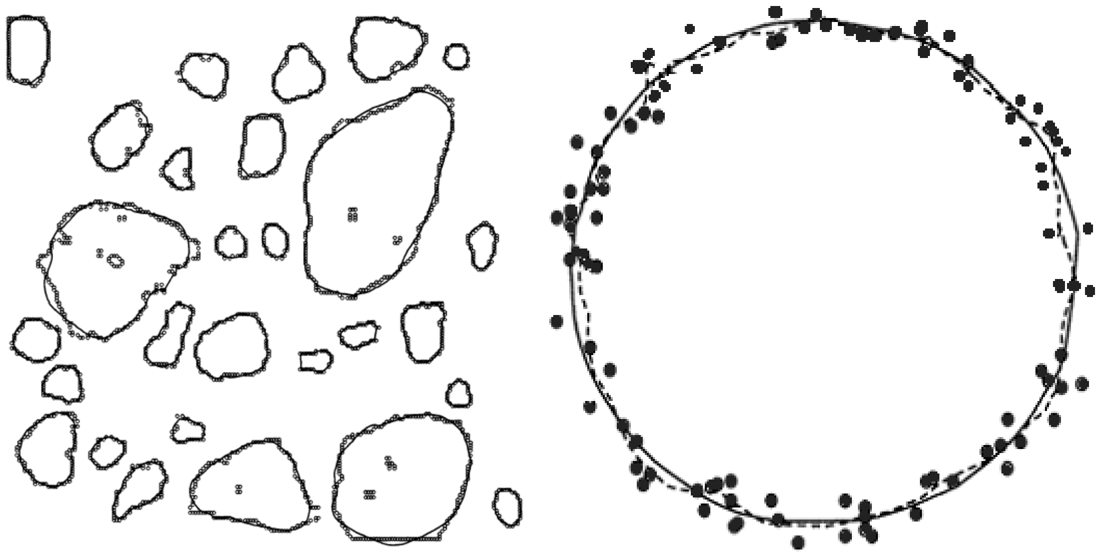

Figure 5.2. On the left, a set of BR curves is shown [BR1992]. On the right, the BR curve (continuous line), and HS curve (dashed line).

In [CG1], an improved variant of the algorithm for principal curves was proposed (hereafter the CG curve). The performance of the BR curve can be improved if the curve is better initialized. The CG principal curve uses the HS curve as initialization and continues with the BR algorithm.

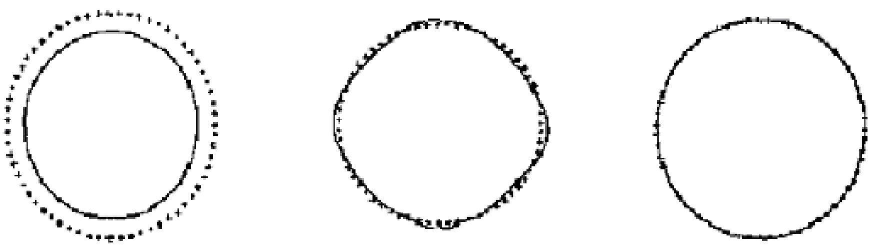

Figure 5.3. Results of HS [H1989] algorithm on left, BR [BR1992] algorithm in the center and HS-BR [CG1] algorithm on right. Image is taken from [CG1].

The principal curve is used for feature extraction and *Principal Curve Classifier* is introduced. The Principal Curve Classifier builds a principal curve template for each class. The *Principal Curve Feature Extractor* reduces the dimensionality of the space replacing *K* linear dimensions with one non-linear dimension represented by the principal curve. The space is split in subspaces based on principal components, and from each subspace a principal curve is computed: the first curve is computed from the first *K* components subspace, the second curve from the next *K* components and so on. Thus, for one point, each *K* linear coordinates can be stored in one value, representing the value of the projection on the principal curve.

Principal curves are defined in [KKLZ2000] as continuous curves of a given maximal length, which minimize the expected squared distance to the points of the space randomly chosen according to a given distribution. This curve will be referred to as the *principal curve with length constraint*. A practical method for the construction of the curve as a heuristic iterative algorithm that approximates it by a polygonal line is given in the referred paper. Because this variant of curve is used in our method for clustering, it will be detailed later in this section.

An incremental method to find principal curves is introduced in [VVK2002] (see Figure 5.4). Line segments are fitted to data using local linear models and then the segments are connected to form a polygonal line. The algorithm is similar to K-means iterative algorithm, but the centroids are generalized to segments. The set of unconnected segments is then transformed into a polygonal line.

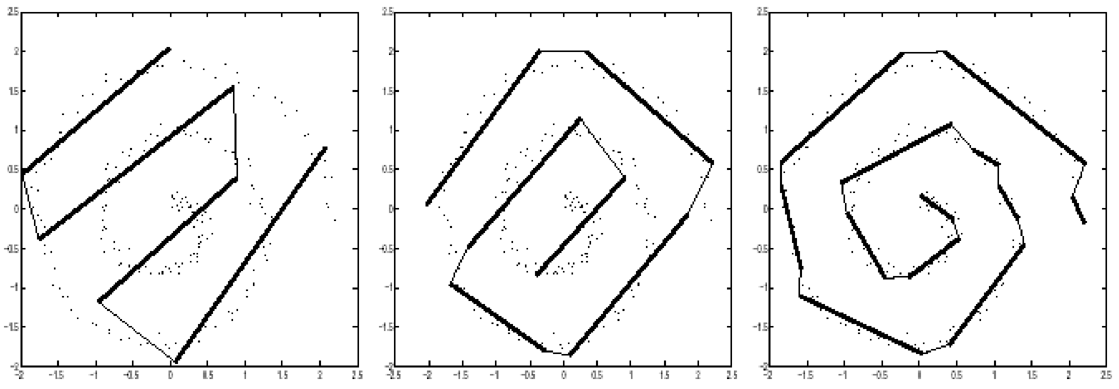

Figure 5.4. Polygonal curves [VVK2002] having different number of segments.

Instead of considering the total length of the curve as in [KKLZ2000], the method in [SK2002] constrains the sum of the angles along the curve (*the total turn*). Actually, the total turn and the total length of the curve are closely related. As it has been shown in [KKLZ2000] and will be discussed later in Section 5.3, minimizing the turn indirectly implies modification in the length of the curve. The authors also present a theoretical algorithm for learning such curves.

5.2. Other Methods

Multi-dimensional scaling (MDS) [YH1987] finds a low dimensional representation of a higher dimensional data set preserving the proximity relations in the original set. *Classical MDS* considers the values of the (dis)similarity distances while *ordinal MDS* just the relation among them. Usually, the method is used for visualization and the dimension of the reduced space is two or three.

Other methods try to find a continuous map to transform a high-dimensional space into a lattice space of a fixed number of dimensions. These methods are sometimes referred to as *topologically continuous maps*; this term is used rather than *self-organizing-maps* (SOM) [F2002], because the latter one might be confused with the particular method named *Kohonen's self-organizing-maps* (KSOM).

Kohonen's self-organizing-maps [K2001] learn in an unsupervised way to map a space to a (usually) two-dimensional lattice. The method can be easily extended to a map of different dimensionality, also one-dimensional. The map can be seen as a type of neural network: a (two-dimensional) array of neurons. During the training, the activation of one neuron determines modifications not only to itself but also to its neighbors. This way, similar vectors will activate neurons in the same neighborhood. The method is usually used for visualization.

Density networks [M1995] apply *Bayesian learning* to a data set with known probability distribution parameters to model data from the latent variables. A special type with density network based on a constrained mixture of Gaussians model, called *Generative Topographic Mapping* (GTM) [BSW1998], uses an *Expectation-Maximization* (EM) algorithm for parameter estimation. The method provides an alternative to KSOM and overcomes its limitations, namely no criteria for optimization and no proof of convergence.

5.3. Principal Curve with Length Constraint

The main property of the principal curves with length constraint [KKLZ2000] is that it minimizes the expected squared distance to the curve $\Delta(f)$, given the maximal length of the curve. For any random vector X with a density function and for any $\varepsilon>0$, there exists a smooth curve so that $\Delta(f)<\varepsilon$. If the density function covers a space with non-zero hyper-volume, as the value of ε decreases, the length of the curve increases unlimited towards infinite. On the other hand, if the distribution of f is concentrated on a polygonal line, the polygonal line itself minimizes $\Delta(f)$ to 0, but the polygonal line is not smooth. Due to the two reasons above, two important consequences are found in the new definition of the curve: the curve is not necessarily smooth and the length is constrained.

Definition: *A curve f^* is called a principal curve of length L for X if f^* minimizes $\Delta(f)$ over all curves of length less than or equal to L.*

The authors prove the existence of this curve for any X that has finite second moments. For the practical case when the distribution X is not known but only as a data set representing values of the random vector X, the authors propose a learning algorithm. They prove first that if the points represent a closed convex set, the curve exists and the expected squared loss of the empirically optimal polygonal line with K segments ($K\sim N^{1/3}$) and maximal length L converges, as $N \to \infty$, to the squared loss of the principal curve L at a rate $O(N^{-1/3})$.

Unfortunately, the authors could not construct the curve according to their learning scheme, but they propose a sub-optimal *polygonal line algorithm*. This method is based on theoretical results obtained above, but the algorithm is mainly heuristic. The main difference to the theoretical algorithm is that the length L is not provided as a parameter but it is optimized by the algorithm.

The algorithm is iterative, at each step a new polygonal approximation with larger number of segments is found. It is initialized with a segment on the first principal component as the first approximation for the curve. Every iteration includes a projection step, and the segment with the largest number of projected data points is split into two equal parts. After splitting, an optimization loop that minimizes a function of error and angle is performed for each vertex (see equation 5.4). The algorithm stops when the number of segments exceeds a given limit (which will be detailed later, see equation 5.6). The pseudo-code of the algorithm is shown in Figure 5.5, and a visual example is presented in Figure 5.6.

```
PROCEDURE Make_Principal_Curve

Parameters: X
Output: Principal_Curve
Locals: k

BEGIN

    k := 1;
    Initialize (Principal_Curve, X);

    WHILE NOT Stopping_condition (Principal_Curve, X, k)
        k := k + 1;
        Find_segment_to_split (Principal_Curve, X);
        Split_segment (Principal_Curve, X);

        WHILE NOT convergence
            Project (X, Principal_Curve);
            Vertex_optimization (X, Principal_Curve);
        END
    END

END-PROCEDURE
```

Figure 5.5. Principal curve [KKLZ2000] learning algorithm.

