Efficient density-based methods for knowledge discovery in databases

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften
der Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades eines Doktors der
Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
Ralph Krieger

aus Köln

Berichter: Universitätsprofessor Dr. rer. nat. Thomas Seidl
Professor Dr. Bart Goethals

Tag der mündlichen Prüfung: 09. Juli 2008

Contents

List of Figures

Efficient density-based methods for classification

Acknowledgements

There are many people who supported and encouraged me while I was working on my thesis. At this point I would express my gratitude to all of them, even if I cannot mention everyone here.

First of all, I would like to thank my supervisor Prof. Dr. Thomas Seidl. This thesis would not have been possible without your advice, helpful discussions and support. Thank you for always being interested in my work.

I am also very grateful to Prof. Dr. Bart Goethals for his time and valuable discussions. Thank you also for agreeing to review this thesis.

Special thanks also to my colleagues for the inspiring discussions and fruitful cooperations. Working in an interested and humorous team was very helpful for this thesis.

Last but not least I would like to thank my parents, family and friends. But especially I want to thank my wife Melanie for all the love and great encouragement during the last years. I would also like to thank Julia for just being there and reminding me that there exists a task more important than this work.

Abstract

Today's data storage facilities allow recording of billions of transactions from business applications, scientific sensor readings, monitoring systems etc. Scientists developing new drugs, system administrators monitoring complex technical processes, and decision makers being responsible for complex social or technical systems require an overview and even a deeper understanding of their respective data. The knowledge discovery in databases (KDD) process has been designed to identify hidden patterns in large data resources. A central step of the KDD process is the data mining task. Major data mining tasks are clustering and classification. Density-based approaches have proven to be very effective for many data mining methods. However, the good effectiveness often comes at the cost of a high runtime complexity. This thesis presents new efficient density-based approaches for different data mining applications whereas the effectiveness of the new developed methods is always kept in mind.

The first part of this thesis is concerned with new density-based clustering methods. Clustering is a data mining task for summarizing data such that similar objects are grouped together while dissimilar ones are separated. Density-based approaches have shown to successfully mine arbitrary shaped clusters even in the presence of noise. In multi-dimensional or high dimensional data, clusters are typically hidden by irrelevant attributes and do not show across the full space. As relevance of attributes is not globally uniform for all clusters, global dimensionality reduction approaches are not adequate. Subspace clustering aims at automatically detecting clusters and their relevant attribute projections. This work presents a new clustering model DUSC which guarantees a comparable and redundancy free subspace clustering result. As the number of possible subspaces is exponential in the number of dimensions subspace clustering is a computationally challenging task. The algorithm eDUSC developed in this work is based on a filter-and-refinement

architecture which avoids repeated database scans. Further on, this work proposes a new visualization technique for subspace clusters and a specialized clustering technique for multi-dimensional sequence databases.

The second part of this thesis proposes new density-based methods for classification. Classification aims at assigning a class label to unknown objects. Various approaches for classifying objects have been investigated in the last decades. Classifiers based on statistical approaches have been most intensively studied in the literature and results like asymptotical behavior and classification bias have been derived. To apply statistical classifiers the density of objects has to be estimated. In this work, a hierarchy of density estimators is proposed which makes the classification of objects possible anytime. Additionally, a new classification method using subspace clusters for higher dimensionalities is developed in this thesis.

The proposed density-based clustering and classification methods are evaluated in terms of both efficiency and effectiveness in thorough experiments on real world and synthetic data.

Zusammenfassung

Moderne Datenspeicheranlagen ermöglichen die Erfassung von Billionen von Geschäftstransaktionen, wissenschaftlichen Sensormessungen, Meldungen von Überwachungssystemen etc. Verantwortliche Wissenschaftler in der Arzneimittelentwicklung, Systemadministratoren, die komplizierte technische Prozesse überwachen und Entscheidungsträger komplexer sozialer oder technischer Systeme benötigen eine Übersicht über bzw. einen tieferen Einblick in ihre erfassten Daten. Der "Knowledge discovery in databases" (KDD) Prozess wurde entwickelt, um versteckte Muster innerhalb großer Datenbanken ausfindig zu machen. Ein zentraler Schritt des KDD Prozesses ist das Data Mining. Hauptaufgaben des Data Minings sind das Clustering und die Klassifikation von Daten. Dichtebasierte Ansätze haben sich als sehr effektive Data Mining Methoden bewährt. Jedoch bringt die hohe Effektivität eine hohe Laufzeitkomplexität mit sich. In dieser Doktorarbeit werden neue, effiziente, dichtebasierte Ansätze für verschiedene Datenanalyseanwendungen vorgestellt, wobei die Effektivität nicht außer Acht gelassen wird.

Der erste Teil dieser Arbeit befasst sich mit neuen dichtebasierten Clustering Methoden. Clustering ist eine Data Mining Aufgabe, welche Daten so zusammenfasst, dass Gruppen ähnlicher Objekte von unähnlichen separiert werden. Dichtebasierte Ansätze haben sich als erfolgreich bei der Suche beliebig geformter Cluster innerhalb verrauschter Datensätze herausgestellt. In mehr- oder hochdimensionalen Daten werden Cluster normalerweise durch irrelevante Attribute versteckt und sind daher im vollen Datenraum nicht zu erkennen. Da die Relevanz von Attributen nicht für alle Cluster global einheitlich ist, können globale Dimensionsreduktionstechniken nicht sinnvoll eingesetzt werden. Die Zielsetzung von Subspace Clustering Algorithmen ist das automatische Auffinden von Clustern mit der zugehörigen Attributprojektion. Diese Arbeit präsentiert DUSC, ein neues Clustering Modell, das vergleichbare und redundanzfreie Clustering Ergebnisse garantiert.

Aus Sicht des Berechnungsaufwandes stellt Subspace Clustering, wegen der exponentiellen Abhängigkeit der Anzahl möglicher Teilräume von der Anzahl Dimensionen, eine Herausforderung dar. Der Algorithmus eDUSC, welcher im Rahmen dieser Arbeit entwickelt wurde, basiert auf einer Filter-und-Verfeinerungsmethode, wodurch das wiederholte Durchsuchen der Datenbank vermieden wird. Weiterhin werden in dieser Arbeit Visualisierungstechniken für Subspace Cluster vorgestellt, sowie eine spezialisierte Clustering Technik für mehrdimensionale Sequenzdatenbanken.

Im zweiten Teil dieser Doktorarbeit werden neue dichtebasierte Methoden zur Klassifikation vorgestellt. Das Ziel der Klassifikation ist die Bestimmung eines Klassenlabels für unbekannte Objekte. In den letzen Jahrzehnten wurden verschiedene Ansätze für die Klassifikation von Objekten vorgestellt. Klassifikatoren, welche auf statistischen Ansätzen basieren, wurden in der Literatur sehr intensiv untersucht und Ergebnisse über das asymptotische Verhalten und die Klassifikationstendenz wurden hergeleitet. Zur Anwendung statistischer Verfahren ist das Schätzen der Dichte für Objekte notwendig. In dieser Arbeit wird eine Hierarchie von Dichteschätzern vorgestellt, die Klassifikation von Objekten zu jedem Zeitpunkt möglich macht. Weiterhin wird in dieser Doktorarbeit ein neuer Klassifikator für hochdimensionale Daten auf Basis von Subspace Clusterings entwickelt. In umfangreichen Experimenten wird mit Hilfe von synthetischen und realen Daten sowohl die Effizienz als auch die Effektivität der vorgestellten dichtebasierten Clustering- und Klassifikationsmethoden untersucht.

Density-based methods for knowledge discovery in databases

Knowledge discovery in databases

Increasingly large data resources in life sciences, mobile information and communication, e-commerce, and other application domains require computer-based techniques for gaining knowledge. More and more data are produced by sensor networks, technical or financial monitoring systems, scientific experiments, or telecommunication networks. For decision makers, scientists and other data analysts unknown dependencies in the observed measurements are crucial for development of new models that detect and explain causalities. Knowledge discovery in databases aims at generating novel and interesting information hidden in the data [HK01]. To find interesting patterns in a data set a knowledge discovery process is typically divided into four steps (see also Figure 1). The first step creates a consistent view on the data by integrating and cleaning data stored in different sources. The combined data is often managed in a data warehouses. Depending on the desired analysis the task relevant data is selected from the data warehouse or

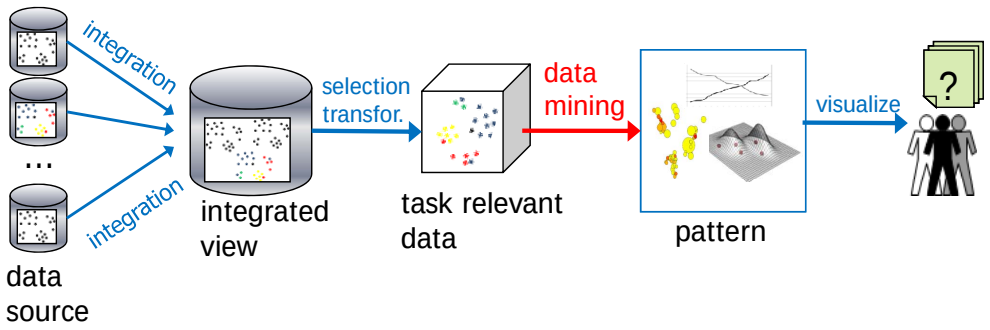

Figure 1: Knowledge Discovery in Databases (KDD) process

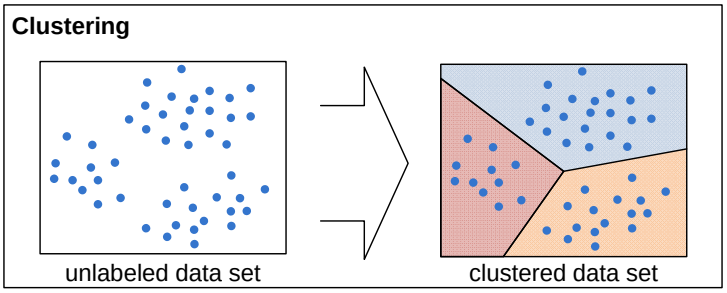

Figure 2: Clustering a data set

transformations and projections are calculated. Subsequently the data mining step extracts patterns from the task relevant data. Finally visualization techniques are used to present the extracted patterns to the user.

As a central step of the KDD process the data mining step has attracted much attention from scientific research. Typical data mining approaches are association rule mining, clustering and classification methods. Clustering and classification methods are often used to analyze multi-dimensional real valued data sets. Common subtasks of many clustering or classification algorithms are density-based (i.e. they rely on densities of regions or objects). Since databases tremendously grow in size classification and clustering algorithms have a special need for efficient density-based methods.

Density-based methods for clustering

Clustering aims at grouping data into similarity-based subgroups. Typically clustering methods do not assume any prior knowledge, i.e. the number of groups contained in the data or the type of clusters is unknown. Consequently clustering methods are often called *unsupervised*. Figure 2 gives an abstract example for a clustering. In this example the given data set is grouped into three regions.

In the literature, several clustering paradigms exist [HK01]. Density-based clustering defines clusters as dense areas separated by sparsely populated areas. It has been shown to successfully detect clusters of arbitrary shape in many settings [EKSX96, HK98]. Density of an object is measured either by mere counting of objects or by more complex functions on the number and location of objects in the neighborhood. The underlying den-

sity model, from mere counting of objects in a neighborhood range to local density attractors, has great impact on the clustering result.

For any paradigm, clustering in high dimensional spaces is obstructed by the noise of irrelevant attributes. The "curse of dimensionality" describes the effect that distances become more and more similar as dimensionality increases. Consequently, meaningful clusters no longer exist [BGRS99].

One solution for clustering data in multi-dimensional or high dimensional spaces is subspace clustering. In scenarios with many attributes or with noise, clusters are often hidden in subspaces of the data and do not show up in the full dimensional space. A global reduction to relevant attributes is often infeasible, as relevance of attributes is not necessarily globally uniform. Varying relevance of attributes for individual clusters requires clustering over any possible subset of the attributes.

Subspace clustering therefore aims at detecting clusters in any possible attribute combination. Density-based approaches are very popular to determine clusters in subspace. As the number of subspace projections is exponential ly with the number of dimensions, subspace clustering methods have a tremendous need for efficient density-based methods. We will go into details of subspace clustering in Part I.

Density-based methods for classification

Classification methods aim at assigning unlabeled objects to predefined groups. As the number of groups are known in advance classification methods are called *supervised* methods. Many applications have the need to predict class labels for new objects like speech and image recognition systems, e-mail spam and denial of service attacks detection systems etc. Since an a priori knowledge about the classes is given the goal of a classifier is to learn the structure of the predefined groups from a given labeled data set (the training data). Figure 3 illustrates an abstract concept for a classifier which learns the predefined classes from dense regions (different classes are represented by different colors).

Using statistical classifiers is a popular approach in pattern recognition system [DHS01]. Many of these methods are based on density estimations (e.g. the Bayes classifier). As the density distribution of the data is typically not known in advance densities are estimated from the training set. A typical

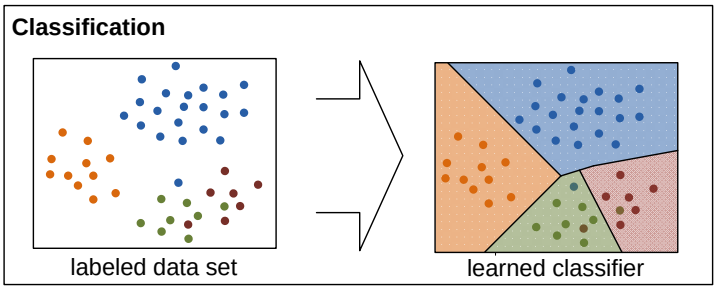

Figure 3: Classifier learned from data set

approach is to estimate the missing parameters of a density model from the training set (e.g. estimate mean and variance for a Gaussian distribution). However, assuming a concrete model is often not appropriate as many density models are unimodal which is a too strong constraint especially for multi-dimensional data sets. Mixture models relax the assumption of one concrete model by using a multimodal mixture of densities. A popular solution is to use an expectation maximization (EM) algorithm to learn a mixture of Gaussian densities from the training set.

Nonparametric functions on the other hand can be used for arbitrary distributions as they make no assumption about the form of the distribution. Kernel densities (or parzen windows) are nonparametric functions which estimate the density of a query object directly from the distribution of the objects contained in the surrounding region. Kernel densities have proven to be an effective method for many applications. However, as the distribution for each query-region has to be determined from the training objects the efficiency of kernel estimators is often poor. In Part II we present details about efficient density-based methods used for classification.

Outline of this work

In this work we present new efficient density-based methods for various clustering and classification applications. The thesis is divided into two major parts:

Part I is concerned with new density-based clustering methods.

In *Chapter 1* we discuss clustering of multi-dimensional data. We further on give an overview over existing clustering methods and discuss problems oc-

curring in multi-dimensional or high dimensional data sets. We then present existing solutions for finding clusters in projections of the data and present preliminary definitions for density-based clustering in subspaces.

Existing subspace clustering methods ignore an effect we call dimensionality bias. *Chapter 2* formalizes this effect and proposes a new subspace clustering model DUSC which solves this problem. Additionally to dimensionality bias the redundancy of clusters is evaluated in thorough experiments on synthetic and real world data sets.

As the number of possible subspaces is exponential in the number of dimensions, subspace clustering is a computationally challenging task. In *Chapter 3* we develop an efficient method eDSUC for our DUSC subspace clustering method. eDUSC achieves a high efficiency by using a filter-and-refinement architecture for avoiding computational intensive database scans. Based on a depth-first algorithm eDUSC exploits in-process redundancy pruning as well as indexing dense regions. Comparing eDUSC with state-of-the-art subspace clustering methods proves the efficiency and effectiveness of this approach.

An important step of the KDD process is to visualize the result of the data mining method to the user. Visualizing the result of subspace clustering algorithms is no trivial task as structures contained in different possible overlapping subspaces have to be presented to the user. The VISA approach presented in *Chapter 4* proposes two new visualization techniques for subspace clusterings.

Chapter 5 proposes a specialized subspace clustering technique for analyzing databases of multi-dimensional sequences which was developed in collaborations with hydrologists. In a current project of the German government different structural quality measures for more than hundred thousand river segments have been recorded. The special challenge in this project was to develop a subspace clustering method for finding clusters of arbitrary sequence length and in any subset of the attributes. The efficiency of the new clustering model is guaranteed by using a two phase approach utilizing different monotonicity properties.

Part II is concerned with new density-based classification methods:

In *Chapter 6* we give a brief review over existing classification algorithms proposed in the literature. Further more the basic notations of the Bayes and nearest neighbor classifier are presented in this chapter.

Based on the Bayesian decision theory an efficient classifier for classifying

objects at any point in time is developed in *Chapter 7*. A novel hierarchy of densities for different classes (the Bayes tree) is exploited for efficient classification. We evaluate the accuracy of our Bayes tree for varying stream inter-arrival rates on synthetic and real world data sets.

Chapter 8 proposes a new classification technique which incorporates the information the class distribution into a specialized subspaces clustering algorithm. Classification based on these classifying subspace clusters exploits both class and local correlation information. In collaboration with a local company optimizing airport scheduling purposes we investigate the classification accuracy of flight delays.

Finally we summarize the major contributions of this work and give an outlook on future research direction.

Part I

Efficient density-based methods for clustering

Chapter 1

Clustering multi-dimensional data

To gain insight into large data resources, data mining provides automatic techniques to extract information from large databases. One of the major tasks for knowledge discovery in databases is clustering. Clustering aims at grouping data such that objects within groups are similar while objects in different groups are dissimilar.

Many different paradigms for clustering data sets have been proposed in the last decades. We first give a brief overview over traditional clustering methods in Section 1.1 before we discuss challenges in clustering multi-dimensional and high dimensional data (see Section 1.2). We then review existing solutions for clustering multi-dimensional data in Section 1.3. Subsequently we give the basic definitions for density-based subspace clustering. As clusters in higher dimensional spaces are often blurred by noise subspace clustering algorithms identify clusters in any possible subspace. In the following chapters of this part we will discuss different aspects of density-based subspaces clustering.

1.1 Full-space clustering

Traditional clustering methods use all attributes available to identify clusters contained in the data. Hence, we refer to traditional clustering methods which do not consider lower dimensional projections of the data as full-spaces clustering methods.

One of the first approaches proposed in the literature to cluster a data set is partitioning clustering. As their name suggests, partitioning clustering methods create a disjoint grouping of the data. Typically they iteratively improve an initial partition of the data until a cost function converges. A well-known example is the k-means algorithm [Mac67], or more statistically founded, EM (Expectation Maximization) [Lau95]. Partitioning algorithms are limited to the detection of convex clusters in data without noise where the number of clusters is known in advance.

As the number of clusters is often not known, hierarchical clustering methods do not determine one unique clustering structure but a hierarchy of clusters. Two different approaches for hierarchical clustering methods can be distinguished: divisive (top-down) and agglomerative (bottom-up) methods. Divisive methods start with a single cluster that contains all objects and recursively pick one cluster for splitting [JD88]. Agglomerative methods first assign each object to an individual cluster and then respectively link the two closest clusters together w.r.t. a chosen distance function. Different distance functions for agglomerative clusterings have been proposed in the literature [Sib73, Def77]. To visualize a hierarchical clustering, dendrograms represent the clustering in a tree-based organization. Each leaf node of the dendrogram corresponds to one object. Inner nodes represent clusters which contain all objects of the leaf nodes of the respective subtree. In each level the two closest clusters are linked together. By choosing different levels in the dendrogram users obtain different groupings of the objects. As no concrete clustering is mined by hierarchical clustering algorithm it is often difficult to extract hidden structures contained in the data based on the clustering result. Hence, visualization techniques like dendrograms are crucial to comprehend the hierarchy of clusters mined by hierarchical clustering algorithms.

Density-based algorithms are capable of detecting arbitrarily shaped clusters and have proven to work remarkably well in noisy settings. The basic idea is that clusters are dense areas separated by sparsely populated areas as defined in DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [EKSX96, HK98]. The initial definition of density has been shown to oversimplify the model of the real density distribution. Fixed neighborhoods of a pre-given ε-range are checked whether or not they contain the defined minimum number of points, thus the distribution of objects is ignored and sensitivity to parameter settings is a challenge. DENCLUE (Density Clustering) thus extends DBSCAN using influence functions to model local densities

of objects [HK98]. Due to efficiency considerations, an approximate computation of density is used. The recent approach of DENCLU2 speeds up the density estimation by a hill-climbing approach with an adjustable step size [HG07]. A hierarchical clustering based on densities instead of distances has been proposed by OPTICS [ABKS99]. OPTICS visualizes a hierarchy of density-connected clusters by computing a density plot. We extend density-based clustering methods to subspace projections in the next sections.

1.2 Problems for clustering multi-dimensional data

For any clustering paradigm, full space clustering algorithms do not scale to higher dimensional spaces. They suffer from the so called "curse of dimensionality". For clustering this means that clusters do not show across all attributes as they are hidden by irrelevant attributes or blurred by noise.

Clustering methods are typically either based on distances (like partitioning and hierarchical clustering) or on densities (like density-based methods). The effects of higher dimensional spaces on distances and density distributions have been widely studied in the literature. In [BGRS99] the authors study the effects of high dimensions on the nearest neighbor $d_{min}(o)$ and the farthest neighbor $d_{max}(o)$ of an object o in detail. They have proven the following equation for different distributions:

$$\forall \varepsilon \geq 0 : lim_{dim \to \infty} P\left(d_{max}(o) < (1+\varepsilon)\, d_{min}(o)\right) = 1$$

This statement formalizes that with growing dimensionalities (dim) the distance to the nearest neighbor is nearly equal to the distance to the farthest neighbor (distances become more and more similar). Consequently, clustering methods based on distance functions have problems to extract meaningful patterns in high dimensional spaces as they either cluster only one object (the nearest neighbor) or nearly the complete data set (the farthest neighbor).

Densities also suffer from the "curse of dimensionality". In [Sil86] the authors describe an effect of higher dimensions on density distributions: *99% of the mass of a ten-dimensional normal distribution is at points whose distance from the origin is greater than 1.6.* This effect is directly opposite in lower dimensional spaces: 90% of the objects have a distance of less than 1.6 from

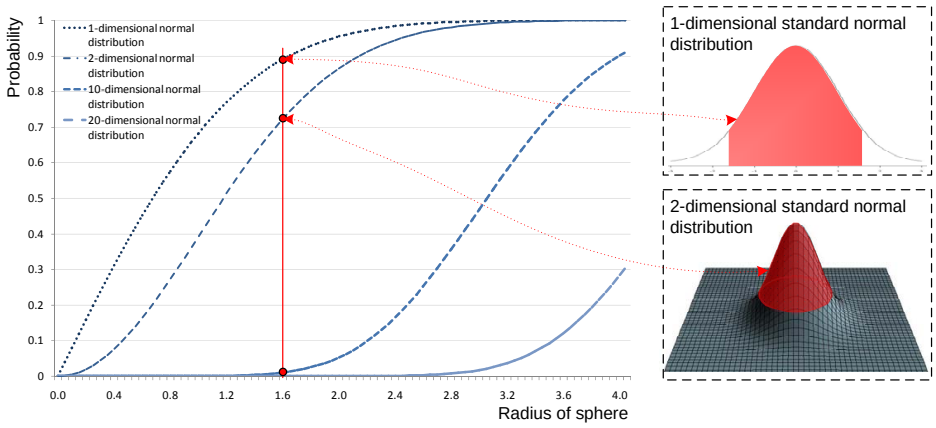

Figure 1.1: Probability of different standard normal distributions

the origin regarding a one-dimensional distribution.

We additionally illustrate the behavior of higher dimensional standard normal distributions in Figure 1.1. For different dimensionalities we measured the probability of an object to be contained in a sphere positioned at the origin w.r.t. the radius of the sphere. On the right part of the figure the probability is graphically illustrated for a sphere of radius 1.6 for a one and two-dimensional standard normal distribution (the corresponding probabilities are marked by red points in the graph). Please recall that density functions are normalized to one and hence the volume of a region under a density function corresponds to the probability of that region. As we can see the probability for an object to be contained in the center of a space is dropping extremely for higher dimensions. In a 20-dimensional space most of the objects have a distance of more than four times the variance per dimension. Consequently nearly every point is positioned at the border of the data space even though the highest density of a high dimensional normal distribution is still at the origin. Density-based clustering methods hence have problems to determine the density of a region as the objects are scattered over the data space.

One possible approach to clustering high dimensional data sets is to apply dimensionality reduction techniques in advance. After reducing the dimensionality clusters can be identified. Dimensionality reduction techniques like PCA (principle components analysis) aim at discarding irrelevant dimensions [Jol86]. However, in many practical applications, no globally irrelevant di-

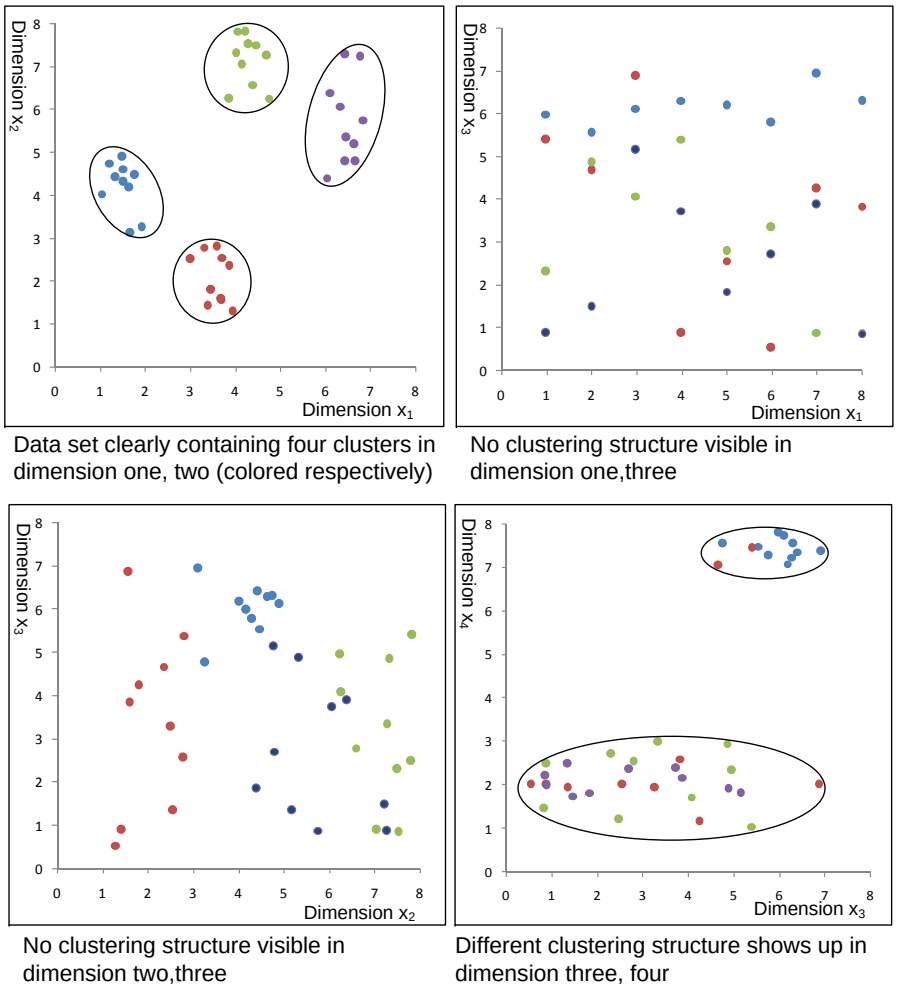

Figure 1.2: Problems of dimensionality reduction and projected clustering techniques

mensions exist. Thus, for these settings, dimensionality reduction can only discover a subset of the actual clusters as some of the original dimensions are ignored.

Figure 1.2 gives an example for a 4-dimensional data set containing clusters in different projections. We illustrate four different projections of the data set in Figure 1.2 (to dimensions x_1x_2, x_1x_3, x_2x_3 and x_3x_4). As depicted clusters are clearly visible in two different projection (x_1x_2 and x_3x_4). Since each of the four dimensions is relevant for some clusters globally irrelevant dimensions do not exist. Hence, removing dimensions will discard

some cluster: e.g. if dimension x_3 and x_4 are removed in advance the four clusters contained in the projection x_1x_2 can be identified by a clustering algorithm (upper left part of Figure 1.2). As we can see dimension x_3 is irrelevant for this clustering as both projections to dimensions x_1x_3 and x_2x_3 does not contain any clusters (see upper right and lower left part of Figure 1.2). However, if dimension x_3 is removed the clustering structure obtained by projecting the data to the dimensions x_3x_4 is lost. Hence, dimensionality reduction techniques can not be applied without loss of clusters. Further on, both clustering structures contained in this example are distorted if all four dimensions are considered. Consequently using a full-space clustering method is also not appropriate for this example data set.

1.3 Solutions for clustering multi-dimensional data

As discussed in the last section finding structures in multi-dimensional to high dimensional spaces is problematic due to the curse of dimensionality. However, clusters still exist in lower dimensional projections. Specialized clustering algorithms have been developed in the last decade which identify clusters in subspaces of the data space. Many traditional clustering algorithms have been extended to projected or subspace clustering algorithms. Methods for identifying clusters in subspaces can be categorized into *top-down* and *bottom-up* methods or into *projected* and *subspace* methods. We discuss the two algorithmic concepts of top-down and bottom-up methods before we present different projected and subspace clustering models in the next section. In this thesis we will concentrate on subspace clustering methods.

The "top" and "bottom" of top-down and bottom-up methods refer to the lattice of subspaces. Figure 1.3 presents the lattice for a five-dimensional space. At the top the full-dimensional space containing all for dimensions is illustrated (using the index set $(1, 2, 3, 4, 5)$) and the empty space at the bottom, accordingly. Each level in the lattice contains all subspaces of a specific dimensionality. Two subspaces are connected if they only differ in one dimension. Bottom-up methods start at the bottom of the lattice (with one-dimensional spaces) and iteratively join subspaces to higher 3dimensional subspace. To prune the search space bottom-up methods typically use clus-

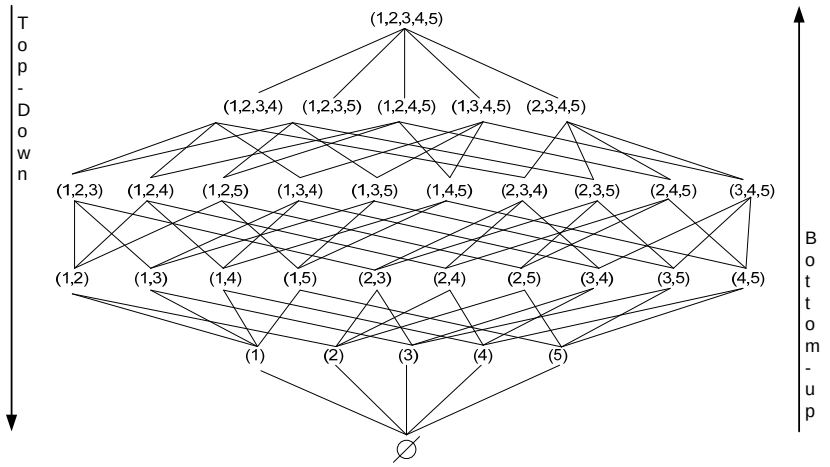

Figure 1.3: Lattice of subspace clusters

tering models which implement the downward closure property [KKZ07]. For density-based methods the density threshold has to be fixed to ensure the downward closure of clusters or subspaces. This assumption is often a problem as it leads to incomparable clustering results. We will discuss these effects in Chapter 2 in detail.

Top-down methods start by evaluating the full-dimensional space. In each step the current clustering result is refined by projecting individual clusters to lower dimensional spaces. The local neighborhood of a cluster in the high dimensional space is commonly used to determine the lower dimensional projection [KKZ07]. The quality of top-down methods often depends on the quality of the initial clustering in the full-dimensional space. Since clustering multi-dimensional data sets is often problematic projected clustering algorithms do not scale well to higher dimensional data. Further on, starting in high dimensional spaces is also not unproblematic as we discussed in the last chapter (see "curse of dimensionality").

1.3.1 Projected clustering

One approach to find clusters in multi-dimensional data sets is projected clustering. Projected clustering methods identify projections of the data into lower dimensional spaces where disjoint clusters may be identified. Projected clustering algorithms try to identify the relevant dimensions for each cluster individually. Typical projected clustering approaches extend partitioning

clustering methods by computing the relevant dimensions for each cluster.