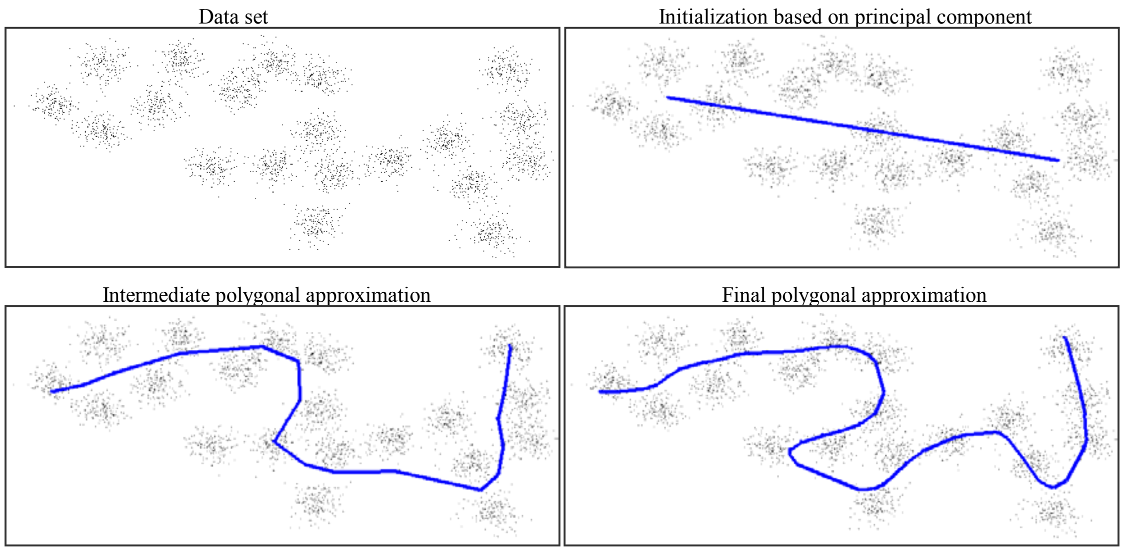

Figure 5.6. The curve is constructed iteratively; it is initialized with the first principal component and, at each iteration, the number of segments is increased by 1.

For the projection step, each point projects on the closest point of the curve (see Figure 5.7). For finding the projection point, first of all the closest segment is found and then the point is projected on the support line of that segment. If the projection point is outside the segment, the projection point on the curve is chosen as the closest extremity of the segment. This results in a big amount of different points in the space that project on the boundaries of the segments.

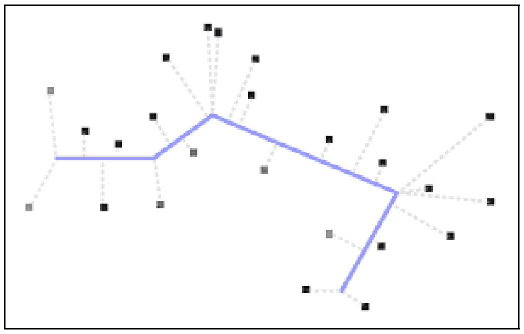

Figure 5.7. Data set projection over the polygonal line.

The most complex phase in the algorithm is the vertex optimization process. At the optimization process, the length of the curve is indirectly controlled. Each vertex position is separately optimized, keeping all the other vertexes fixed. The optimization step minimizes a function depending on both squared error and curvature, and it is defined as:

$$G_N(v) = \frac{1}{N}\Delta_N(v) + \lambda\frac{1}{k+1}P(v), \tag{5.4}$$

where v is the vertex to optimize, N is the number of points, Δ_N is the local error, λ is the smoothing coefficient, k is the current number of segments and *P(v)* is the penalty function. The local error Δ_N considers the distortion of the points that project on the vertex and on the adjacent segments; the penalty function *P(v)* depends on the angle between the two segments and the other two angles the segments are adjacent to (more details in [KKLZ2000]). Thus, the function *G*() minimizes is a *Lagrangian* formulation combining the local distortion of the curve and the local curvature. Given the definition of the curve, it would appear more normal to introduce the length instead of the curvature. The reason why the formulation considers the curvature is that it confers more stability in practice.

The factor λ (smoothing factor) determines the amount of smoothing. One can easily notice that if λ is 0, only the error is minimized. In this case, if the stopping criterion would be disabled, the curve will become a path in the graph and the total error would be 0. On the other hand, if λ was very large the curve would be very smooth (at the extreme, just one line), but the error would be large. The formulation of λ, heuristically found is given as:

$$\lambda = \lambda_c k N^{-\frac{1}{3}} \Delta_N(f_{k,N})^{\frac{1}{2}} r^{-1}, \tag{5.5}$$

where k is the current number of segments, N the number of points in the set, $\Delta_N(f_{k,N})$ the total distortion of the set, r the radius of the data set and λ_c a constant coefficient (penalty coefficient).

Another aspect of the algorithm consists in the stopping criterion for the number of segments. This was also determined heuristically, and the algorithm stops when k exceeds:

$$K = \beta N^{-\frac{1}{3}} \Delta_N (f_{k,N})^{-\frac{1}{2}} r \tag{5.6}$$

Increasing the number of segments does not modify further on the shape of the polygonal line because as the number of segments increases, the curvature penalty coefficient also modifies. This way, although more segments are split, the new vertexes hardly modify their positions (see Figure 5.8).

Regarding the constant λ_c parameter that controls the smoothing coefficient, modifications determine significant changes in the shape of the curve. Its value was determined by experiments to 0.13. A bigger value gives a smoother curve (see Figure 5.8).

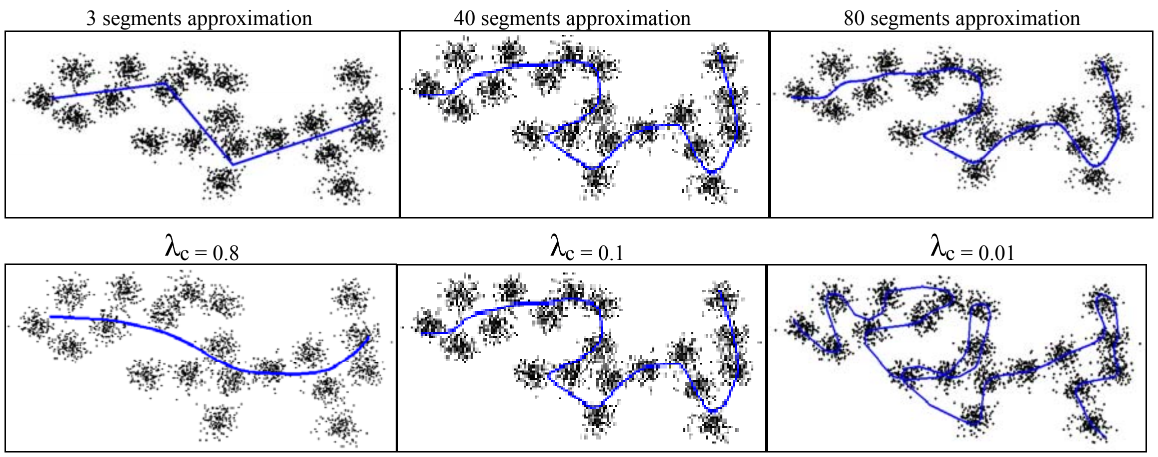

Figure 5.8. The images above show the result when parameterized by the number of segments, and the images below when parameterized by the smoothing factor.

6. Clustering Based on Principal Curve

As the principal axis based clustering did not yield very good results because the principal axis is too rigid to model the data, we propose to use the principal curve. The principal curve is a projection space that can capture the non-linearity from the data and can preserve better the adjacency from the space in the generated order.

Among the many definitions of the principal curves, we have chosen the principal curve with length constraint [KKLZ2000]. The principal curve as it is defined in [KKLZ2000], minimizes the distortion of the points to the curve. This assures that the curve is as close as possible to the points, and the adjacency is better preserved. This way, the biggest drawback for the principal axis clustering algorithm is overcome.

The clustering algorithm performs the same main steps as the principal axis clustering algorithm. First the curve is constructed, and second the data set is projected and ordered. The same dynamic programming algorithm for order constrained clustering is applied. The pseudo-code for the principal curve based clustering is shown in Figure 6.1 and a visual example is shown in Figure 6.2. The algorithm will also be referred to as PCU.

```
PROCEDURE PCU

Parameters: X, K
Output: Q = {$q_1$,…,$q_K$}
Locals: Principal_Curve

BEGIN

    Principal_Curve := Make_Principal_Curve (X);
    Order_Dataset (X, N, Principal_Curve);
    Q := DP_Cluster (X, K, N);

END-PROCEDURE
```

Figure 6.1. Clustering on principal curve (PCU).

After the algorithm is run and the code vectors are obtained, the constraint of the sequence order is removed and the optimal partitioning (Voronoi cells) is formed. Compared to the principal axis based algorithm, the results are much better. The algorithm can be applied to much more complicated data sets and gives good results (see Figure 6.3).

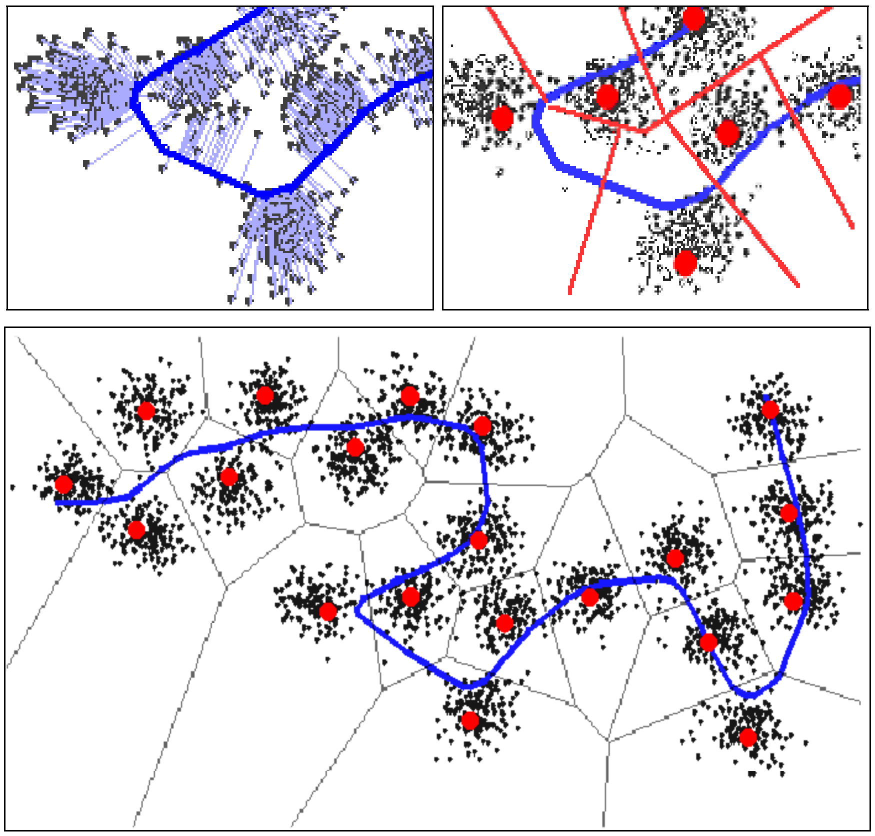

Figure 6.2. Projection step (up left), partitioning step (up right), and Voronoi partitions (bottom).