PROCLUS (PROjected CLUStering) [AWY+99] extends the k-medoid algorithm by iteratively refining a full-space k-medoid clustering. Hence, PROCLUS works in a top-down manner. In each iteration the clusters are projected to a lower dimensional axis-parallel subspace and the data points are reassigned to their closest medoid.

ORCLUS (arbitrarily ORiented projected CLUSter generation) extends PROCLUS by using arbitrary projections for each cluster [AY00]. Both algorithms need the number of clusters and the (average) dimensionality of the clusters as input parameter.

LAC (Locally Adaptive Clustering) [DPGM04] weights the dimensions for each cluster by measuring the variance. The reassignment of points is based on the k-means approach and like ORCLUS and PROCLUS LAC starts with an initial full-space clustering before the individual weights per dimension are computed. The influence of the variance on the weight of a dimension is controlled by a user specified parameter.

Recently P3C (Projected Clustering via Cluster Cores) has been proposed [MSE06]. P3C works in a bottom-up manner by combining one-dimensional cluster cores to higher dimensional clusters. By using these cluster cores as an initialization for the EM algorithm, a partitioning is computed. As discussed before, clusters may overlap in different projections, and hence these approaches cannot detect all clusters.

Projected clustering algorithms are not able to find different clustering in overlapping subspaces. Reconsider the example data set presented in Figure 1.2. Since projected clustering algorithms only detect one clustering one of the two structures contained in this example would not be found. Hence, projected clusterings also lose some clusters if clusters are hidden in different projections.

1.3.2 Subspace clustering

To identify clusters in different projections subspace clustering methods aim at detecting clusters in any subspace. In principle, any clustering algorithm could be used to mine clusters in all subspaces of the data. This naive approach is of very high complexity as the number of subspaces is exponential in the number of dimensions and the result size is typically overwhelming. Consequently, subspace clustering approaches have largely focused on reduc-

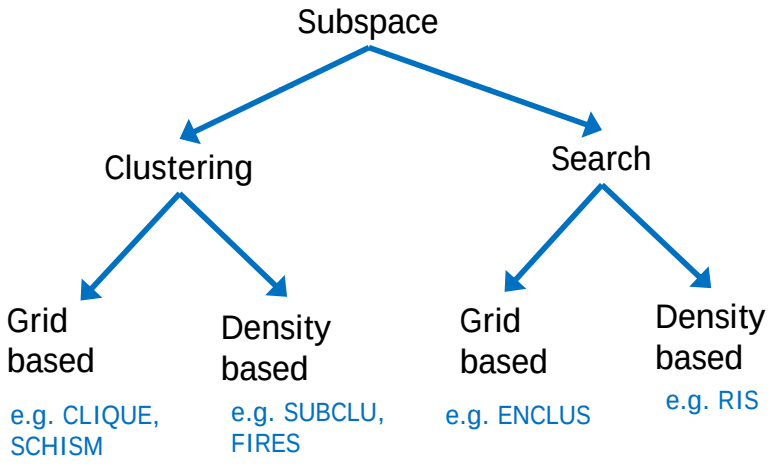

Figure 1.4: Categorization of existing subspace clustering concepts

ing the search space via heuristics, discretization of the data space via grids, etc. Another approach to efficiently compute subspace clusters is to search for relevant subspaces prior to clustering. A categorization of the different approaches is given in Figure 1.4.

Grid-based

CLIQUE (Clustering In QUEst) is the first subspace clustering method proposed in the literature. CLIQUE uses a grid to discretize the search space [AGGR98]. Monotonicity on the density of grid cells is used for pruning the search space in a bottom-up algorithm. Grids greatly reduce the computational complexity, yet clusters which spread across several cells might be missed. Moreover, CLIQUE measures density via simple counting of objects per cell. MAFIA [NGC99] extends CLIQUE via a data-adapted grid to reduce the number of clusters lost in discretization. Density is then computed depending on the size of a cell.

DOC and its variant FastDOC [PJAM02] are not directly based on a grid like discretization but define subspace clusters as dense hypercubes of a user defined size. The relevant dimensions for a subspace cluster are selected by using a Monte Carlo algorithm. An optimal subspace cluster is then defined by taking the number of points and the dimensionality of the subspace into account. Since clusters are defined using cells, clusters might also be cut apart. Further on, the density of arbitrarily shaped clusters is only

approximated by the hypercube as the actual distribution of points is not considered.

SCHISM (Support and Chernoff-Hoeffding bound-based Interesting Subspace Miner) [SZ04] extends CLIQUE using a variable threshold to cope with different dimensionalities, yet relies on heuristics and a grid-based discretization for pruning. Consequently, completeness is lost as in all grid-based approaches.

Density-based

Density-based clustering has been extended to subspace clustering in previous works. SUBCLU (density-connected Subspace Clustering) uses a gridless approach for effective subspace cluster mining [KKK04]. Using the earlier DBSCAN density notion, a density monotonicity property is used to prune subspaces. The algorithm uses an apriori like scheme (discussed first in association rule mining [AS94]) to detect subspace clusters in a bottom-up fashion. As dimensionality is ignored, it suffers from dimensionality bias, i.e. clusters cannot be separated from noise across subspaces (see also Chapter 2).

FIRES [KKRW05] is a generic framework for subspace clustering which allows using different clustering notions. It relies on approximative techniques in a filter-refinement scheme to scale to high dimensional spaces.

Subspace search

As mentioned above, faced with the huge number of subspace clusters different heuristics are used to efficiently determine subspace clusters. Subspace search methods use a two step approach. The first step searches for subspaces having a high cluster tendency. These subspaces are typically ranked using a scoring function. Actual subspace clusters are determined in a second step using traditional clustering methods.

ENCLUS [CWZZ99] (Entropy-Based Subspace Clustering) discretizes the data to compute the entropy and information gain of a subspace. Working bottom-up one-dimensional subspaces are evaluated first and then combined to higher dimensional subspaces. Entropy and information gain thresholds are used to prune the search space. The result is then clustered using a grid-based method like CLIQUE. An extended version of ENCLUS uses a

normalized entropy measure and an evolutionary approach to identify subspaces having a high cluster tendency [AKSS06].

RIS [KKKW03] (Ranking Interesting Subspaces) uses a density-based method to determine the interestingness of a subspace. Subspaces are ranked according to their ratio of actual and expected number of dense objects (core objects). Clusters are then mined using a density-based algorithm.

As subspace search methods do not mine the actual subspace clusters, the scoring function for subspaces does not necessarily reflect the differences in clusters contained:

1.4 Basic notions

In this section, we give the formal definitions for density-based subspace clustering as used in our algorithmic concepts. Density-based subspace clustering algorithms identify arbitrarily shaped clusters in multi-dimensional feature databases. For notational convenience let us first introduce some basic notions:

Definition 1.1 ***Preliminary Definitions***

We assume mining clusters in a data set based on the following definitions. Given:

- *a d-dimensional feature space with the corresponding index set* $\mathbf{D} = \{1, \ldots, d\}$
- *a universal domain* $\mathbf{U} = [0, \mathbf{v}]$ *for all dimensions*
- *a database* $\mathbf{DB} \subseteq \mathbf{U}^{|\mathbf{D}|}$ *containing* $|\mathbf{DB}| = n$ *objects* $o = (\nu_1 \ldots \nu_d)$

we define projections of data objects onto different subspaces based on:

- *the index set of a subspace as* $\mathbf{S} = \{s_1, \ldots, s_r\} \subseteq \mathbf{D}$
- *a subspace* $\mathbf{U}^{|\mathbf{S}|}$ *as the projection of* $\mathbf{U}^{|\mathbf{D}|}$ *to the r dimensions specified by the index set* $\mathbf{S}$
- $\mathbf{DB}^{|\mathbf{S}|}$ *as the projection of* $\mathbf{DB}^{|\mathbf{D}|}$ *to the dimensions in* $\mathbf{S}$
- *a projection of object* $o = (\nu_1 \ldots \nu_d)$ *to the subspace* $\mathbf{U}^{|\mathbf{S}|}$ *as* $o^{\mathbf{S}} = (\nu_{s_1} \ldots \nu_{s_r})$

In density-based clustering, *clusters* are defined as dense areas separated by sparsely populated areas. Hence, a basic definition for density-based clustering is a density measure which can be evaluated for every object. Usually the density value is determined by considering an area of influence surrounding the data object. This area of influence is specified using a norm $\|.\|$ and a parameter ε which defines the radius of the area:

Definition 1.2 ***Area of influence***
The area of influence $\mathbf{A}_\varepsilon$ *for a given norm* $\|.\|$

$$\|o\| : \mathbf{U}^{|\mathbf{D}|} \to \mathcal{R}$$

and a parameter ε *is defined as:*

$$\mathbf{A}_\varepsilon(o) = \{p \mid p \in \mathbf{DB}, \|p - o\| \leq \varepsilon\}$$

As mentioned before, in many applications clusters are hidden in subspaces and cannot be revealed by any cluster analysis that mines all dimensions simultaneously. Subspace clustering methods, on the other hand, aim at detecting clusters by automatically focusing to the respectively relevant subsets of the dimensions.

Hence, for subspace clustering it is essential that the density measure is capable of handling objects of different dimensionalities (see also Figure 1.5). In [KKK04] the standard notion of density is extended for the evaluation of subspaces. Following the paradigms of subspace clustering we extend $\|.\|$ to $\|.\|^{\mathbf{S}}$ by restricting the norm $\|.\| : \mathbf{U}^{|\mathbf{D}|} \to \mathcal{R}$ to the dimensions in subspace $\mathbf{S}$. For ease of notation, we refer to a subspace $\mathbf{U}^{|\mathbf{S}|}$ only by its index set $\mathbf{S}$. As a result, we obtain an adapted notion of the area of influence as the $(\mathbf{S}, \varepsilon)$-neighborhood. Figure 1.5 illustrates the projection of the area of influence for object o onto two different subspaces ($\mathbf{S} = \{1, 2\}$ and $\mathbf{S} = \{3\}$).

Definition 1.3 ***Subspace area of influence***
The area of influence $\mathcal{A}_\varepsilon^{\mathbf{S}}$ *in a subspace* $\mathbf{S}$ *and the respective norm* $\|.\|^{\mathbf{S}}$

$$\|o\|^{\mathbf{S}} = \left\|o^{\mathbf{S}}\right\| : \mathbf{U}^{|\mathbf{S}|} \to \mathcal{R}$$

is defined as:

$$\mathcal{A}_\varepsilon^{\mathbf{S}}(o) = \{p \mid p \in \mathbf{DB}, \|p - o\|^{\mathbf{S}} \leq \varepsilon\}$$

with ε *the influence parameter.*

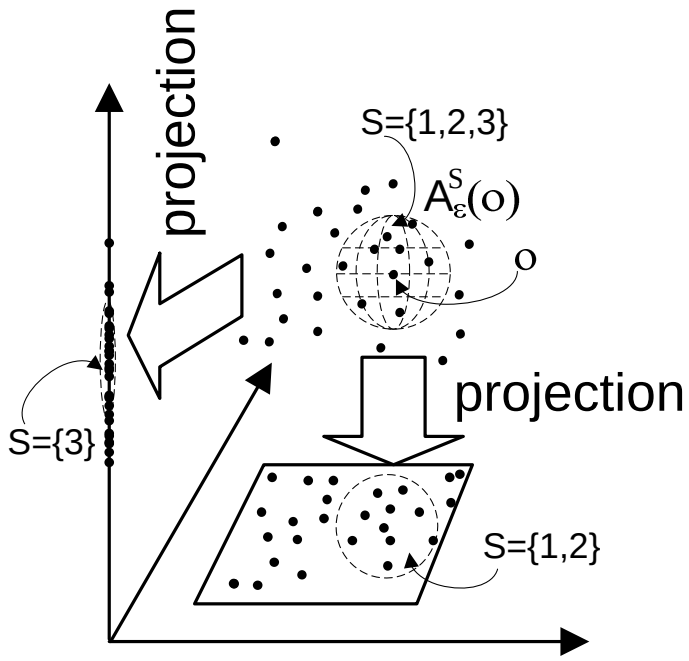

Figure 1.5: Projection and area of Influence

In density-based clustering, an object o is defined as dense if its density exceeds a threshold τ. Typically, density of an object o is determined by simply counting the number of objects in $\mathcal{A}^{\mathbf{S}}_{\varepsilon}(o)$. We generalize this idea by assigning weights to each object contained in $\mathcal{A}^{\mathbf{S}}_{\varepsilon}(o)$. This is depicted in Figure 1.6: the left neighborhood range shows a different distribution than the one on the right, even though both contain the same number of objects. By assigning less weight to objects further away, this effect is modeled in the density definition. Based on a monotonously falling weighting function $W : \mathcal{R} \to \mathcal{R}$ we define a density measure as:

Definition 1.4 ***Generalized Density Measure***
Let $\mathcal{W}$ be an arbitrary weighting function $\mathcal{W} : \mathcal{R} \to \mathcal{R}$. Based on $\mathcal{W}$ a generalized density measure $\varphi^{\mathbf{S}}_{\varepsilon}(o)$ for an object o in subspace $\mathbf{S}$ is defined as:

$$\varphi^{\mathbf{S}}_{\varepsilon}(o) = \sum_{p \in \mathcal{A}^{\mathbf{S}}_{\varepsilon}(o)} \mathcal{W}\left(\|p - o\|^{\mathbf{S}}\right)$$

Using a weighting function $\mathcal{W}$ a density measure $\varphi^{\mathbf{S}}_{\varepsilon}(o)$ weights the objects within the neighborhood according to their distance from the object

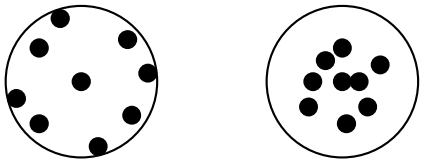

Figure 1.6: Density distribution within neighborhood

o. Consequently, an object o in subspace $\mathbf{S}$ is called dense if the weighted objects contained in its area of influence sum up to more than a given density threshold τ. The weight of an object contained in the area of influence is determined by a norm which reflects the distance of the object from the point of evaluation.

Definition 1.5 ***Subspace Density-Connected.***
An object o is dense in subspace $\mathbf{S}$ ($\mathbf{S}$-dense) with respect to the area of influence ε if its density exceeds the density-threshold τ:

$$\mathbf{S}\text{-}dense_{\varepsilon}^{\tau}(o) \;\Leftrightarrow\; \varphi_{\varepsilon}^{\mathbf{S}}(o) \geq \tau$$

A subset $\mathbf{C} \subseteq \mathbf{DB}$ is connected with respect to a subspace $\mathbf{S}$ ($\mathbf{S}$-connected$_{\mathbf{C}}$) if there is a chain of neighboring objects between all pairs of objects $(p, q) \in \mathbf{C}$:

$$\begin{aligned} \forall (p,q) \in \mathbf{C} \;\; : \;\; & \mathbf{S}\text{-}connected_{\mathbf{C}}(p,q) \;\Leftrightarrow \\ & \exists\, o_1 \ldots o_n \in \mathbf{C} : \\ & \quad o_1 = p, o_n = q \;\wedge \\ & \quad \forall\; i = 1 \ldots n-1 : \|o_i - o_{i+1}\|^{\mathbf{S}} \leq \varepsilon. \end{aligned}$$

A density-based cluster is then defined as the transitive closure of all dense connected objects, i.e. the maximal set of objects which are *density-connected*. Clusters thus are elements of chains of objects which are mutually included in one another's neighborhoods. Consequently, arbitrarily shaped clusters can be successfully detected even in noisy settings [EKSX96].

Definition 1.6 ***Density-based Subspace Cluster***
Let $\mathbf{U}$ be a domain, $\mathbf{S} \subseteq \mathbf{D}$ be an index set denoting a subspace $\mathbf{U}^{\mathbf{S}}$ of $\mathbf{U}^{\mathbf{D}}$. A subspace cluster $(\mathbf{C}, \mathbf{S})$ is a set of objects $\mathbf{C} \subseteq \mathbf{DB}$ for which the following holds:

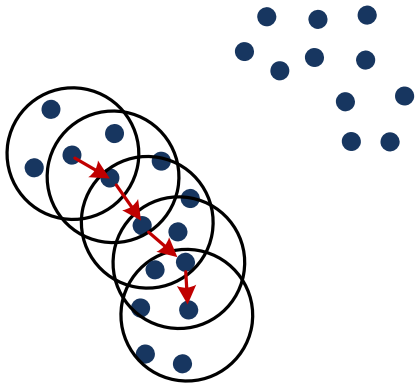

Figure 1.7: Density-based clustering

- *All objects in* $\mathbf{C}$ *are* $\mathbf{S}$***-dense****:*
  $\forall o \in \mathbf{C}: \ \mathbf{S}\text{-}dense^{\tau}_{\varepsilon}(o)$

- *All objects in* $\mathbf{C}$ *are* $\mathbf{S}$***-connected***
  $\forall p, q \in \mathbf{C} : \mathbf{S}\text{-}connected_{\mathbf{C}}(p, q)$

- *Cluster* $\mathbf{S}$ *is* ***maximal*** *in subspace* $\mathbf{S}$
  $\forall p \in \mathbf{DB} : \mathbf{S}\text{-}dense^{\tau}_{\varepsilon}(p) \wedge (\exists q \in \mathbf{C} : \mathbf{S}\text{-}connected_{\mathbf{C}}(p, q)) \Rightarrow p \in \mathbf{C}$

Definition 1.6 introduces subspace clusters as maximal density-connected sets of objects with respect to a subspace $\mathbf{S}$ of the universe $\mathbf{U}^{\mathbf{S}}$. Let us note that an individual object o may belong to different subspace clusters $(\mathbf{C_1}, \mathbf{S_1})$ and $(\mathbf{C_2}, \mathbf{S_2})$. This does not contradict maximality as long as the clusters focus on different subspaces $\mathbf{S_1} \neq \mathbf{S_2}$. If an object is not contained in any subspace cluster $(\mathbf{C_i}, \mathbf{S_i})$ the object is termed noise.

SUBCLU [KKK04] first introduced the extension of the density-based clustering model of DBSCAN [EKSX96] to subspaces. In each subspace, objects must contain a minimal number of objects in its area of influence to be defined as core objects (to be dense). Similar to DBSCAN, SUBCLU distinguishes between core objects and border objects. Border objects may belong to multiple clusters, even in the same subspace, while core objects define the area belonging to a cluster.

Our definition of density generalizes the idea of core objects to dense objects with respect to a subspace $\mathbf{S}$ and a threshold τ. In SUBCLU all

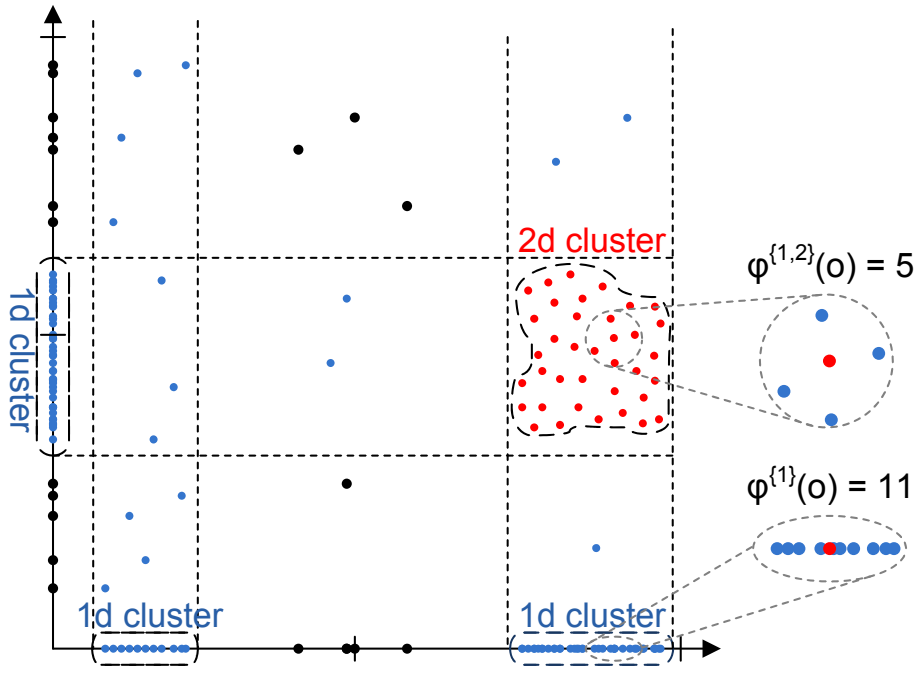

Figure 1.8: Density-based Subspace Clusters

objects in $\mathcal{A}_\varepsilon^{\mathbf{S}}(o)$ are assigned the same weight: $\mathcal{W}(x) = 1, \forall x \in \mathcal{R}$. Hence, to determine the density of an object the number of points contained in the area of influence is counted. Thus an object is termed dense if more than $\tau = minPoints$ objects are contained in area of influence. To determine the area of influence SUBCLU uses the Euclidean distance. As this is a common approach we also focus on the Euclidean norm, but other norms could be used as well:

$$\|p - o\|^{\mathbf{S}} = \sqrt{\sum_{i \in \mathbf{S}} (p_i - o_i)^2}$$

Figure 1.8 illustrates a basic subspace clustering scenario. Clusters i.e. dense regions are identified in all possible subspace. In this example four different subspace clusters are identified. The area of influence surrounding an object is determined by a norm. E.g. in SUBCLU object o would be determined as dense in subspace $\mathbf{S} = \{1, 2\}$ if the density threshold τ is set to 5 or less (see Figure 1.8). Table 1.1 summarizes our notations.

$\mathbf{v}$	the maximal possible value for all domains
$\mathbf{U}$	a universal domain $\mathbf{U} = [0, \mathbf{v}]$ for all dimensions
$\mathbf{D}$	the complete data space $\mathbf{D} = \{1, \ldots, d\}$
$\mathbf{S}$	a index set of a subspace $\mathbf{S} = \{s_1, \ldots, s_r\} \subseteq \mathbf{D}$
$\mathbf{DB}$	the database containing n objects: $\mathbf{DB} \subseteq \mathbf{U}^{\lvert\mathbf{D}\rvert}$
o	an objects $o \in \mathbf{DB}$ with d attributes $o = (o_1, \ldots_d)$
$o^{\mathbf{S}}$	projection of objects o to subspace $\mathbf{S}$: $o = (o_1, \ldots, o_d)$
$\lVert o \rVert^{\mathbf{S}}$	the norm of an object o in subspace $\mathbf{S}$
$\mathcal{W}(x)$	a weighting function for $x \in \mathcal{R}$
ε	the parameter specifying the size of the area of influence
$\mathcal{A}_\varepsilon^{\mathbf{S}}(o)$	the objects contained in the area of influence of object o in subspace $\mathbf{S}$
$\varphi_\varepsilon^{\mathbf{S}}(o)$	a density measure for an object o based on a weighting function $\mathcal{W}$, a norm $\lVert . \rVert$ and the respective area of influence $\mathcal{A}_\varepsilon^{\mathbf{S}}(o)$
τ	the threshold parameter for the density measure: $gd(o) \geq \tau$

Table 1.1: Notation for subspace clustering

Chapter 2

Unbiased density-based subspace clustering

As discussed in the last chapter, in scenarios with many attributes or noise, clusters are often hidden in subspaces of the data and do not show up in the full dimensional space. For these applications, subspace clustering methods aim at detecting clusters in any subspace. Existing subspace clustering approaches fall prey to an effect we call dimensionality bias. As dimensionality of subspaces varies, approaches which do not take this effect into account fail to separate clusters from noise.

In this chapter, we focus on eliminating the dimensionality bias. We give a formal definition of dimensionality bias and analyze consequences for subspace clustering. A new density-based subspace clustering approach (DUSC) based on statistical foundations is proposed which takes the dimensionality of the respective subspace into account. We show that this method eliminates dimensionality bias and leads to comparable clustering results between subspaces of different dimensionalities. In thorough experiments on synthetic and real world data sets, we demonstrate that our new dimensionality unbiased subspace clustering models clearly outperforms existing subspace clusterings in terms of accuracy.

2.1 Introduction

Density-based clustering algorithms are capable of detecting meaningful patterns in many applications. As described in the last chapter the density-based

clustering paradigm has been extended to subspaces clustering in previous work. However, the straight forward extension of density estimators to subspaces ignores the dimensionality of the investigated subspace.

In this chapter, we prove that density measures which ignore the dimensionality of the subspace are biased. Assuming a simple setup of uniformly distributed data, we show that biased density measures cannot distinguish this pseudo-cluster scenario from true clusters in all subspaces. As a consequence, dimensionality bias means failing at the very core of density-based subspace clustering. We demonstrate that existing dimensionality independent approaches check incomparable density values against the same constant threshold. Hence, these approaches fail from separating clusters from noise. Depending on the setting of the fixed density threshold existing subspace clustering algorithms either lose clusters or detect numerous pseudo-clusters. These effects have serious consequences for the quality of the result.

Summing up, our contributions include:

- definition and analysis of dimensionality bias and its consequences for subspace clustering
- definition of density-based clustering on statistical foundations
- dimensionality unbiased subspace clustering model

This chapter is structured as follows: we shortly review density measures of existing subspace clustering algorithms in the following section. Then, in Section 2.3, we discuss dimensionality bias. A novel model of density-based subspace clusters is defined. We demonstrate that this definition perfectly eliminates dimensionality bias. Analysis of our model is exploited to derive powerful pruning properties which do not jeopardize accuracy. We demonstrate the usefulness and effectiveness of our approach in thorough experiments on both synthetic and real world data sets in the experiments Section 2.7.

2.2 Related work

In this section we shortly review the density measures of exiting clustering methods. Many existing subspace clustering algorithms do not adapt the density model to the dimensionality. For example, CLIQUE [AGGR98]

partitions the data space into equi-width cells and computes the number of objects per cell. A cell is dense if it contains more than a specified number of objects. This is biased as the ratio of the volume of cells to the overall volume decreases exponentially with the dimensionality. Similar cluster models have been used in MAFIA [NGC99], CBF [CJ02] and DOC [PJAM02]. All methods use a dimensionality independent threshold to determine if a cell is dense.

As mentioned in the last chapter, SUBCLU [KKK04] determines the density of an object by counting the number of objects contained in the ε-neighborhood. As objects in higher dimensional spaces are more spread out, virtually no high dimensional clusters are found.

Simply comparing the density of subspaces of different dimensionalities according to the same density measure ignores the effect of dimensionalities on density. The higher dimensional the subspace, the lower its expected density. More precisely, evaluating a set of objects in a higher dimensional subspace far less likely to be found a density-based cluster. This must be taken into account when defining a density measure for subspace. Two recent approaches, FIRES [KKRW05] and SCHISM [SZ04] use dimensionality dependent density measures, yet do not overcome dimensionality bias. FIRES uses an approximation to combine one dimensional clusters to high dimensional clusters. SCHISM uses heuristics to prune low dimensional subspace clusters and computes approximate densities on a per-cell basis.

Projected clustering algorithms typically take the dimensionality of the subspace into account but do not analyze the effect of different dimensionalities on the clustering result. PROCLUS [AWY+99] for example measures the segmental Manhattan distance (the distance relative to the number of dimensions) between an object and the center of a cluster. ORCLUS [AY00] measures the projected energy of a clustering which also considers the dimensionality of a subspace. P3C [MSE06] uses a Poisson threshold which depends on the dimensionality of the investigated subspace. However, no clustering algorithms investigate if the clusterings from different dimensionalities are comparable.

2.3 Dimensionality bias

Subspace clustering methods analyze data spaces of different dimensionalities. Consequently, avoiding an effect which we call dimensionality bias is an important issue. Dimensionality bias refers to a dependency of density on the dimensionality of the subspace: as dimensionality increases, average distances between objects increase and cluster radii grow. At the same time, the expected density within the area of influence drops accordingly. Thus, ignoring the dependency of density on the dimensionality of the subspace leads to incomparable density values.

Incomparable density values pose the following problem: the high discrepancy in density scales of low dimensional or high dimensional subspaces makes it impossible to find a suitable parameter for a fixed density threshold τ. If on the one hand τ is parameterized such that high dimensional clusters with low expected density are detected then numerous excess pseudoclusters are generated in low dimensional spaces where expected density is high. On the other hand, a parameterization of τ which separates clusters from noise in low dimensional spaces loses clusters in high dimensional spaces.

We call a density measure to be dimensionality unbiased if the expected density value of the density measure is independent of the dimensionality of the subspace. Statistically speaking, this corresponds to the same expected density value regardless of the dimensionality of the subspace. For a generalized density measure as proposed in Definition 1.4 this statement is formalized as:

Definition 2.1 ***Dimensionality Unbiased Density Measure***
A density measure $\varphi_\varepsilon^{\mathbf{S}}$ is dimensionality unbiased if its expected density is the same for any two subspaces $\mathbf{S}_1$ and $\mathbf{S}_2 \subseteq \mathbf{D}$:

$$\forall\ \mathbf{S}_1, \mathbf{S}_2 : \quad E\left[\varphi_\varepsilon^{\mathbf{S_1}}\right] = E\left[\varphi_\varepsilon^{\mathbf{S_2}}\right]$$

i.e. the expected density is varying over subspaces of different dimensionalities.

Definition 2.1 formalizes the notion of dimensionality unbiased density measures. It states that bias leads to different expected densities for the

same object depending on the subspace. It is crucial for density-based approaches to separate dense from sparse regions. This is infeasible using a biased density measure. For example, uniformly distributed data does not contain any dense or sparse regions as all regions have the same density. Hence, it does not contain any cluster. The density value computed by a biased density measure for a uniformly distributed space varies for different dimensionalities. Consequently, for a biased density measure it is not clear how to specify the density threshold for separating dense regions from noise. An unbiased density measure does not consider a uniformly distributed region as dense if the density threshold is above the expected density value. Thus an unbiased density measure is capable of distinguishing dense and sparse regions in subspaces of different dimensionalities.

We now show how dimensionality bias can be eliminated for any density estimator. As the expected density should be the same for any two subspaces, we normalize density estimators with their expected density.

Theorem 2.1 ***Eliminating dimensionality bias.***
For any density measure $\varphi_\varepsilon^{\mathbf{S}}$, the weighted density measure:

$$\frac{1}{E[\varphi_\varepsilon^{\mathbf{S}}]}\varphi_\varepsilon^{\mathbf{S}}$$

is dimensionality unbiased.

Proof. With linearity property of the expectation value, the proof is straightforward:

$$\forall\, \mathbf{S} \subseteq \mathbf{D} : E\left[\frac{1}{E\left[\varphi_\varepsilon^{\mathbf{S}}\right]}\varphi_\varepsilon^{\mathbf{S}}\right] = \frac{1}{E\left[\varphi_\varepsilon^{\mathbf{S}}\right]}E\left[\varphi_\varepsilon^{\mathbf{S}}\right] = 1$$

Thus, for any two subspaces, normalizing the density measure by the expected value of the subspace yields comparable density values for any two subspaces $\mathbf{S}_1$ and $\mathbf{S}_2$. Normalization could be achieved by other means such as subtracting the expected value, but, as we will see later, dividing by the expected value simplifies the choice of density parameters in subspace clustering.

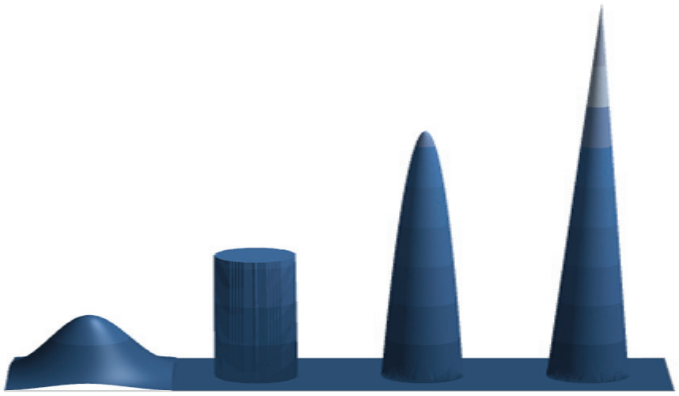

Figure 2.1: Gauss, Rectangular, Epanechnikov and Triangular Kernel (from left to right)

2.4 An unbiased density estimator

In this section we use statistical analysis to develop an unbiased density measure for subspace clustering. In statistics, *kernel estimators* are used to estimate density functions from a set of data objects. A kernel weights the observations in the data set to compute the density value at any position in the data space. Kernel estimators are used to estimate a probabilistic density function and hence the weighting function (called kernel function K) satisfies the condition $\int_{-\infty}^{+\infty} K(x)dx = 1$. Different kernel functions correspond to differently shaped curves, resulting in slightly different density assessments. Using a rectangular kernel objects within the area of influence are just counted which would correspond to the SUBCLU approach. Hence the density computation tends to be overly sensitive to the choice of the area of influence. Using a kernel function which assigns higher values to closer objects and lower values to objects further away, density is more accurately measured for many applications than by mere counting of objects within the ε-neighborhood [HK98, Sil86].