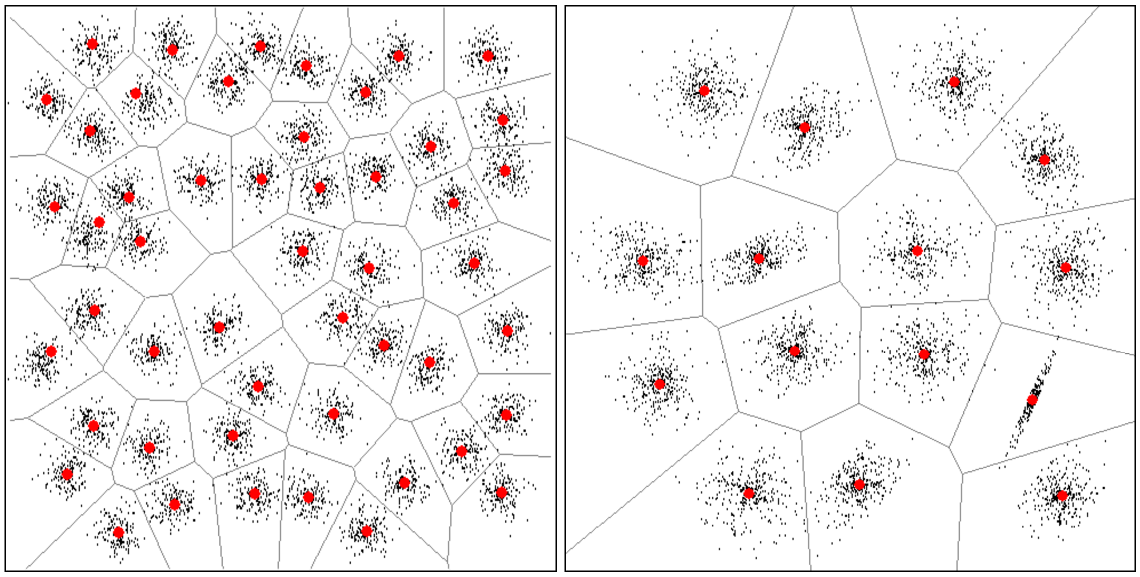

Figure 6.3. Results of the algorithm for two different data sets.

6.1. Choice of the Parameters

Except for the number of clusters, the other parameters for the clustering algorithm are used for the curve construction. One parameter influences the stopping criterion (the number of segments) and the other one influences the curvature. While the algorithm works well for different types of data sets, the quality of the results varies significantly with the parameters of the curve.

For our purpose, the heuristic value proposed by [KKLZ2000] for the stopping criterion is good enough. If one considers less run-time for the algorithm, the number of segments of the curve can be even reduced. However, if the number of segments is too small, clusters will project on the same regions of the curve, overlapping and thus worsening the quality of the codebook.

The penalty coefficient has more influence over clustering (see Figure 6.4). It is desirable that the points from one cluster project on only one region of the curve. Longer curves make possible for points from one cluster to project on several regions of the curve. On the other hand, shorter curves allow different clusters to project on overlapping regions. This coefficient must therefore be tuned depending on the data.

We have tested the influence of the penalty coefficient for different types of data sets, real and artificial, presenting or not clear evidence of clusters.

For data sets that present distinct clusters, the mean squared error as function of penalty coefficient has a minimum. The optimum is obtained for lower values of penalty coefficient than those found in [KKLZ2000]. The curve used in our proposed algorithm should be longer than the original curve that is computed just to summarize the data. As the number of clusters increases and the data set is more complicated, the coefficient should be even smaller (and correspondingly, the curve longer). Good values for penalty coefficient have been found in range 0.01 to 0.08 (see Figure 6.4). Still, if the data set is not very complicated, the correct partitions can be found for larger values of the penalty coefficient. If the algorithm is used for quantization and the dataset does not reveal clear clusters, even values below 0.01 can be used for penalty coefficient. If one knows the characteristics of the data set, the coefficient can be set accordingly. Else, an algorithm that optimizes the value of the coefficient for optimum error can be applied.

6.2. Improvement by K-means

The resulted codebook can be further on optimized by iterating the K-means algorithm. The amount of improvement varies with the penalty coefficient used for curve. If the curve assures a good mean squared error for the codebook, the codebook is also very close to the local optimum and a small number of iterations is needed (see Figure 6.4).

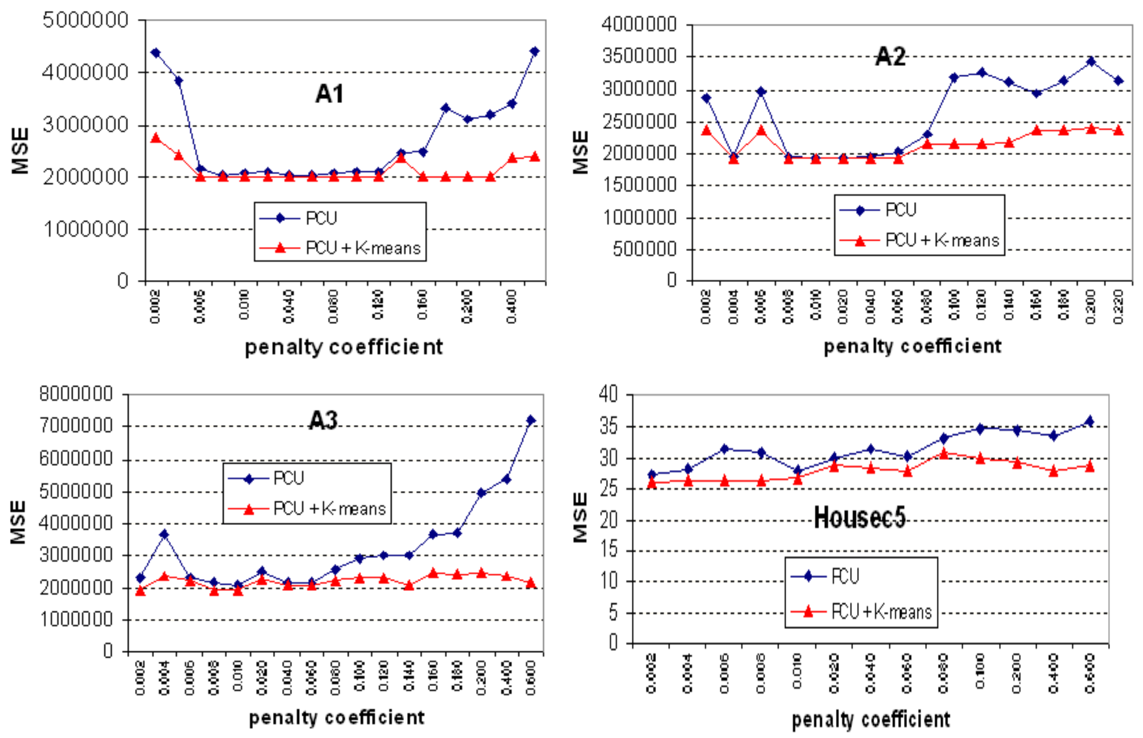

Figure 6.4. Improvements of error after K-means tuning, depending on the penalty coefficient.

6.3. Hierarchical Clustering on Principal Curve

Clustering based on principal curve performs well if the data can be modeled by a curve. Worse results are observed for more complex data sets, when the penalty parameter for the curve needs to be fixed in a narrow range. Moreover, many data sets cannot be meaningfully modeled by the principal curve (see Figure 6.5), so that the algorithm presented in the previous section cannot obtain good results.

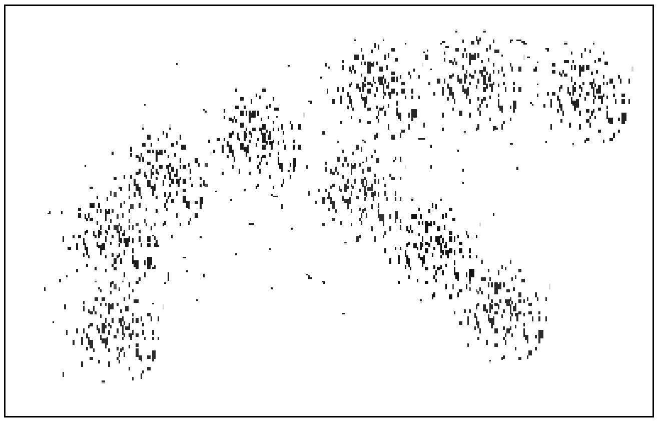

Figure 6.5. The data cannot be modeled by a curve.

A way to overcome this problem is to split the data space and to apply the algorithm hierarchically. We split the data set attempting to create subsets that can be modeled better by principal curves. We cluster each subset using the principal curve based algorithm, and then we create the codebook of the whole data set using the partial codebooks.

A pre-clustering algorithm followed by *minimal spanning tree* (MST) computation of the result can give a better model of the data. The MST of the codebook is simple to construct and has a more regular form than the MST for the whole data set. The MST can be split into branches and this way, the whole data set can be partitioned, as in Figure 6.6. The goal is to create a subset for each branch that can be modeled easily by the principal curve. For this, we will consider the next two rules:

(1) As long as a node of the tree has only one descendent, both of them can belong to the same branch.

(2) If one node has more descendents, they will be considered in different branches.

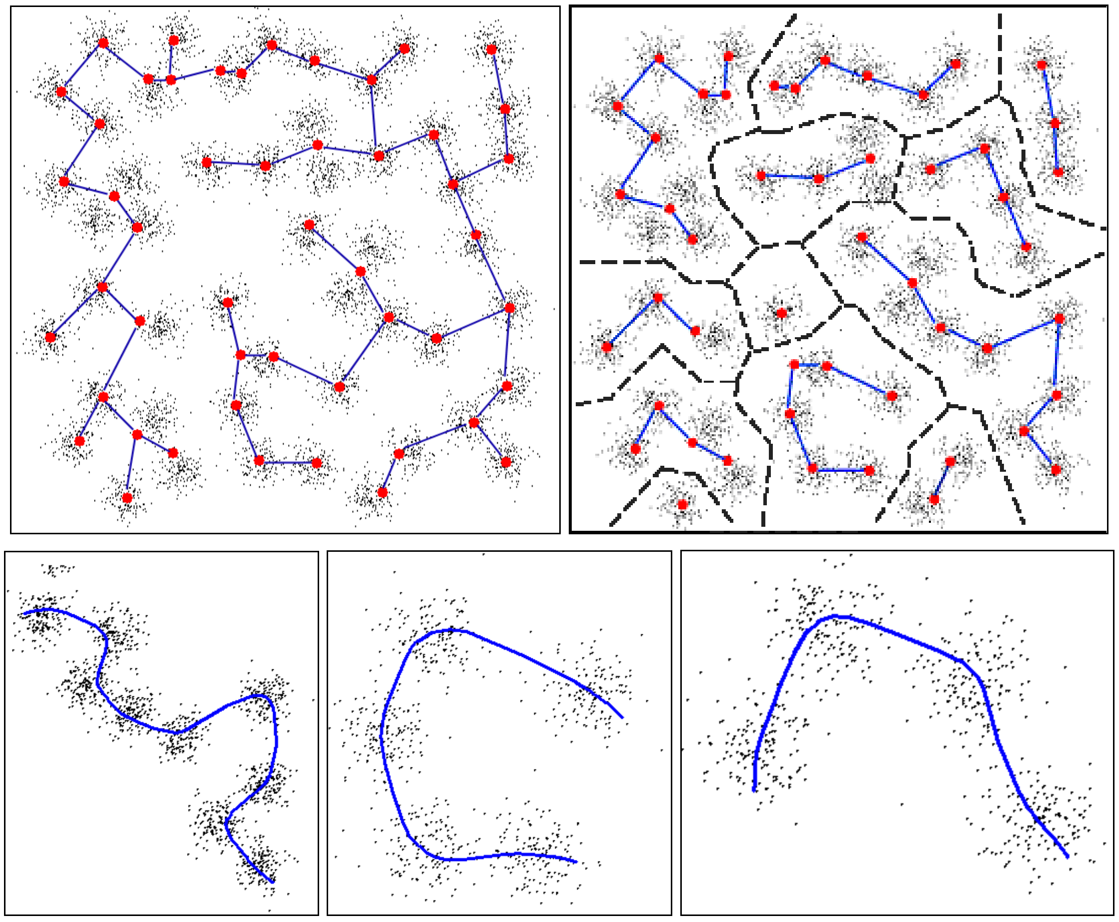

Figure 6.6. The data set and the MST formed from the codebook (up left), the corresponding subsets (up right) and the principal curves for some of the subsets (bottom).

Initial clustering is performed and the data set is split according to the MST of the initial codebook. The tree is traced in a depth-first-search (DFS) manner. The root belongs to the first branch. If a node has only one descendent, both the node and its descendent belong to the same branch. If a node has more descendents, only one of them belongs to the same branch as the parent node. For the rest of the descendents, new branches are created. Thus, all the nodes of the tree correspond to some branches. On the other hand, each node corresponds to a code vector and to a cluster. The data points that are contained in the clusters corresponding to the nodes from one branch are grouped in one subset. The algorithm is detailed in Figure 6.7, and an example is shown in Figure 6.6.

```
PROCEDURE Split_Data_Set

Parameters: X, MST
Output: X_set= {$X_1$,…,$X_M$}
Locals: current_node

BEGIN

    current_node := Get_Root (MST);
    i := 1;
    Make_New_Subset ($X_i$);
    Add (X_set, $X_i$);
    Add_To_Subset ($X_i$, current_node);
    Mark current_node as visited;

    WHILE MST has unvisited nodes

        IF current_node has one descendent
            current_node := Get_Descendent (MST, current_node);
            Add_To_Subset ($X_i$, current_node);
            Mark current_node as visited;
        END

        IF current_node has more descendents
            current_node = Get_Next_Descendent (MST, current_node);

            IF current_node is the first descendent
                Add_To_Subset ($X_i$, current_node);
                Mark current_node as visited;
            ELSE
                i := i + 1;
                Make_New_Subset ($X_i$);
                Add (X_set, $X_i$);
                Add_To_Subset ($X_i$, current_node);
                Mark current_node as visited;
            END
        END

        IF current_node has no descendents
            current_node := Get_Next_Node_DFS (MST);
        END
    END

END-PROCEDURE
```

Figure 6.7. The algorithm that splits the data set based on the MST.

Each subset is clustered by the principal curve algorithm, and the results are combined to form the codebook for the whole data set. A problem is the choice of the codebook size for each part of the data set. The principal curve clustering algorithm, as it has been defined, solves the K-clustering problem. After the data set has been split, the codebook size for each subset has to be decided. We propose to set the codebook size for one subset to the number of nodes in the corresponding branch. The general algorithm is presented in Figure 6.8, and will be referred as *hierarchical principal curve clustering algorithm*, or HPCU.

```
PROCEDURE HPCU

Parameters: X, K
Output: Q = {q_1,...,q_K}
Locals: Initial_Q, MST, X_temp, X_set, Q_temp, K_temp

BEGIN
    Q := {};
    Initial_Q := Precluster (X, K);
    MST := Build_MST(Initial_Q);

    X_set := Split_Data_Set (X, MST);FOR EACH subset X_temp of X
    contained in X_set

        K_temp := Get_CB_Size (X_temp, Initial_Q);
        Q_temp := Principal_Curve_Cluster(X_temp, K_temp);
        Add (Q, Q_temp);
    END

END-PROCEDURE
```

Figure 6.8. The basic hierarchical clustering using principal curves algorithm.

6.4. Optimal Combination of Codebooks

Considering that the curve will better fit the smaller data subsets, and thus the clustering will be better, the hierarchical algorithm proposed before will improve the initial solution. The drawback of the solution consists in the fact that the solution is improved only locally. If the first clustering algorithm did not fit well the number of clusters to each subset, although the error will be improved by the hierarchical algorithm, it might be far from the optimal solution. We propose a way to overcome this problem, by considering multiple codebook sizes for each subset and combine them to get the optimal result.

Instead of considering just the optimization of the codebook of the local subsets, we will consider also the optimization of the codebook sizes. Our solution for this problem is to compute more codebooks for each data subset (see Figure 6.9) and combine them to keep the codebook size of the whole data set constant, but minimize the total distortion. Solving the K-clustering problem by using the principal curve as it was presented in this work solves actually all the clustering problems with the number of clusters smaller than K, without extra computational cost. Thus, the new approach does not introduce considerable computational delay.

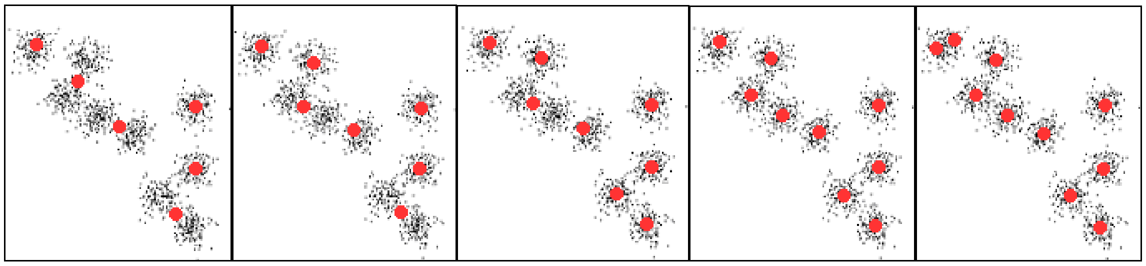

Figure 6.9. Codebooks of different sizes constructed for the same data set.

Let us consider the initial data set X and the number of clusters K. After a first cluster and split of the data set, the sub-problem will consist of clustering M subsets X_1, X_2, …,X_M. We have to find the codebooks Q_1, Q_2, …, Q_M, having the corresponding codebook sizes K_1, K_2, … , K_M and distortions D_1, D_2, … , D_M, so that the total distortion D is minimum and the total codebook size is exactly K:

$$D = D_1 + \ldots + D_M \qquad (6.1)$$

$$K = K_1 + \ldots + K_M \qquad (6.2)$$

This is a classical optimization problem and the solution can be found by solving the recursion:

$$D(m,k) = \min_{k_m}(D(m-1,k-k_m) + D_m(k_m)), \qquad (6.3)$$

where $D(m, k)$ is the distortion of the first m data subsets having together the codebook size k, $D_m(k_m)$ is the distortion of one codebook of size k_m for the data subset X_m. The optimal distortion for all the subsets X_i of X is $D(M,K)$ and can be found using the dynamic programming technique to solve the recursion (6.3).