The most commonly used ones are Gauss, Epanechnikov, Bisquare and Triangular kernels (see also Figure 2.1). Please note that the volume of statistical kernels is normalized to one and consequently the kernels depicted in Figure 2.1 have a different height. Any of these kernels could be used in principle for density estimation. Gauss, however, assigns non-zero values to all objects in the database. This makes it is a poor density estimator in terms

of efficiency as the area of influence always contains the entire database. Silverman et al. also prove that the Gauss kernel is also less effective for density estimation than other kernels [Sil86].

The Epanechnikov kernel is both an efficient and effective choice, since it is computationally efficient and minimizes the mean integrated squared error [Sil86]. Thus, we use Epanechnikov kernel in the following, but in principle any kernel could be used as well. Within an area of influence, the Epanechnikov kernel assigns decreasing weights to objects with increasing distance.

For a subspace $\mathbf{S}$, the Epanechnikov kernel function $K^{\mathbf{S}}$ is defined as:

$$K^{\mathbf{S}}(x) = \begin{cases} \frac{|\mathbf{S}|+2}{2c_{|\mathbf{S}|}} \left(1 - \left(\|x\|^S\right)^2\right), & \|x\|^{\mathbf{S}} \leq 1 \\ 0, & \text{else.} \end{cases} \tag{2.1}$$

where $|\mathbf{S}|$ denotes the dimensionality of the subspace and $c_{|\mathbf{S}|} = \frac{\pi^{|\mathbf{S}|/2}}{\Gamma(|\mathbf{S}|/2+1)}$ is the volume of the $|\mathbf{S}|$-dimensional unit sphere and the gamma function is defined by $\Gamma(n+1) = n * \Gamma(n), \Gamma(1) = 1, \Gamma(1/2) = \sqrt{\pi}$.

Each kernel is scaled in width according to a *bandwidth* ε which corresponds to the area of influence of a density-based subspace clustering algorithm. For subspace clustering, we need only the Epanechnikov kernel weights $\left(1 - \left(\|x\|^{\mathbf{S}}\right)^2\right)$ to obtain the following density measure with its respective weighting function (see Definition 1.4):

Definition 2.2 ***Epanechnikov Density Measure***
Let $\mathcal{W}(t) = 1 - t^2$ be the Epanechnikov weighting function. We define the Epanechnikov density measure for an area of influence given by ε as:

$$\varphi_{\varepsilon}^{\mathbf{S}}(o) = \sum_{p \in \mathcal{A}_{\varepsilon}^{\mathbf{S}}(o)} \left(1 - \left(\frac{\|o-p\|^{\mathbf{S}}}{\varepsilon}\right)^2\right)$$

Following Theorem 2.1, we can remove dimensionality bias by taking the expected density for subspaces into account. As clustering aims at detecting dense regions in a given data set, clusters should have higher density values

than data without any clusters. A data set without clusters corresponds to uniformly distributed data, i.e. all values are taken with the same probability. By requiring that density should exceed the expected density of uniformly distributed subspaces, we ensure that no pseudo-clusters are "detected".

To remove dimensionality bias we examine a database containing n objects which are uniformly distributed in space. Following the definition of a uniform distribution each position in space has the same density. Thus the density f for an object in a uniform and identical distributed space is independent and equal for each point with: $f(x) = \frac{1}{\mathbf{v}^{|\mathbf{S}|}}$.

Hence, the expected density for $\varphi_\varepsilon^{\mathbf{S}}(o)$ can be computed by integrating over the weighting function for all possible positions in all dimensions multiplied by their probability density. From statistics, we have that any kernel function is normalized to one: $\int\limits_{x \in \mathcal{R}^{|\mathbf{S}|}} K^{\mathbf{S}}(x)dx = 1$. Hence, for the Epanechnikov Density measure we can derive (compare Equation 2.1):

$$\int\limits_{\substack{x \in \mathcal{R}^{|\mathbf{S}|} \\ \|x\| \leq 1}} \left(1 - \left(\|x\|^{\mathbf{S}}\right)^2\right) dx = \frac{2c_{|\mathbf{S}|}}{|S| + 2} \tag{2.2}$$

For a given database containing n uniformly distributed objects ($f(x) = \frac{1}{\mathbf{v}^{|\mathbf{S}|}}$) the expected density $E\left[\varphi_\varepsilon^{\mathbf{S}}(o)\right]$ for an object o can be computed by:

$$\begin{aligned} E\left[\varphi_\varepsilon^{\mathbf{S}}(o)\right] &= \int\limits_{\substack{x \in \mathcal{R}^{|\mathbf{S}|}, \\ \|o-x\| \leq \varepsilon}} \sum_{i=1}^{n} \left(1 - \left(\frac{\|o-x\|^{\mathbf{S}}}{\varepsilon}\right)^2\right) \cdot f(x)\ dx \\ &= n \cdot \int\limits_{\substack{x \in \mathcal{R}^{|\mathbf{S}|}, \\ \|o-x\| \leq \varepsilon}} \left(1 - \left(\frac{\|o-x\|^{\mathbf{S}}}{\varepsilon}\right)^2\right) \cdot \frac{1}{\mathbf{v}^{|\mathbf{S}|}}\ dx \end{aligned} \tag{2.3}$$

To solve the integral we substitute $\frac{\|o-x\|^S}{\varepsilon}$ by $\|t\|$. Since t is a multivariate $|\mathbf{S}|$-dimensional variable we have to substitute $dx = \varepsilon^{|\mathbf{S}|}dt$. After substitution, we can apply Equation 2.2 from above to solve the integral.

$$E\left[\varphi_\varepsilon^{\mathbf{S}}(o)\right] = n \cdot \int_{\substack{t \in \mathcal{R}^{|\mathbf{S}|}, \\ \|t\| \leq 1}} \left(1 - \|t\|^2\right) \cdot \frac{\varepsilon^{|\mathbf{S}|}}{\mathbf{v}^{|\mathbf{S}|}}\, dt$$
$$= n \cdot \frac{\varepsilon^{|\mathbf{S}|} \cdot \left(2c_{|\mathbf{S}|}\right)}{\mathbf{v}^{|\mathbf{S}|}\left(|\mathbf{S}| + 2\right)}$$

Note that since in (subspace) clustering prior knowledge about the data distribution is typically not available and not reasonable, assuming uniform data distribution as a baseline comparison is reasonable. The above reasoning, however, holds for other density distributions as well.

By applying Theorem 2.1 on the Epanechnikov density measure $\varphi_\varepsilon^{\mathbf{S}}$ we obtain the unbiased Epanechnikov density measure $\frac{1}{E[\varphi_\varepsilon^{\mathbf{S}}]}\varphi_\varepsilon^{\mathbf{S}}$:

Definition 2.3 ***Unbiased Epanechnikov Density Measure***
The unbiased density measure for the Epanechnikov influence function $\varphi_\varepsilon^{\mathbf{S}}$ is given by

$$\frac{1}{\alpha(\mathbf{S}, \varepsilon)}\varphi_\varepsilon^{\mathbf{S}}(o) \quad \textit{with}$$

$$\alpha(\mathbf{S}, \varepsilon) = E_{\mathbf{S}}\left[\varphi_\varepsilon^{\mathbf{S}}(o)\right] = \frac{2n\varepsilon^{|\mathbf{S}|}c_{|\mathbf{S}|}}{\mathbf{v}^{|\mathbf{S}|}(|\mathbf{S}| + 2)}$$

Dimensionality bias can be removed for other kernel density estimators as well by normalizing the density measure with the reciprocal expected density. We briefly illustrate how to remove the dimensionality bias from the rectangular kernel in Chapter 3.3. Using kernels known from statistics has two advantages: the effectiveness of kernel estimators has been studied in theoretical and practical settings, and computation of the expected density for probability density functions follows standard methods.

2.5 DUSC subspace clustering

We introduced an unbiased density measure for subspace clustering. In this section, we define our subspace clustering model DUSC (dimensionality unbiased subspace clustering) based on three important properties. First, since the number of possible subspace projections is exponential in the number of

dimensions subspace clustering algorithms often produce numerous ***redundant*** subspace clusters. To avoid excessive cluster outputs which contain essentially the same information repeated in different dimensionalities, we restrict the result to subspace clusters of the highest possible dimensionality. Second, the so-called "***empty space***" effect in high dimensional spaces means that objects are spread out extremely, leading to useless expected density values. This could result in single objects being detected as pseudo-clusters. And third, parameters should be ***intuitive*** in the sense that their effect on the result can be clearly stated.

Intuitive density threshold

The density threshold is a core parameter since it sets the dividing line between dense objects and noise. As this parameter has to be set by the user it is important for users to have an intuitive understanding of this parameter. Commonly, users do not know density distribution apriori, which makes the choice of a density value difficult. We exploit the fact that in our approach density is measured with respect to the expected density as discussed before. Consequently, users do not need to specify absolute density thresholds, but only a factor by which the expected density has to be exceeded. Following the definition in the previous section, an object o is dense in subspace $\mathbf{S}$ according to the expected density $\alpha(\mathbf{S}, \varepsilon)$ iff:

$$\frac{1}{\alpha(\mathbf{S}, \varepsilon)} \varphi_{\varepsilon}^{\mathbf{S}}(o) \geq \mathbf{F} \tag{2.4}$$

where $\mathbf{F}$ denotes the density threshold. As the density factor $\mathbf{F}$ is independent of the dimensionality and data set size, it is much easier to specify than traditional density thresholds. Moreover, we demonstrate in the experiments that this parameter is robust with a setting of $\mathbf{F} > 50$ for many applications.

Empty space problem.

With increasing dimensionality the expected density and hence the expected number of objects contained in an area of influence drops exponentially [BGRS99]. This effect is termed "empty space problem" in statistics [Sil86]. For subspace clustering this means that compared to the expected density an object may be determined as dense even if the area of influence is nearly devoid of observations, resulting in pseudo-dense single objects.

To remove pseudo-dense objects we introduce a specific density constraint. This constraint ensures that the density value of a dense object is not only $\mathbf{F}$ times more dense than expected, but additionally exceeds the expected density of η objects in the area of influence. The expected density value $E_\eta\left[\frac{1}{\alpha(\mathbf{S},\varepsilon)}\varphi_\varepsilon^{\mathbf{S}}(o)\right]$ of an object o which contains η objects in the area of influence can be derived in a manner similar to the reasoning in the previous section. The probability of an object x at an arbitrary position within the area of influence of o is $f(x) = \frac{1}{c_{|\mathbf{S}|}\varepsilon^{|\mathbf{S}|}}$. Hence the density constraint removing pseudo-dense objects can be formalized for the Epanechnikov kernel as follows:

$$\begin{aligned}
\frac{1}{\alpha(\mathbf{S},\varepsilon)}\varphi_\varepsilon^{\mathbf{S}}(o) &\geq E_\eta\left[\frac{1}{\alpha(\mathbf{S},\varepsilon)}\varphi_\varepsilon^{\mathbf{S}}(o)\right] \\
\varphi_\varepsilon^{\mathbf{S}}(o) &\geq E_\eta\left[\varphi_\varepsilon^{\mathbf{S}}(o)\right] \\
\varphi_\varepsilon^{\mathbf{S}}(o) &\geq \eta \int\limits_{\substack{o\in\mathcal{R}^{|\mathbf{S}|} \\ \|o-x\|\leq\varepsilon}} f(x)\left(1-\left(\frac{\|o-x\|^{\mathbf{S}}}{\varepsilon}\right)^2\right)dx \\
\varphi_\varepsilon^{\mathbf{S}}(o) &\geq \eta\cdot\frac{1}{c_{|\mathbf{S}|}\cdot\varepsilon^{|\mathbf{S}|}}\cdot\frac{2\cdot c_{|\mathbf{S}|}\cdot\varepsilon^{|\mathbf{S}|}}{|S|+2} \\
\varphi_\varepsilon^{\mathbf{S}}(o) &\geq \eta\cdot\frac{2}{|S|+2}
\end{aligned}$$

Hence, let us introduce the pseudo density factor ω by the following definition:

$$\Rightarrow \omega(\mathbf{S}) := \frac{2}{|S|+2} \tag{2.5}$$

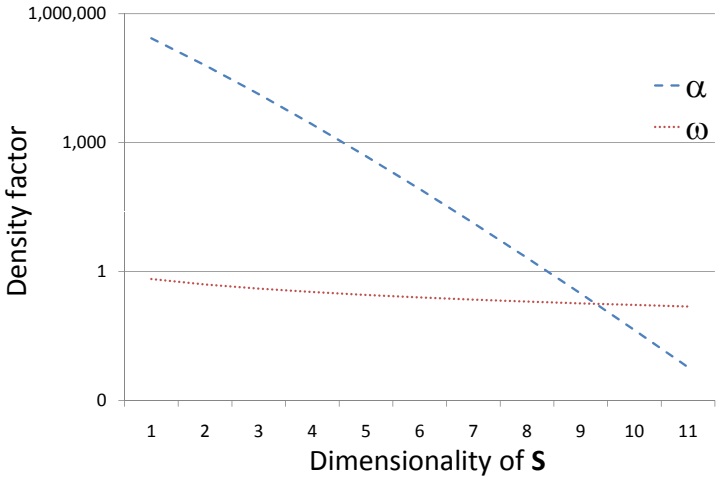

Figure 2.2: Density thresholds α and ω

To guarantee that objects are not considered dense if the ε sphere is virtually empty, a very small value for η is sufficient (generally two or three). Users typically do not need to change this value. Our new density-based subspace clustering model below combines the density constraints α and ω given in formula 2.4 and 2.5.

Figure 2.2 illustrates the effect of increasing dimensionality on the two density thresholds α and ω [for parameter setting $\eta = 2; \varepsilon = 0.2; v = 1; n = 100000$]. The expected density α decreases exponentially with the dimensionality, while ω (the expected density of η objects) only depends linearly on the dimensionality of the subspace (please note the logarithmic scale). The very low value for α in higher dimensions confirms the *empty space* problem. The α threshold thus applies to most dimensionalities, while ω concerns only very high dimensional spaces. The parameters α and ω combined ensure an unbiased density notion without defining objects in nearly empty regions as dense.

Redundancy

Subspace clustering algorithms search for clusters in all possible projections of the data. As lower dimensional projections of a subspace cluster often exhibit additional clusters in the respective subspace the number of subspace clusters is often huge (typically even more subspace clusters exist than

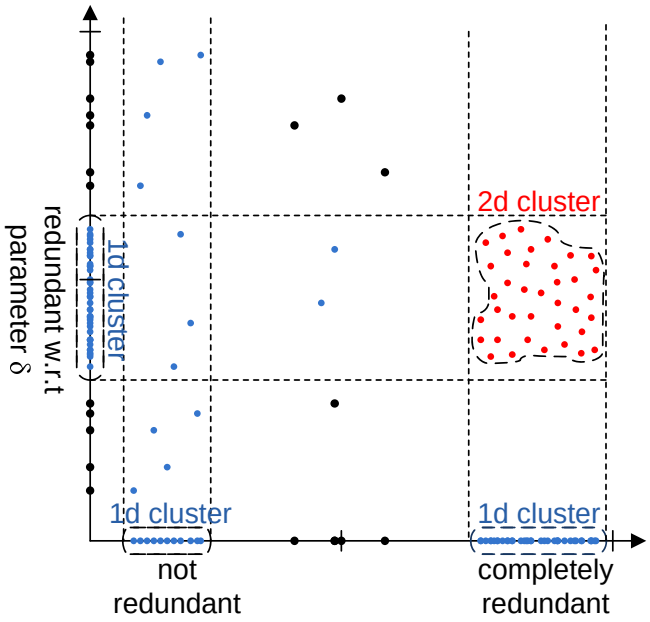

Figure 2.3: Redundancy in subspace clusterings

objects). However, the lower dimensional projections of a cluster typically reflect the same correlation and hence do not contain any additional information. We call lower dimensional projections of a density-connected set redundant if (nearly) the same object set is clustered again.