Suppose that the initial codebook estimation for X finds the codebooks Q_1', Q_2', …, Q_M' with codebook sizes K_1', K_2', … , K_M'. We consider that the optimal allocation of codebook sizes might not be far from the initial allocation, so there is no need to search the whole solution space. Let us consider a limit ε, and we will search for the solution within the bounds of the formula (6.4).

$$\varepsilon \geq \max_i \left|K_i' - K_i\right| \qquad (6.4)$$

We search for codebooks of size from K_i' - ε to K_i' + ε for each subset X_i. Computation of all this codebooks is easily done by solving (K_i'+ε)-clustering problem by dynamic programming. Optimal re-combination of partial codebooks to obtain the codebook for all data set will minimize the total distortion, and the sum of the codebook sizes will be constrained to K. The algorithm is presented in Figure 6.10. The codebook re-combination algorithm receives as parameters a matrix of distortions, containing for each of the M subsets the distortions for different codebook sizes.

```
PROCEDURE **DP_Combine**
Parameters: M, ε, K, Q_matrix [M, 2*ε], Dist_matrix [M, 2*ε], Local_K [M];
Output: Q = {$q_1$,…,$q_K$};
Locals: D[M, K], TB[M, K];

BEGIN
    FOR j := max(1, Local_K[1] – ε) TO min(K, Local_K[1] + ε)
        D[1, j] = Dist_matrix[1 , j+1-max(1, Local_K[1] – ε) ];
    END
    FOR i := 2 TO M
        FOR j := 1 TO K
            D[i , i] := min { t = 1 : j-1, D[i-1,t] exists, ε ≥ j-t |
                            D[i-1,t] + Dist_matrix[i , j-t]};
            TB[i , j] := arg min { t = 1 : j-1, D[i-1,t] exists, ε ≥ j-t |
                            D[i-1,t] + Dist_matrix[i , j-t]};
        END
    END
    limit := M;
    FOR i:=M TO 2
        add (Q, Q_matrix [i, limit – TB[i, limit]);
        limit := TB[i,limit];
    END
    Add (Q, Q_matrix [1, limit] );
END-PROCEDURE
```

Figure 6.10. DP optimal combination of codebooks.

To the basic hierarchical algorithms presented in Figure 6.8, we will add a new step that optimally combines the codebooks by the algorithm in Figure 6.10. For simplicity, we will further refer to the new algorithm as hierarchical clustering using principal curves (HPCU). The pseudo-code of the new algorithm is presented in Figure 6.11. For clarity, the pseudo-code of the HPCU algorithm does not present in detail all the parameters to the function DP_combine.

```
PROCEDURE HPCU

Parameters: X, K, ε
Output: Q = {q_1,…,q_K}
Locals: Initial_Q, MST, X_temp, X_set, Q_set, Q_temp, K_temp

BEGIN
        Q := {};
        Initial_Q := Precluster (X, K);
        MST := Build_MST(Initial_Q);
        X_set := Split_Data_Set (X, MST);

        FOR EACH subset X_temp of X contained in X_set
                K_temp := Get_CB_Size (X_temp, Initial_Q);

                FOR i := max (0, K_temp - ε) TO min (K, K_temp + ε)
                        Q_temp := Principal_Curve_Cluster(X_temp, K_temp);
                        Add (Q_set, Q_temp);
                END

                DP_Combine (Q_set, K)
        END

END-PROCEDURE
```

Figure 6.11. The optimized hierarchical clustering using principal curves (HPCU).

6.5. Complexity of the Method

The basic principal curve clustering algorithm consists of three parts. In the first part, the principal curve is built. Each iteration contains a projection step that has the complexity of O(SN), S being the current number of segments of the iteration and N the total number of points of the data set. Each step, the number of segments is increased by one and in the end, the complexity becomes O(S^2N).The stopping criterion which is used considers that the final number of segments is proportional to the cube root of N; the final complexity becomes O($N^{5/3}$). The computational complexity of the principal curve construction algorithm can be improved by adding more vertices at one step or by considering a smaller number of segments.

Once the final curve is obtained and the data points are projected on it, they are sorted. The sorting procedure takes O($N\log N$) time and the dynamic programming approach takes O(KN^2), K being the codebook dimension. This part of the algorithm is similar to the principal axis based clustering.

The most computational demanding part is the dynamic programming procedure for order constrained clustering which gives the overall algorithm complexity as O(KN^2).

The hierarchical principal curve clustering algorithm involves four major steps:

(1) pre-clustering to K codebook size,
(2) construction of the MST for codebook and partition of the space,
(3) clustering along principal curve on each subset,
(4) optimal construction of the codebook of dimension K for the whole set.

For the first step, we have chosen to apply the principal curve clustering algorithm. However, any other clustering algorithm can be applied. In our case, the complexity of this first part is O(KN^2).

There are more methods to construct the MST. We have used Prim's algorithm. It is a greedy algorithm with the complexity O(K^3), K being the number of clusters after the first quantization.

Suppose the number of subsets is M, and each subset has N_i elements, $N_1 + ... + N_M = N$. One can easily notice that the time complexity for solving all M clustering problems is lower than the time complexity for solving the problem for the whole subset. For the dynamic programming optimal combination of codebooks, the complexity does not depend on N as it is O($KM\varepsilon$).The parameter ε should be anyway lower than K, so that the upper bound for this part could be considered O(MK^2).

The overall complexity depends on the clustering method used for pre-clustering. In this work, we have used the principal curve based clustering but we did not restrict the hierarchical principal curve algorithm to this. Still, excepting the pre-clustering part, the complexity of the other three steps is upper bounded by O(KN^2).

7. Estimating the Number of Clusters

A very important, yet mostly unsolved problem in clustering analysis consists in determining the number of clusters. Part of the difficulty of this problem is the fact that not all data sets present clusters, and many times the partitions are very subjective. Numerous approaches have been considered in this sense and a set of references about them can be found in [H1996].

The parametric methods assume models for the clusters (e.g. distribution type), so they require some assumptions; however, they can give results for a large class of problems. On the other hand, the non-parametric models have usually been developed for specific applications.

Clusters are regions with higher density of data points, separated by areas of lower density. One can estimate the local density around each data point and form the clusters based on this indication. The density around one point can be computed using the distances among the data points in the neighborhood. The *K-nearest-neighbor* (KNN) graph is a data structure that can be used for this purpose. However, if the local neighborhood is too small, the density structure can be irregular and clusters unidentified. On the other hand, if the local neighborhood is too large, clusters can be merged. The problem arises when the size of the local neighborhood is chosen. Good results can be achieved if the size of the local neighborhood is adaptive.

Another method for computing the number of clusters is based on the *kernel method.* Having the *probability density function* (pdf) of the random variable, we could try to find clusters in the regions where the random variable is more probable. Since for the most data sets the real pdf is not known, it has to be estimated.

7.1. Kernel Method

The kernel method [CT2000] is a way to estimate the probability density function from a series of observations of a random vector. Considering the random vector X, known by a series of N observations x_{i}, the Kernel Density Estimation $\widehat{f}(x)$ for the density function $f(x)$ in the point x is given by:

$$\widehat{f}(x) = \frac{1}{hN}\sum_{i=1}^{N} K(\frac{x - x_i}{h}) = \frac{1}{N}\sum_{i=1}^{N} K_h(x - x_i)\,, \tag{7.1}$$

where $K()$ denotes the kernel function and h the bandwidth. There are more possibilities to choose the kernel function: uniform, Epanechnikov, Gaussian.

$$Epanechnikov: \frac{3}{4}(1 - u^2), I(|u| \leq 1)\,, \tag{7.2}$$

$$Quartic: \frac{15}{16}(1 - u^2)^2, I(|u| \leq 1)\,, \tag{7.3}$$

$$Triangular: (1 - |u|), I(|u| \leq 1)\,, \tag{7.4}$$

$$Gauss: (2 \cdot \pi)^{-\frac{1}{2}} e^{-\frac{u^2}{2}}\,, \tag{7.5}$$

$$Uniform: \frac{1}{2}, I(|u| \leq 1)\,. \tag{7.6}$$

From the two parameters required by the kernel method, not necessarily the kernel function *K*() is important, but the bandwidth parameter *h*. The theoretical support for this assumption has been given in [MN1998], by observing that the kernels can be rescaled, so that the difference between the two density estimates becomes negligible.

The bandwidth *h*, also called the *smoothing parameter*, controls the degree of smoothing. For choosing *h*, the same problem appears as in the case of computing the local density using the KNN graph (and choosing the dimension of local neighborhood): a small value will generate functions that exhibit numerous peaks and will look very noisy. A large value will wipe out the details of the estimate. The solution is again choosing an adaptive bandwidth.

There are two major categories of kernel estimators: *balloon* estimators and *sample point* estimators [S1994]. For balloon estimators, a different fixed bandwidth is selected for each estimation point *x*. The estimate at *x* is the average of identical kernel functions, each centered at each data point:

$$\widehat{f}_h(x) = \frac{1}{Nh(x)} \sum_{i=1}^{N} K\left(\frac{x - x_i}{h(x)}\right) \tag{7.7}$$

The sample point estimator uses different kernel bandwidths for each data point and averages them in the estimation points:

$$\widehat{f}_h(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h(x_i)} K\left(\frac{x - x_i}{h(x_i)}\right) \tag{7.8}$$

The balloon estimators are easy to use; their biggest drawback is that the obtained pdf usually fails to integrate to one (a basic property of the density function). Sample point estimators do not have this problem, but they suffer from a phenomenon called *non-locality* [TS1992]: the estimate at a point can be strongly affected by data that is far away.

The bandwidth is typically selected on the basis of *integrated squared error* or *mean integrated squared error*:

$$ISE(f_h) = \int \{f_h(x) - f(x)\}^2 dx \tag{7.9}$$

$$MISE(f_h) = \int E\{f_h(x) - f(x)\}^2 dx \tag{7.10}$$

In the equations 7.9 and 7.10, f_h() is the estimated density function and *f*() is the real function. Unfortunately, the function to estimate *f*() is unknown. Silverman's rule of thumb [S1986] considers *f*() as a normal distribution, and gives the optimal bandwidth for the Gaussian kernel bandwidth:

$$h = \left(\frac{4}{D+2}\right)^{\frac{1}{D+4}} \cdot N^{-\frac{1}{D+4}} \cdot \sigma\,, \tag{7.11}$$

where D is the dimensionality of the space and σ the standard deviation. As the distribution of data is further from normal distribution, the equation (7.11) gives worse results for the pdf estimate. Adaptive bandwidth selection methods that give local values for bandwidths have been developed.

We assume that the projections of the clusters on the principal curve have normal distribution. This is a natural assumption since it has been observed that the distribution of the clusters on subspaces tends to be normal. We have developed an iterative algorithm that uses Silverman's rule of thumb to estimate the probability density function of projected data set. At each iteration, an approximation for the pdf is computed. The first pdf estimation uses the bandwidth given by the rule (7.11) for the whole data set. For the next iteration, the space is partitioned according to the pdf: for each local maximum of the pdf we consider a partition. Each partition is approximated as Gaussian, and the rule of thumb is applied again. For the data set, there will be different bandwidths depending on the partition each point belongs. The global density function is estimated conform to new bandwidths. The process is iterated until it converges (the pdf does not change from one iteration to the next one).

We do not theoretically prove the convergence of the method but experiments on clustered data sets showed that the method converges. We do not restrict the use of this algorithm to estimate the bandwidths as we just show that the principal curve can be successfully used to determine the number of clusters.

For the univariate random variable, the algorithm is easy to implement and yields good results. Considering X a scalar random variable, Figure 7.1 presents the pseudo-code for clustering based on pdf estimation over principal curve.

```
PROCEDURE PDF_Partition

Parameters: X
Output: K, P = {P_1,…, P_K}
Locals: pdf, H, local_h, local_partition

BEGIN
    local_h := Silverman (X);
    FOR i := 1 TO |X|
        H[ i ] = local_h ;
    END
    pdf := Compute_pdf (X , H);
    WHILE PDF does not modify
        FOR EACH local maximum m of pdf
            local_partition := partition around m;
            local_h := Silverman (local_partition);
            FOR i := 1 TO |X|
                IF (X[i] belongs to local_partition)
                    H [ i ] = local_h;
                END
            END
        END
        pdf:= Compute_pdf (X , H);
    END
    K := number of local maxima of pdf;
    FOR EACH local maximum m of pfd
        i := index of local maximum;
        P_i := partition around m;
    END
END-PROCEDURE
```

Figure 7.1. Partitioning univariate data based on PDF.

7.2. Partitioning Based on PDF over Principal Curve

As the number of dimensions increases, it is harder to use the pdf estimation methods because of the curse of dimensionality problem. However, the algorithms based on pdf estimation can be applied after dimensionality reduction. We have adapted the algorithm presented in Figure 7.1 for the principal curve subspace.

There are two possible variants. We can just project the data set on the principal curve and use the projections as the one-dimensional case, or may compute the bandwidths in the multi-dimensional case and project the multi-dimensional pdf on the principal curve. For the former variant, the polygonal approximation of the curve introduces distortions in the knots of the curve. The latter version might work better for a low number of dimensions and for large enough data sets, but as the number of dimensions increases, the results may be worse due to the curse of dimensionality problem.

The results of the algorithm are good if the projections of clusters on the curve are as separate as possible. For that, we split the space in a similar way as in the hierarchical principal clustering algorithm presented in the previous section. Each data subset is simple enough to be well represented by a principal curve. We can determine the clusters for each subset and sum them to obtain the clustering of the whole data set.

One possible way to get a good view of the multi-dimensional density function on one dimension is to project it as shown in Figure 7.2.

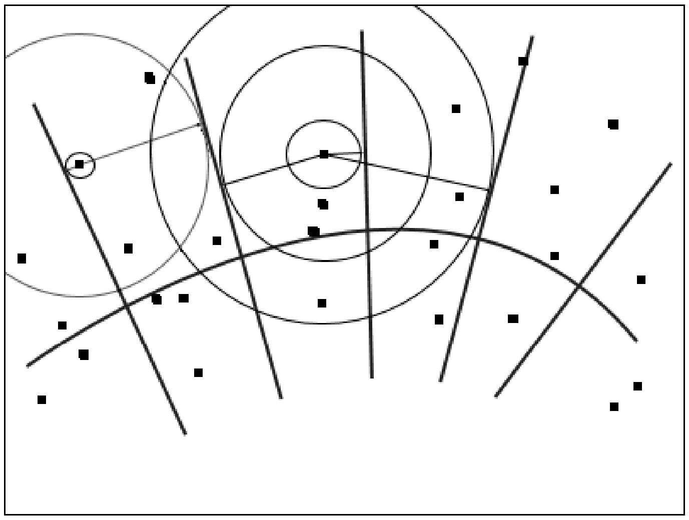

Figure 7.2. Projection of pdf over the principal curve.

To construct the projected pdf estimation on the curve, initially we uniformly sample the principal curve. Distances between kernel centers (data points) and the estimation points are computed based on their projections on the curve, and the pdf values in the estimation points are computed.

We have used the algorithm in Figure 7.1 with one modification. The bandwidths are computed using the Euclidean distances in the multi-dimensional space and the Silverman's rule of thumb for multi-dimensional space. This way, we avoid the errors in the projection introduced by the polygonal approximation of the curve.

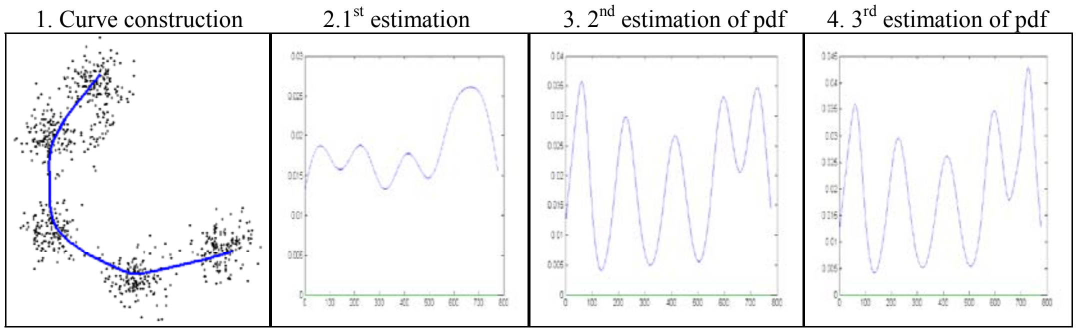

Figure 7.3. Iterative estimation of the projection of the pdf on the principal curve; the algorithm converges in 3 steps.

7.3. Complexity of the Method

The most important step in the method is the computation of the bandwidths. Using Silverman's rule of thumb, standard deviation is computed in linear time, thus the bandwidths are computed linearly. The projection of the pdf on the curve has the complexity O(MN), where M is the number of estimation points and N the number of data points. We consider M=N, so pdf projection takes O(N^2) time. This can be heavily reduced, since there is no need to estimate the kernel for far estimation points (the added value is negligible).

If the method converges, the number of different bandwidths increases each iteration reaching the number of clusters. In this case, it is upper-bounded by N, but for real cases the number of steps is much smaller than N. Thus, the complexity for the algorithm is bounded by O(N^3).

8. Experiments

For experimental purpose, we use three types of data sets:

- A sets,
- S sets,
- natural image data sets.

The A data sets (A1, A2, A3) are artificial and contain different number of two-dimensional Gaussian clusters having about the same characteristics and number of data points. The S sets (S1, S2, S3, S4) are also artificial and two-dimensional with varying complexity in terms of spatial data distributions. The real data sets (housec5, bridge, camera, missa2) do not necessary present natural clusters but they are used merely as test data for vector quantization. See Figures 8.1, 8.2 and 8.3 for illustration of the data sets.

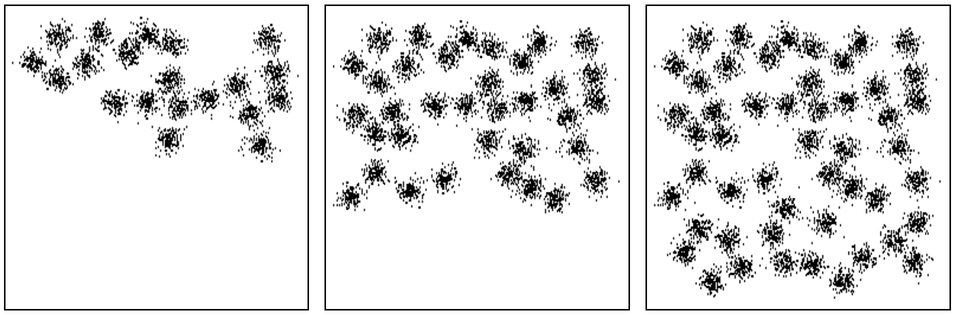

Figure 8.1. A1, A2 and A3 data sets.

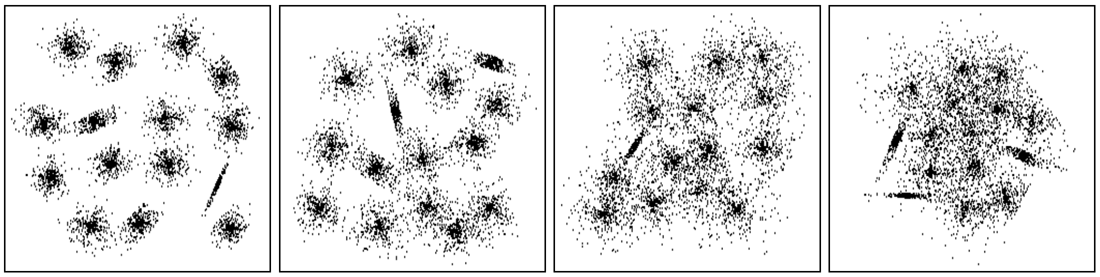

Figure 8.2. S1, S2, S3 and S4 data sets.

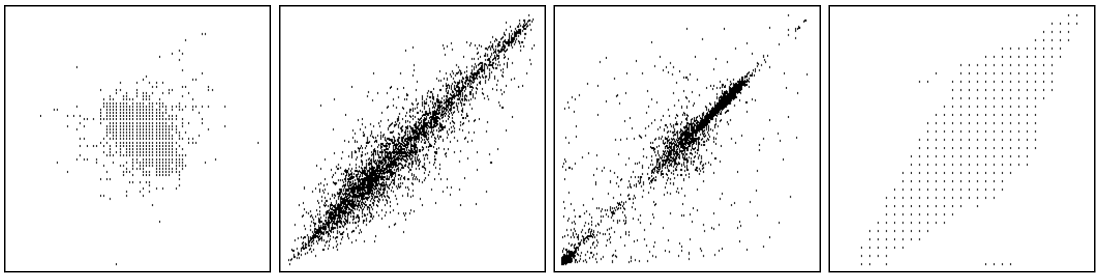

Figure 8.3. Two first dimensions of the natural image data sets Missa2, Bridge, Camera and Housec5.

The real data sets in Figure 8.3 have been generated from the images shown in Figure 8.4. The vectors in the Bridge and Camera sets are 4×4 non-overlapping blocks taken from the corresponding gray-scale images. In Missa2, the vectors were formed as 4×4 difference blocks of two subsequent frames in the video sequence. The last data set, Housec5, consists of the color values of the *RGB* image pre-quantized to 5 bits per color.