Figure 2.3 illustrates a possible subspace clustering result for a two-dimensional data set. In this example one projection of the two-dimensional cluster (illustrated by the red points) is completely redundant as the same objects are clustered.

To obtain non-redundant result sets we propose to remove redundant clusters. Hence, a cluster in subspace $\mathbf{S}$ is removed if the respective set of objects $\mathbf{C}$ is already clustered in a higher dimensional projection $\mathbf{S}'$ ($\mathbf{S} \subseteq \mathbf{S}'$). By projecting a cluster to a lower dimensional space some *noise* objects may intersperse with the cluster objects (see also Figure 2.3). To control the degree of noise which may be assigned to a lower dimensional redundant cluster we introduce a redundancy parameter δ. Depending on the setting of the parameter δ the one-dimensional projection to the y-axis of the two-dimensional cluster depicted in Figure 2.3 is also redundant.

We studied redundant subspace clusters using different real world data sets. Figure 2.4 presents the objects and relevant dimensions of two clusters

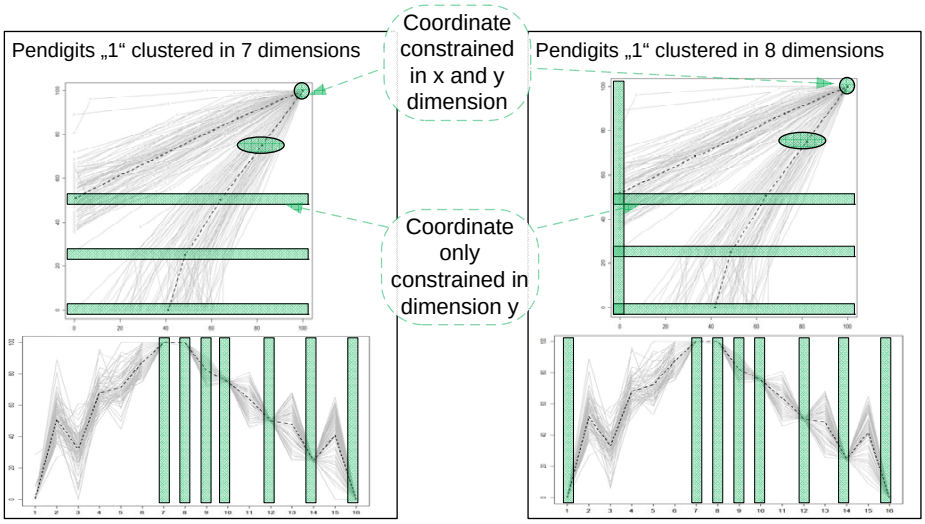

Figure 2.4: Redundant clustering result (Same objects representing the same structure clustered in two different subspaces; left part using 7 dimensions, right part using 8 dimensions)

obtained by clustering the pendigits data set (see also Chapter 2.7). The pendigit data set contains information about the pen movement of handwritten digits. At 8 different time points the (x, y) position of the pen is recorded, creating a 16-dimensional data set. The upper part of Figure 2.4 illustrates the graphical representation of the clustered objects in the geometric space as they were actually drawn on the pen tablet.

Parallel coordinates [Ins85], a more generally applicable technique for visualizing high dimensional data, are used in the lower part of Figure 2.4 to illustrate the clusters. Each d-dimensional point is represented by a polyline which intersects d parallel axes. The position where the polyline intersects the i-axes correspond to the value of the point in the i-dimension. The dashed black line represents the mean value of the cluster. The coordinates of the relevant dimensions belonging to the subspace cluster are marked by green boxes.

In the geometric space (upper parts of Figure 2.4) we mark the relevant dimensions by green ellipses. If one dimension of a (x, y) coordinate is not constrained a box is used to represent the relevant dimension (objects may differ in the other dimension).

According to the definition of traditional subspace clustering models as

SUBCLU [KKK04] both object sets presented in Figure 2.4 are defined as clusters. Both clusters only contain objects representing the digit "1" which can also be seen in the visual representation (e.g. the dashed black line indicating the mean value). The left cluster is using a subspace of 7 dimensions (illustrated by the green boxes). The right cluster uses one additional dimension (the x-value of the first point). As we can see nearly the same cluster is found if 8 instead of 7 dimensions are used for clustering. Since both clusters describe the same correlation users are typically only interested in one of the two clusters. We define the most descriptive one (the right one) as relevant and the other one as redundant. The following redundancy definition solves this problem by removing redundant cluster (as the left cluster in Figure 2.4) from the result:

Definition 2.4 ***Redundancy***

A clustered set of objects $\mathbf{C} \subseteq \mathbf{DB}$ *in subspace* $\mathbf{S} \subseteq \mathbf{D}$ *is redundant w.r.t. the redundancy parameter* δ *if there*

exists a higher dimensional subspace cluster $(\mathbf{C}', \mathbf{S}')$ *with:*

$$(\mathbf{S} \subset \mathbf{S}') \ \wedge \ \mathbf{C}' \subseteq \mathbf{C} \ \wedge \ \left(\frac{|\mathbf{C}'|}{|\mathbf{C}|} \geq \delta \right)$$

Following Definition 2.4 we define a cluster as redundant if (most of) the objects contained in the cluster are also contained in a higher dimensional cluster. We use a parameter δ to specify the degree of redundancy acceptable to the user. A redundancy parameter of $\delta = 0$ specifies a clusters as redundant if at least one object contained in the cluster is part of a higher dimensional cluster (please note, that we will demand clusters to contain a minimum number of objects and hence at least this number of objects has to be contained in a higher dimensional cluster). Using a value of $\delta = 1$ only defines a cluster as redundant if the objects are completely contained in a higher dimensional subspace cluster.

How redundancy should be defined often depends on the application and the kind of information which is relevant for the user. The redundancy parameter δ allows controlling which clusters are considered as redundant. Using different parameter settings results in different cluster concepts mined by our DUSC subspace clustering model. For example it may be interesting

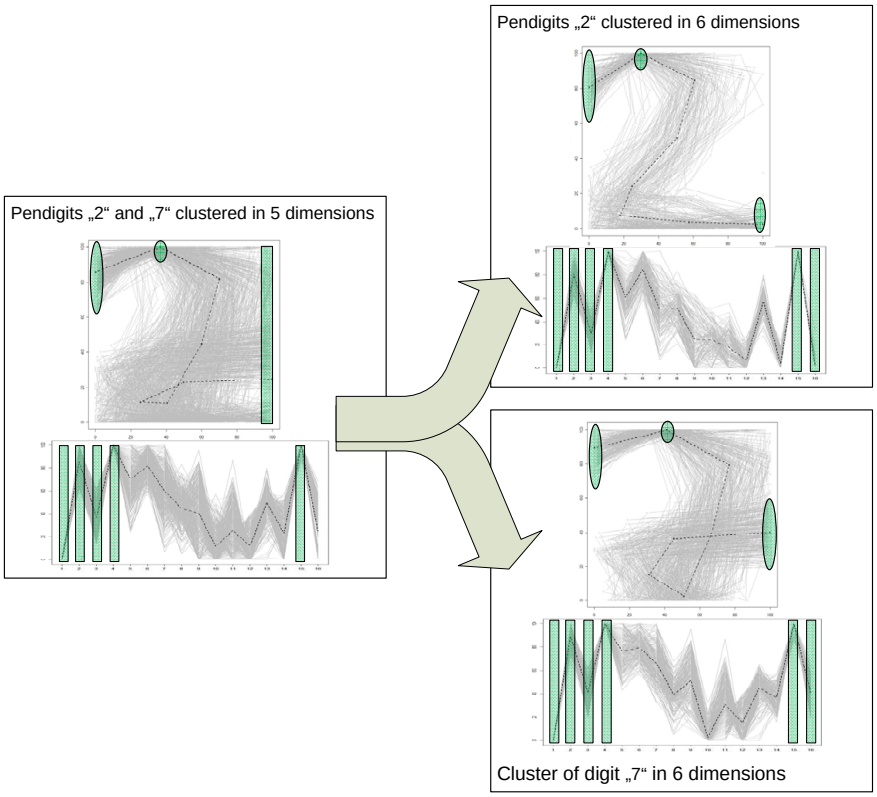

Figure 2.5: Five-dimensional cluster splitting into two six-dimensional clusters

for a user if a low dimensional cluster splits into two or more higher dimensional clusters. In this case the lower dimensional cluster reflects a concept which is contained in two or more other groups.

Figure 2.5 presents a pendigit cluster (left part) which splits up into two different clusters (right part) if an additional dimension is considered. The left part illustrates a five-dimensional cluster containing the digits two and seven. As illustrated by the green marked areas the digit one and two have a common concept. Both digits start in a similar way (the first two positions of many handwritten twos and sevens start similar) and end with a similar x-value. If the y-value of the last position is also considered the digits two and seven are assigned to two different clusters (right part of Figure 2.5). Depending on the application a user might only be interested in the two right clusters. Using a redundancy parameter of $\delta < 0.5$ defines the left cluster as redundant and hence the cluster would be removed from the result

set.

If a user is also interested in a concept which describes the similarity between the digit two and seven the left cluster is not redundant. In this case a redundancy of $\delta > 0.5$ also identifies the left cluster and hence all three clusters are mined.

DUSC subspace clustering model

So far, we have studied the density of individual objects and the redundancy of clusters. Additionally to the extended density constraints we only define object sets as a cluster if they contain a significant part of the data. This constraint is implemented by a *minSize* parameter. The resulting subspace cluster model taking these conclusions into account is formalized in the following:

Definition 2.5 ***DUSC Subspace Cluster***
A set of objects $\mathbf{C} \subseteq \mathbf{DB}$ *in subspace* $\mathbf{S} \subseteq \mathbf{D}$ *is a subspace cluster if:*

- $\mathbf{C}$ *is* $\mathbf{S}$-***connected*** *and* ***maximal*** *(Definition 1.6)*
- $\forall o \in \mathbf{C}$: $\varphi_{\varepsilon}^{\mathbf{S}}(o) \geq max\{\mathbf{F} \cdot \alpha(\mathbf{S}, \varepsilon), \eta \cdot \omega(\mathbf{S})\}$
  (more dense than expected and not pseudo-dense)
- $|\mathbf{C}| \geq minSize \cdot |\mathbf{DB}|$ ***(minimum cluster size)***
- $\neg\exists(\mathbf{C}', \mathbf{S}')$ *a subspace cluster with:* $(\mathbf{S} \subset \mathbf{S}') \ \wedge \ \mathbf{C}' \subseteq \mathbf{C} \ \wedge \ \left(\frac{|\mathbf{C}'|}{|\mathbf{C}|} \geq \delta\right)$
  (not redundant)

Following density-based clustering paradigm Definition 2.5 defines subspace clusters as maximal connected sets of dense objects. The DUSC subspace clustering model extends these notions with statistically sound density computation via normalized Epanechnikov kernel and expected density. To make sure that clusters reflect the inherent structure of the data, they should contain a certain minimum number of objects (typically a certain percentage of the data). Consequently Definition 2.5 only defines larger density-connected sets which reflect a general concept as a cluster.

To sum up, Definition 2.5 ensures that clusters contain a significant part of the data, are not redundant and their elements are more dense than expected. The parameters $minSize, \delta$ and $\mathbf{F}$ introduced by the DUSC subspace

clustering approach model these properties. We demonstrate the robustness of these parameters in Section 2.7.

2.6 Apriori subspace clustering

As subspace clustering mines clusters in multidimensional and high dimensional data spaces, evaluating the cluster model in a naive way is infeasible as the number of possible subspaces (and subspace clusters) is exponential with the dimensionality. One method used by many subspace clustering algorithms for improving the runtime is to apply an apriori style algorithm. Apriori algorithms evaluate the search space in a breadth-first approach (level by level). After subspace clusters of a specific dimensionality are mined a candidate set for the next higher dimensionality is generated. If one projection of a subspace cluster candidate does not contain any cluster the candidate and all higher dimensional projections of the candidate are pruned from search space.

Pruning requires a monotonicity on some property of subspace clusters. If a region which does not form a subspace cluster in some dimensionality implies that this region cannot be a subspace cluster in any higher dimensional subspace, we may safely prune this region from further consideration.

2.6.1 Monotonicity

As the density definition given in Section 1.4 depends on the dimensionality of the analyzed subspace, it is not monotonous in the above sense and thus cannot directly be used for pruning. The higher dimensional a subspace, the lower its expected density. Thus, a region which is not dense according to a low dimensional subspace's density threshold, may be dense with respect to a higher dimensional subspace's threshold. To overcome this problem, we introduce the new concept of weakest density threshold.

Definition 2.6 ***Weak density.***
An object o in a subspace $\mathbf{S}$ is defined as weakDense:

$$weakDense^{\mathbf{S}}(o) \text{ if: } \varphi_{\varepsilon}^{\mathbf{S}}(o) \geq max\{\mathbf{F} \cdot \alpha(\mathbf{D}, \varepsilon), \eta \cdot \omega(\mathbf{D})\}$$

The weak density definition uses the highest, $|\mathbf{D}|$-dimensional threshold $max\{\mathbf{F} \cdot \alpha(\mathbf{D}, \varepsilon), \eta \cdot \omega(\mathbf{D})\}$ for the density of an object o in subspace $\mathbf{S}$. Thus

if an object o is not weakDense, o cannot be dense in any super subspace of $\mathbf{S}$.

Theorem 2.2 *For all subspaces* $\mathbf{T}$ *with* $\mathbf{S} \subseteq \mathbf{T} \subseteq \mathbf{D}$*:*

$$\neg weakDense^{\mathbf{S}}(o) \Rightarrow \neg(\varphi_{\varepsilon}^{\mathbf{T}}(o) \geq max\{\mathbf{F} \cdot \alpha(\mathbf{T}, \varepsilon), \eta \cdot \omega(\mathbf{T})\})$$

Proof: In inequality $\varphi_{\varepsilon}^{\mathbf{S}}(o) \geq max\{\mathbf{F} \cdot \alpha(\mathbf{D}, \varepsilon), \eta \cdot \omega(\mathbf{D})\}$ of the weak density definition, all variables depending on highest dimensionality $|\mathbf{D}|$ are on the right hand side, whereas those depending on $|\mathbf{S}|$ are on the left. We derive two conclusions.

(1): the left hand side decreases with growing dimensionality, as distances between objects may only grow as more dimensions are considered. Density decreases for increasing distances. Consequently, the density on the left decreases when subspace $\mathbf{T}$, is used instead of subspace $\mathbf{S}$:

$$(1)\ \varphi_{\varepsilon}^{\mathbf{T}}(o) \leq \varphi_{\varepsilon}^{\mathbf{S}}(o)$$

(2): the right hand side increases with decreasing dimensionality, as both $\alpha(\mathbf{D}, \varepsilon)$ and $\omega(\mathbf{D})$ grow. Note that the ratio of a $|\mathbf{T}|$-dimensional ε-sphere to the $|\mathbf{T}|$-dimensional data space decreases with growing dimensionality. Thus,

$$(2)\ max\{\mathbf{F} \cdot \alpha(\mathbf{D}, \varepsilon), \eta \cdot \omega(\mathbf{D})\} \leq max\{\mathbf{F} \cdot \alpha(\mathbf{T}, \varepsilon), \eta \cdot \omega(\mathbf{T})\}$$

We conclude:

$$\begin{aligned}
&\neg weakDense^{\mathbf{S}}(o) \\
&\quad \Leftrightarrow \varphi_{\varepsilon}^{\mathbf{S}}(o) < max\{\mathbf{F} \cdot \alpha(\mathbf{D}, \varepsilon), \eta \cdot \omega(\mathbf{D})\} \\
&\quad \overset{(1)}{\Rightarrow} \varphi_{\varepsilon}^{\mathbf{T}}(o) < max\{\mathbf{F} \cdot \alpha(\mathbf{D}, \varepsilon), \eta \cdot \omega(\mathbf{D})\} \\
&\quad \overset{(2)}{\Rightarrow} \varphi_{\varepsilon}^{\mathbf{T}}(o) < max\{\mathbf{F} \cdot \alpha(\mathbf{T}, \varepsilon), \eta \cdot \omega(\mathbf{T})\}
\end{aligned}$$

$\diamond$

Following Theorem 2.2, an object o can be pruned if it violates the weak density condition, as o cannot be dense in any higher dimensional subspace. Secondly, density-connected sets can be pruned if they contain less than *minSize* objects. Pruning based on these two properties, called *weak monotonicity*, is valid in the sense that no cluster is wrongfully dropped from consideration.