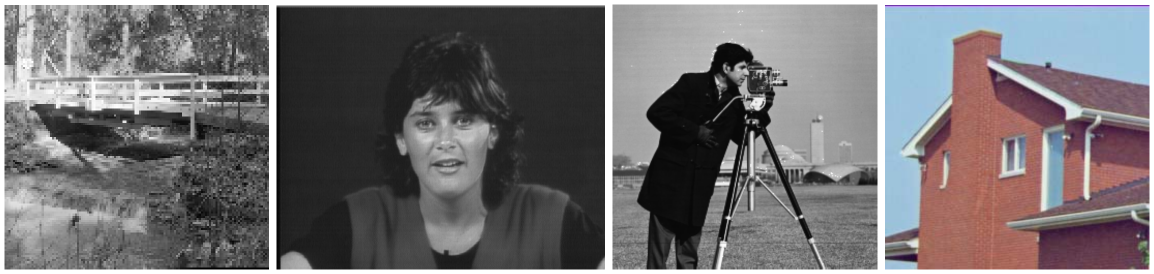

Figure 8.4. Bridge, Miss America, Camera and House images.

8.1. Clustering Based on Principal Axis

The results of the experiments for the algorithm based on principal axis (PAC) is shown in Table 8.1. For the clustered data sets, the number of clusters given to the algorithm is the real one, and the real data sets are quantized to 20 clusters.

For the artificial data sets, the algorithm does not yield good results for the MSE, but it can be used as a tool for K-means initialization. For the natural data sets, the results are better as relatively small improvement was achieved by K-means. The reason is that real data sets come from natural images that present highly linear correlations, which are easier to model by linear projection than the more complex artificial data sets. The data is along the first diagonal, which matches the principal axis to offer a good model for data.

Table 8.1. Results for the PAC and PAC + K-means algorithms.

Dataset	No of clusters	No of dimensions	MSE	MSE after K-means	K-means steps	Improvement ratio
A1	20	2	$8.30*10^6$	$2.02*10^6$	12	4.0
A2	35	2	$15.65*10^6$	$2.74*10^6$	18	5.7
A3	50	2	$17.65*10^6$	$3.69*10^6$	31	4.7
S1	15	2	$8.40*10^9$	$1.43*10^9$	20	5.8
S2	15	2	$7.73*10^9$	$1.86*10^9$	26	4.0
S3	15	2	$5.71*10^9$	$1.68*10^9$	15	3.3
S4	15	2	$6.34*10^9$	1.57*10	24	4.0
Housec5	20	3	51.67	39.35	10	1.2
Bridge	20	16	430.2	366.8	79	1.1
Camera	20	16	355.3	276.1	42	1.2
Missa2	20	16	13.07	10.05	15	1.3

8.2. Clustering Based on Principal Curve

The results of clustering on principal curve (PCU), shown in Table 8.2, are highly influenced by the quality of the curve. The penalty curve parameter determines the shape of the curve. We provide the results for several parameter values of the curve (the penalty coefficient) because it influences the results of clustering.

As the value of the penalty coefficient gets lower, the algorithm suffers from overfitting, and as the value of the penalty coefficient gets higher, the principal curve looks more like the principal axis, and the result of the clustering becomes also worse. In the artificial data sets the minimum in the MSE can be observed more clearly. Unfortunately, the values of the penalty coefficient that give the best results are not the same for all the data sets. Still, values of the penalty coefficient in the rage 0.01 – 0.1 give good results for most of the data sets.

8.3. Hierarchical Clustering Based on Principal Curve

Table 8.3 presents the results for the hierarchical clustering based on principal curve algorithm. The table presents the values for pre-clustering step (based on principal curve), and the final results. The data containing the MSE values for the K-means iterated pre-clustering results is given just for comparison, since the algorithm does not use that step.

The hierarchical principal curve clustering improves the basic algorithm for the complicated data sets, when the data cannot be modeled by the principal curve. The best improvements are observed for the data set with the largest number of clusters, A3. The smallest improvements are observed for the natural image data sets. As a general model, the real image data sets are the simplest ones, as the data is along the first diagonal.

Table 8.2. Results for the PCU and PCU + K-means algorithms.

Dataset	Penalty Coefficient	MSE	MSE after K-means	K-means steps	Improvement ratio
A1	0.01	$2.06*10^6$	$2.02*10^6$	5	1.0
	0.04	$2.03*10^6$	$2.02*10^6$	5	1.0
	0.1	$2.10*10^6$	$2.02*10^6$	5	1.0
A2	0.01	$1.93*10^6$	$1.93*10^6$	3	1.0
	0.04	$1.95*10^6$	$1.93*10^6$	5	1.0
	0.1	$3.18*10^6$	$2.13*10^6$	9	1.5
A3	0.01	$2.05*10^6$	$1.92*10^6$	7	1.1
	0.04	$2.18*10^6$	$2.07*10^6$	14	1.1
	0.1	$2.93*10^6$	$2.32*10^6$	11	1.3
S1	0.01	$8.96*10^8$	$8.91*10^8$	3	1.0
	0.04	$8.91*10^8$	$8.91*10^8$	2	1.0
	0.1	$8.91*10^8$	$8.91*10^8$	2	1.0
S2	0.01	$1.59*10^9$	$1.32*10^9$	6	1.2
	0.04	$1.32*10^9$	$1.32*10^9$	4	1.0
	0.1	$1.33*10^9$	$1.32*10^9$	4	1.0
S3	0.01	$2.23*10^9$	$1.85*10^9$	18	1.2
	0.04	$1.73*10^9$	$1.68*10^9$	7	1.0
	0.1	$1.70*10^9$	$1.68*10^9$	10	1.0
S4	0.01	$1.97*10^9$	$1.65*10^9$	20	1.2
	0.04	$1.68*10^9$	$1.65*10^9$	31	1.0
	0.1	$1.62*10^9$	$1.57*10^9$	14	1.0
Housec5	0.01	37.3	36.3	12	1.0
	0.04	37.6	36.2	13	1.0
	0.1	41.9	38.6	8	1.1
Bridge	0.01	380	371	19	1.0
	0.04	405	369	46	1.1
	0.1	428	373	39	1.1
Camera	0.01	308	282	13	1.1
	0.04	328	277	28	1.2
	0.1	335	279	17	1.2
Missa2	0.01	10.94	9.79	19	1.1
	0.04	11.37	9.70	15	1.2
	0.1	11.51	9.81	9	1.2

Table 8.3. Results for the HPCU and HPCU + K-means algorithms.

Data set	penalty coefficient	Initial distortion	after K-means	K-means steps	after partition distortion	after K-means	K-means steps
A1	0.01	$2.46*10^6$	$2.02*10^6$	7	$2.46*10^6$	$2.02*10^6$	7
	0.04	$2.03*10^6$	$2.02*10^6$	5	$2.02*10^6$	$2.02*10^6$	2
	0.10	$2.11*10^6$	$2.02*10^6$	5	$2.02*10^6$	$2.02*10^6$	4
A2	0.01	$2.11*10^6$	$1.93*10^6$	7	$1.94*10^6$	$1.93*10^6$	4
	0.04	$1.97*10^6$	$1.93*10^6$	4	$1.93*10^6$	$1.93*10^6$	3
	0.10	$2.28*10^6$	$1.92*10^6$	5	$1.99*10^6$	$1.93*10^6$	5
A3	0.01	$2.00*10^6$	$1.92*10^6$	5	$1.93*10^6$	$1.92*10^6$	4
	0.04	$3.14*10^6$	$2.11*10^6$	10	$2.22*10^6$	$2.05*10^6$	14
	0.10	$2.74*10^6$	$2.08*10^6$	5	$2.00*10^6$	$1.92*10^6$	9
S1	0.01	$15.23*10^8$	$14.19*10^8$	7	$14.15*10^8$	$14.11*10^8$	12
	0.04	$9.94*10^8$	$8.91*10^8$	3	$8.91*10^8$	$8.91*10^8$	2
	0.10	$8.93*10^8$	$8.91*10^8$	2	$8.91*10^8$	$8.91*10^8$	2
S2	0.01	$1.36*10^9$	$1.32*10^9$	7	$1.33*10^9$	$1.32*10^9$	6
	0.04	$1.33*10^9$	$1.32*10^9$	4	$1.32*10^9$	$1.32*10^9$	3
	0.10	$1.33*10^9$	$1.32*10^9$	7	$1.32*10^9$	$1.32*10^9$	3
S3	0.01	$2.33*10^9$	$1.88*10^9$	23	$1.78*10^9$	$1.68*10^9$	10
	0.04	$1.69*10^9$	$1.68*10^9$	11	$1.69*10^9$	$1.68*10^9$	12
	0.10	$1.70*10^9$	$1.68*10^9$	10	$1.69*10^9$	$1.68*10^9$	6
S4	0.01	$1.62*10^9$	$1.57*10^9$	15	$1.64*10^9$	$1.57*10^9$	11
	0.04	$1.60*10^9$	$1.57*10^9$	13	$1.58*10^9$	$1.57*10^9$	10
	0.10	$1.62*10^9$	$1.57*10^9$	14	$1.60*10^9$	$1.57*10^9$	11
Housec5	0.01	38.25	36.19	22	38.25	36.19	22
	0.04	37.97	36.26	26	42.96	37.99	36
	0.10	41.48	38.61	7	39.25	36.45	10
Bridge	0.01	377.9	370.8	24	372.3	365.3	28
	0.04	408.2	370.1	40	405.8	366.6	93
	0.10	413.7	366.4	91	406.0	369.2	56
Camera	0.01	302.13	283.1	22	302.1	283.1	22
	0.04	302.43	278.3	21	289.0	277.3	19
	0.10	333.60	280.5	21	296.1	278.8	20
Missa2	0.01	10.13	9.71	11	10.86	9.73	17
	0.04	11.71	10.26	12	10.10	9.62	10
	0.10	12.32	10.03	13	10.43	9.70	13

8.4. Comparison of the Algorithms

Next, we make a comparison between the different methods introduced in this work, see Tables 8.4, 8.5 and 8.6. Comparative results include *randomized local search* (RLS) [FK2000], and the popular K-means. The MSE-values for the RLS method has been considered when the value of the MSE seems to stabilize; a slightly better solution is found after a larger number of iterations. The K-means numbers are the best results obtained by 10 repeated trials.

For the algorithms that are parameterized by the principal curve, we have chosen the best results for comparison. Both the PCU and HPCU algorithms tuned by K-means give values that are very close to the RLS results. The difference to repeated K-means is also very small. The results for these algorithms are very close to the global optimum, for the studied data sets. While the best results of the HPCU algorithm do not show a much better improvement to the best results of the PCU, it is much more stable relative to the penalty coefficient.

Table 8.4. Comparison of the algorithms for the A data sets.

Method	A sets		
	A1	A2	A3
K-means	$20.24*10^5$	$19.32*10^5$	$19.29*10^5$
RLS	$20.24*10^5$	$19.32*10^5$	$19.29*10^5$
PAC	$83.00*10^5$	$156.57*10^5$	$176.59*10^5$
PAC + K-means	$20.24*10^5$	$27.41*10^5$	$36.95*10^5$
PCU	$20.30*10^5$	$19.33*10^5$	$20.59*10^5$
PCU + K-means	$20.24*10^5$	$19.32*10^5$	$19.29*10^5$
HPCU	$20.24*10^5$	$19.42*10^5$	$19.36*10^5$
HPCU + K-means	$20.24*10^5$	$19.32*10^5$	$19.29*10^5$

Table 8.5. Comparison of the algorithms for the S data sets.

Method	S sets			
	S1	S2	S3	S4
K-means	$134.44*10^7$	$13.27*10^8$	$16.88*10^8$	$15.70*10^8$
RLS	$89.17*10^7$	$13.27*10^8$	$16.88*10^8$	$15.70*10^8$
PAC	$840.48*10^7$	$77.34*10^8$	$57.11*10^8$	$63.40*10^8$
PAC + K-means	$143.54*10^7$	$18.65*10^8$	$16.88*10^8$	$15.70*10^8$
PCU	$89.18*10^7$	$13.29*10^8$	$16.94*10^8$	$15.91*10^8$
PCU + K-means	$89.17*10^7$	$13.27*10^8$	$16.88*10^8$	$15.70*10^8$
HPCU	$89.17*10^7$	$13.30*10^8$	$16.90*10^8$	$15.88*10^8$
HPCU + K-means	$89.17*10^7$	$13.27*10^8$	$16.88*10^8$	$15.70*10^8$

Table 8.6. Comparison of the algorithms for the image data sets.

Method	Image data sets			
	Housec5	Bridge	Camera	Missa2
K-means	36.4	365	278	9.64
RLS	35.6	364	270	9.50
PAC	51.6	430	355	13.07
PAC + K-means	39.3	366	276	10.05
PCU	37.3	377	295	9.99
PCU + K-means	36.1	365	273	9.69
HPCU	38.25	372	289	10.10
HPCU + K-means	36.19	365	277	9.62

9. Conclusions

In this work, we have considered different issues related to solving the clustering problem using one-dimensional projections of the data set. The one-dimensional projection is particularly interesting because optimal clusters in one-dimensional spaces, as opposed to VQ, can be optimally solved.

Given the sequence of data, optimal segmentation relative to a usual clustering criterion (such as MSE) can be easily done using the dynamic programming technique. The method has been applied to scalar quantization. Thus, we have studied different ways to sequence the vector space data sets and apply the same method for multi-dimensional data sets.

In the beginning, we have studied the projection on the principal axis. The method has been applied before for color quantization problem. Good results are obtained only for simple data sets, and fine-tuning by K-means is almost always obligatory. We have also tried to optimize the sequence in the sense of TSP problem. The order provided by the TSP sequence may contain the clusters as subsequences, but the optimal TSP sequence is another NP-complete problem. Still, the method proves to increase the accuracy of the results.

The main part of the work concentrates on ordering along the principal curve. It has the advantage of being a non-linear projection that can model more complex types of data. The most sensitive aspect of the method consists in avoiding the overfitting problem. The coefficient that indirectly controls the length of the curve should be dependent on the characteristics of the data set. We have found a way to split the data set, so that the subsets are suitable for being modeled by the principal curves; this method can be applied to complex data sets. It does not completely solve the overfitting problem, but the choice of the parameters is less critical.

The last part of the thesis concentrates on determining the number of clusters using the principal curve. The success of the proposed method depends on the quality of the curve. We did not provide the proof for the convergence of our method, but we showed that the problem can be solved using the projections on the principal curve.

The current work shows that the principal curve can be successfully applied to cluster analysis. Future work should concentrate on defining and constructing curves, avoiding problems such as parameter tuning and overfitting.

References

[BR1992] Banfield J. D. and Raftery A. E. Ice floe identification in satellite images using mathematical morphology and clustering about principal curves. *Journal of the American statistical Association* vol. 87, no. 417., pp. 7-16, 1992.

[B1961] Bellman, R. Adaptive control processes: a guided tour. Princeton University Press, 1961.

[BSW1998] Bishop C. M., Svensen M. and Williams C. K. I. The generative topographic mapping, *Neural Computation*, vol. 10, issue 1, pp. 215-235, 1998.

[CT2000] Cristianini N. and Taylor J. S. An Introduction to SVMs and Other Kernel Based Learning Methods, Cambridge University Press, 2000.

[D1997] Demmel J. Applied Numerical Linear Algebra. SIAM, Philadelphia, 1997.

[F2002] Fodor I. K. A Survey of Dimension Reduction Techniques. LLNL technical report, June 2002. UCRL-ID-148494.

[FKSC2000] Fränti P., Kaukoranta T., Shen D.-F. and Chang K.-S. Fast and memory efficient implementation of the exact PNN, *IEEE Transactions on Image Processing*, vol. 9, no. 5, pp. 773-777, 2000.

[F2000] Fränti P. Genetic algorithm with deterministic crossover for vector quantization. *Pattern Recognition Letters*, vol. 21, no. 1, pp. 61-68, 2000.

[FK2000] Fränti P. and Kivijärvi J. Randomized Local Search Algorithm for the Clustering Problem, *Pattern Analysis and Applications*, vol. 3, pp. 358-369, 2000.

[FT1974] Friedman J. H. Tukey J. W. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, vol. 23, pp. 881-890, 1974.

[F1987] Friedman J. H. Exploratory Projection Pursuit. *Journal of American Statistical Association*, vol. 82, pp. 249-266, 1987.

[GJ1979] Garey M. and Johnson D. Computers and Interactibility. W. H. Freeman, San Francisco, 1979.

[G2002] Gering, D. Linear and Nonlinear Data Dimensionality Reduction, http://www.ai.mit.edu/people/gering/areaexam/areaexam.pdf, 17.04.2002.

[G1980] Gordon A. D. Classification. Chapman and Hall, London, 1980.

[H1996] Hardy A. On the number of clusters. *Computational Statistics and Data Analysis*, vol. 23, issue 1, pp. 83-96, 1996.

[H1989] Hastie T., Stuetzle W. Principal Curves. *Journal of the American Statistical Association* vol. 84, no. 406, pp. 502-516, 1989.

[H1999] Hyvärinen A. Survey on Independent Component Analysis. *Neural Computing Surveys* 2:94--128, 1999.

[JD1988] Jain A. K and Dubes R. C. Algorithms for clustering data. Prentice-Hall advanced reference series. Prentice-Hall, Inc, Upper Saddle River, NJ, 1988.

[JMF1999] Jain A.K., Murty M. N. and Flynn P.J. Data Clustering: A review. *ACM Computing Surveys*, vol. 31, no. 3, 1999.

[JW1992] Johnson R. A., Wichern D. W. Applied Multivariate Statistical Analysis. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[KKLZ2000] Kegl B., Krzyzak A., Linder T. and Zeger K. Learning and Design of Principal Curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence* vol. 22, no. 3, pp. 281-297, 2000.

[K2001] Kohonen T. Self-organizing maps, *Springer Series in Information Sciences*, vol. 30, Springer, Berlin, Heidelberg, New York, 2001

[M1995] MacKay D. J. C. Bayesian Neural Networks and Density Networks. *Nuclear Instruments and Methods in Physics Research*, Section A, vol. 354, no.1, pp. 73-80, 1995.

[M1967] MacQueen J. Some methods for classification and analysis of multivariate observations. *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281-296, 1967.

[MKB1995] Mardia K.V., Kent J.T. and Bibby J.M. Multivariate analysis. Probability and Mathematical Statistics. Academic Press, 1995.

[MN1998] Marron J. S. and Nolan G. Canonical Kernels for Density Estimation. *Statistics and Probability Letters*, vol. 7, pp. 195-199, 1998.

[P1991] Papoulis A. Probability, Random Variables, and Stochastic Process. McGraw-Hill. 3rd Edition, 1991.

[RMH2004] Rome J.E., Miasnikov A.D. and Haralick R.M. A hierarchical projection pursuit algorithm. *Proceedings of the International Conference on Pattern Recognition*, 2004.

[S1994] Sain S. R. Adaptive Kernel Density Estimation. Ph.D. Thesis. Rice University, Houston, Texas, 1994.

[SK2002] Sandilya S and Kulkarni S. R. Principal curves with bounded turn. *IEEE Transactions on Information Theory* vol. 48, issue 10, pp. 2789-2793, 2002.

[S1986] Silverman B. W. Density Estimation for Statistics and Data Analysis. Chapman & Hall, London, New York, 1986.

[SCH1975] Slagle J. L., Chang C. L. and Heller S. L. A clustering and data-reorganization algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, vol. 5, pp. 121-128, 1975.

[SJ1993] Soong F. K. and Juang B. H. Optimal quantization of LSP parameters. *IEEE Transactions on Speech and Audio Processing*, vol. 1, issue 1, pp. 15-24, 1993.

[TS1992] Terrell G. R. and Scott D. W. Variable Kernel Density Estimation. *The Annals of Statistics*, vol. 20, pp. 1236-1265, 1992.

[TGPL2004] Theis F. J., Gruber P., Puntonet C. G. and Lang E. W. Connecting geometric independent component analysis to unsupervised learning algorithms. *Proceedings of International Conference on Engineering of Intelligent Systems*, accepted, 2004.

[VVK2002] Verbeek J.J., Vlassis N. and Krose B. A k-segments algorithm for finding principal curves. *Pattern Recognition Letters* vol. 23, issue 8, pp. 1009-1017, 2002.

[W1992] Wu X. Color Quantization by dynamic programming and principal analysis. *ACM Transactions on Graphics*, vol. 11, issue 4, pp. 348-372, 1992.

[W1991] Wu X. Optimal quantization by matrix searching. *Journal of Algorithms*, vol. 12, issue 4, pp. 663-673, 1991.

[YH1987] Young F.W. and Hamer R. M Multidimensional scaling: history, theory and applications. Erlbaum, New York, 1987.

[Z1971] Zahn C.T. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*. C-20, 68-86, 1971.