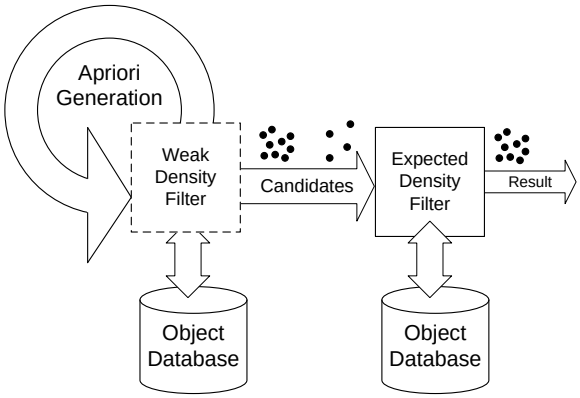

Figure 2.6: Concept of apriori algorithm for DUSC subspace cluster model

2.6.2 An apriori approach for DUSC

In this section we propose an apriori style algorithm for our DUSC subspace clustering model. Apriori based algorithm have been often used for subspace clustering as in [KKK04, AGGR98]. In contrast to the clustering model proposed in [KKK04] the DUSC clustering model additionally considers redundancy, expected density and a minimum required size for clusters. Objects can only be pruned from search space using the weak density definition as described in the last section. Hence, our proposed algorithm uses a filter-and-refinement architecture to prune the search space (see also Figure 2.6). Redundant clusters are finally removed from the result set.

Since the filter steps and the redundancy removal require additional computational effort the apriori based DUSC approach needs even more time to cluster a data set than SUBCLU [KKK04]. Please notice that the authors of SUBLCU already reported runtimes from over one week. We introduce an efficient algorithm for the DUSC subspace clustering model (eDUSC) in Chapter 3. For completeness we include the apriori based DUSC algorithm in this thesis. The apriori method for our DUSC subspace clustering method consists of four parts: the main method, two methods which implement the apriori properties and the density-based clustering method. The method for finding dense clusters is although used in the following chapters.

Before we explain the algorithmic parts in detail we give a short overview:

- Algorithm 1 - AprioriDUSC: main method which iteratively generates new subspace clusters level by level (breadth-first). Finally redundant

clusters are removed.

- Algorithm 2 - AprioriGenerate: generates higher dimensional candidate subspace clusters by joining existing subspace clusters using the apriori principle.
- Algorithm 3 - CheckProjections: utilizes the monotonicity property of weak dense subspace clusters to prune candidates.
- Algorithm 4 - DenseCluster: clustering method which identifies density-based clusters according to a given density threshold.

Algorithm 1: *AprioriDUSC* is the main method of the apriori based DUSC subspace clustering approach. Starting with one dimensional clusters candidates (Line 4) weak dense clusters are identified in Line 8 with respect to the weak density threshold (see Line 2). According to the apriori concept two weak dense subspace clusters are joined to higher dimensional cluster candidates in Line 9. Before higher dimensional weak dense clusters are mined the unbiased clustering result is identified by the refinement step in Line 12 using the expected density threshold (see Line 11). Candidate generation, weak dense clustering and refinement are continued until no more cluster candidates for the considered dimensionality k exist (Line 5). Finally redundant clusters are removed from the result set in Line 17.

Algorithm 2: *AprioriGenerate* illustrates the apriori step of our subspace clustering algorithm. The apriori method joins two subspace clusters of dimensionality k to a subspace cluster candidate of dimensionality $k+1$. A join is only performed if the two subspace clusters overlap in the first $k-1$ dimensions (see Line 6). This concept is described in [AS94, AGGR98, CWZZ99, KKK04].

In [KKK04] the monotonicity of objects has been extended to clusters. Hence, a projection of a density-connected cluster to a lower dimensional subspace is again part of exactly one density-connected cluster in this subspace. Consequently objects belonging to a higher dimensional weak dense cluster must be part of one lower dimensional weak dense cluster. Thus, a candidate set for a higher dimensional weak dense cluster can be obtained

Algorithm 1: AprioriDUSC(**DB**, **D**)

```
   // apriori generation of cluster candidates
1  k = 1
2  τ_weak = max{F · α(D, ε), η · ω(D)}
3  result = ∅
4  aprioriSC[1] = ⋃_{i∈D} {(DB, {i})}
5  while AprioriSC[k] ≠ ∅ do
       // find weak dense clusters for each apriori candidate
6      weakSC[k] = ∅
7      foreach (C, S) ∈ aprioriSC[k] do
8          weakSC[k] = weakSC[k] ∪ DenseCluster(C, S, τ_weak)
9          aprioriSC[k + 1] = AprioriGenerate(weakDensSC[k])
       // identify true unbiased subspace clusters
10     foreach (C, S) ∈ weakSC[k] do
11         τ_exp = max{F · α(S, ε), η · ω(S)}
12         result = result ∪ DenseCluster(C, S, τ_exp)
13     k = k + 1
   // check redundancy of the result set
14 foreach (C, S) ∈ result do
15     foreach (C', S') ∈ result do
16         if (C' ⊆ C) ∧ (S ⊆ S') ∧ (|C'| ≥ r · |C|) then
17             result = result \ (C, S)
18 return result
```

by intersecting the objects belonging to the corresponding lower dimensional clusters (see Line 7). If the candidate set is valid it is added to the result set.

Algorithm 3: As described above the method *CheckProjections* (Line 7) verifies if each k-dimensional projection of a $k+1$-dimensional cluster candidate exists. And if the cluster exists the object set is restricted to the intersection of the two sets.

Algorithm 2: AprioriGenerate($clusterSet$)

```
   // generate candidate clusters of dimensionality k+1
   // for a given clusterSet of dimensionality k
1  result = ∅ // clusters contained in C
2  foreach (C, S) ∈ clusterSet do
       // consider a subspaces with ordered dimensions
3      Let S be s_1, ..., s_k with s_i < s_j for i < j
4      foreach (C', S') ∈ clusterSet do
           // join subspace S' with dimensions in S
5          Let S' be s'_1, ..., s'_k with s'_i < s'_j for i < j
6          if (k = 1 ∨ s_1 = s'_1 ∧ ... ∧ s_{k-1} = s'_{k-1}) ∧ s_k < s'_k then
7              C_cand =CheckProjections(C ∩ C', S ∪ S') if C_cand ≠ ∅ then
8                  result = result ∪ {(C_cand, S ∪ S')}
9  return result
```

To check if a weak dense cluster of dimensionality k corresponds to a lower dimensional projection of the $k+1$-dimensional cluster candidate it is sufficient to check if the cluster shares some objects with the candidate (Line 2 - part 1). As mentioned above, the monotonicity for weak dense objects also holds for weak density-connected objects. Hence, a lower dimensional cluster $(\mathbf{C}', \mathbf{S}')$ which shares objects with a higher dimensional cluster $(\mathbf{C}, \mathbf{S})$ corresponds to the projection of $(\mathbf{C}, \mathbf{S})$ to $\mathbf{S}'$. Thus, we can restrict the candidate set to the intersection $\mathbf{C} \cap \mathbf{C}'$ (see Line 2).

Finally we check if each lower dimensional projection of a cluster candidate has been found (Line 7) and the cluster has the minimal required size (Line 5). Otherwise the candidate $(\mathbf{C}, \mathbf{S})$ is removed from further consideration (Line 10 and 6).

Algorithm 4: For each cluster candidate *DensCluster* identifies density-based clusters according to a given density threshold. The filter step investigates each cluster candidate for weak density-connected clusters using the threshold τ_{weak}. Unbiased subspace clusters are then mined from weak dense clusters using the threshold τ_{exp}.

As a cluster candidate may contain multiple density-connected subspace

Algorithm 3: CheckProjections($\mathbf{C}, \mathbf{S}, clusterSet$)

```
   // check all projections for a given cluster candidate
 1 numProj = 0 foreach (C', S') ∈ clusterSet do
       // check if C', S' is projection of C, S
 2     if S ∩ S' = S' ∧ C ∩ C' ≠ ∅ then
 3         C = C ∪ C'
 4         numProj = numProj + 1
 5         if |C| < minSize · |DB| then
               // cluster candidate is too small
 6             return ∅
   // if all lower dimensional projections exist
 7 if numProj = |S| then
       // return constrained data set
 8     return C
 9 else
       // cluster is no candidate
10     return ∅
11 return result
```

clusters the algorithm returns a set of subspace clusters (Line 1). A density-connected cluster is mined by calculating the transitive closure for each object p in $\mathbf{C}$ (Line 2): Line 11 computes the density value *density* and the area of influence $\mathcal{A}$ for each object. If the object is dense the object belongs to a cluster (Line 15). Further on, all objects within the area of influence may also belong to the cluster. To evaluate the density of these objects the area of influence is added to the set *set* (Line 16). Next, this *set* is extended until all density-connected connected objects have been picked up. If the identified cluster is large enough (Line 18) the subspace cluster is added to the result set (Line 19). As the objects belonging to one cluster cannot belong to another cluster the objects are removed from the remaining object set $\mathbf{C}$.

Algorithm 4: DenseCluster($\mathbf{C}$, $\mathbf{S}$, τ)

```
1  result = ∅ // clusters contained in C
2  while C ≠ ∅ do
3      Let p be an object in C
4      set = {p} // set of possible cluster objects
5      cluster = ∅ // objects belonging to the cluster
       // calculate cluster for object p
6      while set ≠ ∅ do
7          Let o be an object in set
8          A = ∅ // set of possible cluster objects
9          density = 0 // and density for object o
10         foreach q ∈ DB do
11             if ‖o − q‖^S ≤ ε then
12                 A = A ∪ {q}
13                 density = W(|o − q‖^S)
14         if density ≥ τ then
15             cluster = cluster ∪ {o}
16             set = set ∪ A
17         set = set \ {o}
18     if |cluster| ≥ minSize · |DB| then
19         result = result ∪ {(cluster, S)}
20     C = C \ cluster
21 return result
```

2.7 Experiments

We ran extensive experiments on real world data to demonstrate the accuracy of the DUSC subspace clustering model. We do not evaluate the runtime behavior in this section as the apriori based DUSC approach is similar to SUBCLU since both algorithm use the same algorithmic concept. In the next chapter we present an efficient approach for mining subspace clusters. We therefore evaluate the runtime behavior of the different subspace clustering methods in the following chapters and concentrate on the accuracy evaluation in this section.

Measurements

Efficiency is simply measured in terms of runtime, accuracy is determined as quality and coverage. Corresponding roughly to the measures of precision and recall, quality accounts for purity of the clustering, while coverage measures the size of the clustering. We use two quality measures in this thesis. One quality measure uses the entropy. I.e. for k class labels $Class_i$ ($i = 1..k$) and a cluster $\mathbf{C}$ the entropy is calculated by

$$H(\mathbf{C}) = -\sum_{i=1}^{k} p(Class_i|\mathbf{C}) \cdot \log(p(Class_i|\mathbf{C}))$$

with $p(Class_i|\mathbf{C})$ determines the relative frequency of the objects contained in $\mathbf{C}$ belonging to $Class_i$. Entropy is an information theoretic indicator for the homogeneity of the data, which is used for evaluating the quality of subspace clusterings [SZ04]. For readability, we take the inverse entropy and normalize it to a range of 0% to 100% by dividing by the maximum entropy $log(k)$. More precisely, the quality of a set of clusters $\mathcal{C} = \{\mathbf{C}_1, \ldots, \mathbf{C}_n\}$ is given by:

$$Quality(\mathcal{C}) = 1 - \frac{\sum_{j=1}^{n} |\mathbf{C}_i| \cdot H(\mathbf{C}_j)}{\log(k) \sum_{j=1}^{n} |\mathbf{C}_j|}$$

Coverage is the percentage of objects of the complete data set which is contained in any subspace cluster. It indicates the ratio of clustered objects to noise. The amount of noise in a data set is typically not known apriori, but noise is present in most real world data sets. As sparsely populated regions

often exhibit a weak correlation to the class label, quality can be improved if less objects belong to a cluster (coverage drops). Thus we always evaluate quality and coverage in combination.

Another measure to determine the accuracy is the F1-value that is commonly used in evaluation of classifiers and recently also for subspace or projected clustering as well [WF05, MSE06]. It is computed as the harmonic mean of recall ("are all classes detected?") and precision ("are the classes accurately detected?") values, respectively. The F1-value of the whole clustering is simply the average of all F1-values. The class label assigned to any detected subspace cluster is its most frequent class label. The F1-value for a cluster set containing n clusters with k classes is calculated by:

$$\begin{aligned}
\mathcal{I}sClass(\mathbf{C}) &:= Class_i : \text{if } \mathbf{C} \text{ has maximal support for } Class_i \\
\mathcal{C}last(Class_i) &:= \bigcup_{j=1}^{n} \{\mathbf{C} \ : \ \mathcal{I}sClass(\mathbf{C}) = Class_i\} \\
\mathcal{P} &:= \sum_{i=1}^{k} \frac{\left|\{o \in \mathcal{C}last(Class_i) \ : \ o \text{ has label } Class_i\}\right|}{\left|\bigcup_{j=1}^{k} \mathcal{C}last(Class_j)\right|} \\
\mathcal{R} &:= \sum_{i=1}^{k} \frac{\left|\{o \in \mathcal{C}last(Class_i) \ : \ o \text{ has label } Class_i\}\right|}{|DB|} \\
F1 &:= \frac{2}{\frac{1}{\mathcal{P}} + \frac{1}{\mathcal{R}}}
\end{aligned}$$

We use both quality measures the entropy and the F1 measure in our experiments.

Parameters

Our algorithm has few and intuitive parameters. They can be easily understood by users and are very robust on different data sets. Recall that $minSize$ is the minimum number of objects required per subspace cluster. Values around 1% of the data lead to manageable result sizes. Lower values produce more subspace clusters than users might want to study, whereas higher values diminishes the output size. η is fixed to two to eliminate the

empty-space problem. Obviously, at least two objects should be contained even in very high dimensional subspace clusters.

The density-thresholds α and ω are directly computed from the expected density. Users only provide a factor **F** which describes by how much the expected density should be exceeded. This **F** is independent of the database size and the dimensionality. Finally, the bandwidth ε regulates the density measure. This parameter can be estimated using standard methods from statistics [Sil86].

We demonstrate robustness of our DUSC algorithm on two real world data sets with very different data distributions (Pendigits, Glass - see Table 2.1). The first experiment studies the effect of density threshold **F** on quality and coverage. The results shown in Figure 2.7 confirm that **F** is independent of the dimensionality and data size that DUSC is remarkably robust with respect to **F**.

Similar effects can be observed for the redundancy parameter r (see Figure 2.8). The heuristic $r = 5\%$ works very well for both data sets. Varying r has nearly no effect on the quality and coverage of the Glass data set. The coverage of Pendigit data increases from 74% to 92% with increasing redundancy while the quality only goes down by 5 percentage points. This effect is due to the fact that lower dimensional clusters are more likely to be removed if the tolerated redundancy is set to a very low value. Thus allowing a certain amount of redundancy has a beneficial effect on coverage while quality remains stable.

Accuracy

We evaluated the accuracy [quality (Q) and the coverage (C)] of DUSC against SUBCLU and SCHISM using the data sets illustrated in Table 2.1. For DUSC we used the default values for **F** according to the heuristic in

	Objects	Dimensions	Classes	Source
Pendigits	7494	16	10	[AN07]
Glass	214	9	6	[AN07]
Vowel	990	10	11	[AN07]
Shapes	160	17	9	[KWX+06]

Table 2.1: Real world data sets

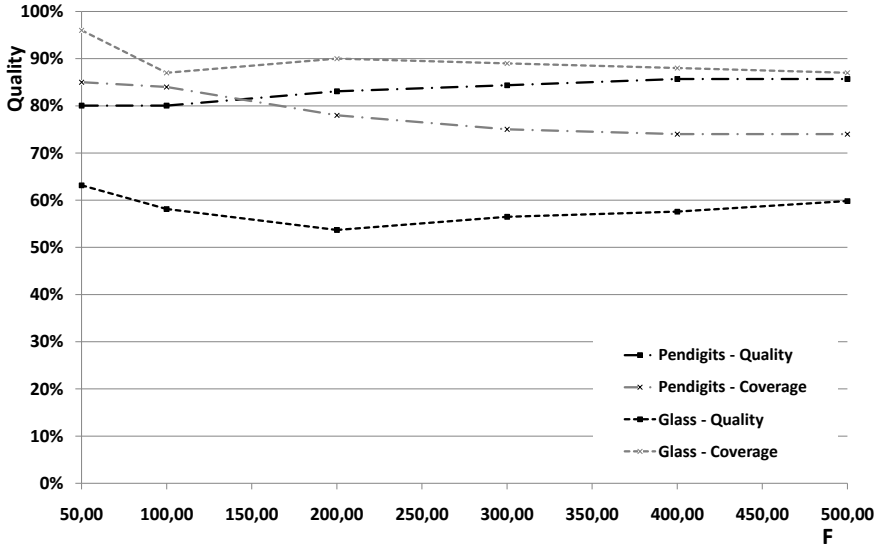

Figure 2.7: Quality vs. density threshold **F**

section 2.5. As SUBCLU and DUSC are both density-based clustering algorithms we used the heuristic presented in [KKKW03] to determine ε. SCHISM is a grid based approach. Its parameters ξ, τ are also determined using the original heuristic in [SZ04]. For their third parameter u which cuts off low dimensional clusters, we noticed that the heuristic does not yield good results in terms of accuracy. To obtain better quality results for SCHISM that explain more about the true differences between different density models, we used an even lower value for u than suggested by the authors. However as SCHISM is a grid based approach it still does not reach the quality of DUSC. The first column in Figure 2.9 shows the quality results of DUSC with redundancy set to zero which are the best measured qualities for all data sets. Allowing more redundancy, coverage increases significantly and quality goes only slightly down. However, even for $r = 10\%$ DUSC shows better quality than the competing algorithms. The fact that coverage is not 100% indicates that DUSC can distinguish between noise and clusters in subspaces of varying dimensionalities. The pendigits data set, for example, contains handwritten numbers, some of which are clearly different from the rest of the data set. Biased algorithms like SCHISM and SUBCLU do not detect noise, but assign all objects to clusters.

The last data set SHAPE contains rotated versions of 9 different shapes, but only 3 of the shapes clearly form clusters. Thus most of the objects

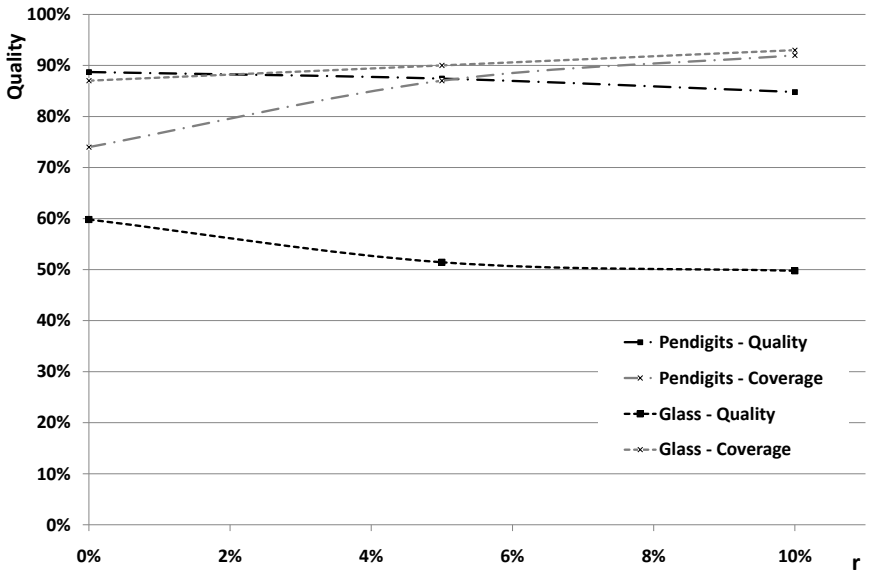

Figure 2.8: Quality vs. redundancy parameter r

	DUSC 0%		DUSC 5%		DUSC 10%		SUBCLU		SCHISM	
	Q	C	Q	C	Q	C	Q	C	Q	C
Pendigits	86	74	83	87	81	92	58	100	77	100
Glass	60	87	51	90	50	93	44	100	44	99
Vowel	82	70	79	100	74	100	10	100	42	100
Shape	100	31	100	31	100	31	98	82	100	1

Figure 2.9: Comparing accuracy of DUSC with other algorithms

have to be considered noise. DUSC detects the given clusters correctly while SCHISM detects only a small part of the clusters and SUBCLU mixes up clusters with noise (less than 100% quality).

In our last experiment we evaluate the quality of our Epanechnikov and Rectangular density measure using the F1-value. For both density measures we removed the dimensionality measure as discussed in Chapter 2.4. We give details about the unbiased Rectangular density measure in the next chapter. The F1-value directly combines accuracy and coverage measured by precision and recall to one value. From the results presented in Figure 2.10 we can see that our DUSC subspace clustering model achieves a much better accuracy for all three data set for both density measures. Further on, the Epanechnikov density measure always performs better than the Rectangular

density measure.

Please recall that SUBCLU is also based on a Rectangular density measure. However, for the pendigits data set DUSC with Rectangular density measure nearly achieves a quality which is twice as good as SCHISM or SUBCLU (for the Epanechnikov measure DUSC beats both algorithms by more than a factor of two). Since the normalized rectangular density measure also performs very well this result indicates that removing dimensionality bias is important to obtain a good accuracy for density-based clustering in subspaces.

	DUSC 0% Epanechnikov F1-value	DUSC 0% Rectangular F1-value	SCHISM F1-value	SUBCLU F1-value
Pendigits	55%	48%	25%	24%
Glass	60%	57%	47%	52%
Vowel	38%	35%	21%	17%
Shape	52%	51%	2%	40%

Figure 2.10: F1-value on real world data using different density measures

Summing up, DUSC shows both very high quality results. Figure 2.9 and 2.10 shows that the quality of DUSC is always superior to SCHISM and SUBCLU in all setups. Thus, our statistically sound density computation indeed leads to a higher accuracy in real world applications. Approximating by grids or pruning via fixed thresholds, as in the competing approaches, leads to subspace cluster loss. DUSC clearly shows best accuracy.

2.8 Conclusion

We introduced DUSC, an efficient density-based subspace clustering algorithm. Using both statistical density estimation and expected density in varying subspaces, we are capable of accurately grasping the inherent data structure without dimensionality bias. We proposed an algorithm which relies on three pruning properties to reduce the search space. Our experiments on large high dimensional synthetic and real world data sets show that DUSC outperforms other subspace clustering algorithms in terms of accuracy and runtimes.

Chapter 3

Efficient unbiased subspace clustering

Subspace clustering algorithms search for clusters in all projections of the data. An exhaustive search of all subspace clusters, however, entails runtimes which are infeasible for large high dimensional data sets, since the number of possible subspace projections is exponential in the dimensionality of the data. Consequently, existing subspace clustering algorithms trade off efficiency for accuracy. Heuristics or grid-based discretization provide efficiency benefits, yet clusters cut apart by the grid or wrongfully pruned are lost in the process.

Moreover, the resulting subspace clusters are often highly redundant, i.e. many clusters are detected multiply in several projections. Containing essentially the same information, redundant subspace clusters have to be removed to allow users to review the entire output. In addition, removal of redundancy actually improves quality.

In this chapter, we propose lossless efficient detection of subspace clusters. We define a new density-conserving grid for substantial efficiency gains without losing any clusters. Based on this grid an efficient density unbiased subspace clustering algorithm (**eDUSC**) is developed which uses a filter-and-refinement architecture to determine dense regions. Additionally we introduce the SC-tree used for efficient subspace indexing. eDUSC and the SC-tree work in a depth-first fashion and hence make in-process-removal of redundant clusters possible and allow merging of subspace cells.

In thorough experiments on synthetic and real world data sets, we demonstrate that eDUSC supported by the SC-tree is faster than existing subspace

clustering algorithms by orders of magnitude. Additionally we investigate the effects of removing redundancy on the quality of the resulting subspace clustering. Our results indicate that removing even improves the accuracy of a subspace clustering.

3.1 Introduction

As discussed in the last chapters for high dimensional data clusters are typically hidden in noisy and irrelevant attributes. To detect clusters in any possible subspace projection, subspace clustering seeks relevant attributes for each individual subspace cluster. As the number of possible subspace projections is exponential in the dimensionality of the data space, efficiency is a crucial issue in subspace clustering [AGGR98, KKK04].

Naive re-running of full-space clustering algorithms for the exponentially number of subspace projections is clearly infeasible. Hence, grid-based discretization of the space or other lossy approximations have been proposed [AGGR98, NGC99, SZ04]. Whereas these algorithms show far better runtimes, they lose clusters which are cut apart by the grid or missed by the heuristic strategy. Many subspace clustering algorithms as proposed in the last chapter are based on the apriori principle: assuming monotonicity of clusters with respect to the dimensionality of the subspace, higher dimensional projections of sparse regions are discarded as sparse as well [AS94, KKK04].

The number of resulting subspace clusters is usually exponential as well. Subspace clusters typically show in different projections of the space, generating redundant subspace clusters which contain essentially the same information. To allow users to analyze reasonable result sizes, redundant subspace clusters have to be removed. Pruning of redundant subspace clusters significantly reduces the number of subspace regions that have to be mined. This pruning is lossless, i.e. not based on approximations or heuristics.

As subspace clustering is extremely complex, many existing subspace clustering algorithms focus on efficiency of mining. Apriori-based algorithms work in a bottom-up fashion on the dimensionality of the subspaces. Starting with one-dimensional subspace clusters, candidate subspace clusters are generated from the detected subspace clusters in smaller dimensionalities. These bottom-up approaches suffer from two drawbacks in terms of efficiency: First, they have to generate the entire set of one- and two-dimensional subspace

clusters before any pruning can be performed. Second, the output is highly redundant and can only be cleared out once all redundant subspace clusters have been generated in a costly process. Immediate pruning of redundant subspace clusters is thus infeasible. Consequently, runtimes of several weeks for density-based subspace clustering are reported in the literature [KKK04].

Further on, indexing is a challenge. As subspace clusters are mined in different dimensionalities, indexing other than by inverted lists on the individual dimensions is inapplicable as the number of subspaces is exponential in the number of dimensions. Consequently many subspace clustering algorithms do not scale well in terms of dimensionality. Faced with the huge number of subspace projections existing subspace clustering algorithms resort to approximations and are thus lossy. Non-approximative approaches do not scale to these high dimensions and are thus intractable for real world applications.

In this chapter, we propose a new concept for overcoming the existing trade-off between accuracy and efficiency. We present a novel efficient subspace clustering algorithm for our dimensionality unbiased subspace clustering model (**eDUSC**). eDUSC is based on a multistep filter-and-refinement architecture and hence is capable of mining results of high quality at far better runtimes. We achieve substantial efficiency gains by a novel density-conserving grid with new density conserving borders that guarantee completeness. Our lossless subspace clustering algorithm mines subspace clusters of arbitrary dimensionality.

The high efficiency of eDUSC is additionally based on a depth-first concept. Existing approaches as CLIQUE or SUBCLU [AGGR98, KKK04] are similar to the apriori algorithm [AS94] originally introduced in frequent itemset mining in that they work their way bottom up and breadth first on the subspace lattice. Starting from one dimensional results, all subspace clusters of dimensionality k are mined, before candidates of dimensionality $k + 1$ are generated, and so on. For example, in Figure 3.1 (left part), all the subspaces clusters in the first $(\{1\}, \{2\}, \{3\} \dots)$ and second $(\{1, 2\}, \{1, 3\}, \dots)$ level are mined before reaching the three-dimensional subspaces.

We identify three problems of breadth-first mining: first, large sets of candidates in low-dimensional projections, second, no in-process-removal of redundant subspace clusters, and third, repeated density computations due to lack of index support. All of these problems contribute to high runtimes of breadth-first algorithms.

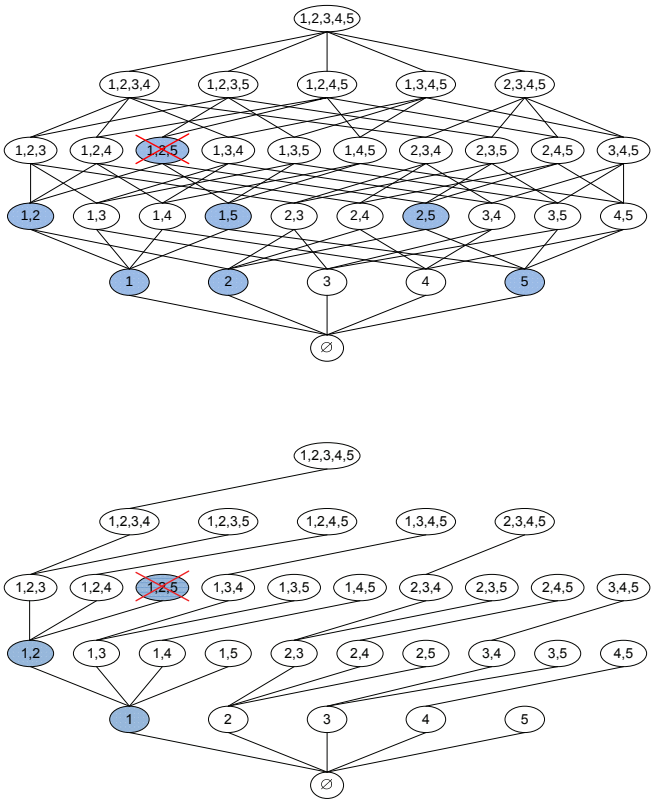

Figure 3.1: Example for a breadth first (top) and depth first (bottom) pruning of subspace clusters in a five dimensional lattice

Our depth first approach overcomes these drawbacks and reduce runtimes substantially by in-process-removal of redundant subspace clusters. In Figure 3.1 (right part) we assume a cluster found in subspace $\{1, 2, 5\}$. During the depth-first processing the shaded subspaces are recursively restricted to $\{1, 2, 5\}$. In contrast to breadth-first approaches $\{1\}$, $\{2\}$, $\{5\}$, $\{1, 2\}$, $\{1, 5\}$ and $\{2, 5\}$ do not have to be clustered to get to this subspace, if a cluster found in $\{1, 2, 5\}$ is the only non-redundant result. In this case we avoid the costly density computations by our filter-and-refinement architecture for all of the lower dimensional projections of the subspace cluster $\{1, 2, 5\}$. By our novel in-process removal we focus on new subspace clusters not yet found in higher dimensional projections.

Moreover the density conserving grid of eDUSC makes a meaningful in-

dex support for potential subspace cluster regions possible. We introduce the **SC-tree** for indexing of grid cells which supports the depth-first approach of eDUSC. To the best of our knowledge, this is the first index for subspace clusters in the literature. The SC-tree is a compact representation of potential subspace cluster regions which greatly reduces database scans for subspace clustering. In-process redundancy pruning as well as pruning based on the density of regions is integrated.

The contributions of this chapter yield substantial efficiency improvements for the unbiased density-based subspace clustering model described in the last chapter by including:

- **Accuracy**: unbiased lossless non-redundant density-based subspace clustering
- **Efficiency**: an efficient filter-and-refinement subspace clustering algorithm
- **Indexing**: a novel powerful index for density-connected subspace clusters
- **Pruning**: pruning the search space and automatic in-process-removal of redundancy

Accuracy is achieved through a novel lossless density-conserving grid. Unlike traditional grid based discretization, our approach guarantees detection of all subspace clusters. Efficiency is ensured by our novel filter-and-refinement architecture which mines density-connected regions without scanning the complete database. The compact density-conserving grid structure makes indexing possible and since eDUSC works depth-first powerful subspace cluster pruning is possible without apriori-based candidate generation.

This chapter is structured as follows: we discuss related work in the next section. Section 3.3 reviews the density-based subspace clustering model before we present our novel density-conserving grid in Section 3.4.1. Section 3.4.3 gives our new multistep algorithm, which quickly generates potential subspace clusters without false dismissals, i.e. completeness is guaranteed. We then discuss how to efficiently index potentially dense regions and how to remove redundancy in process. We analyze our algorithm on both synthetic and real world data sets in the experiments section 3.6. We demonstrate far better runtimes than existing subspace clustering approaches.

3.2 Related work

In this section we shortly review different approaches for efficiently finding subspace clusters. Subspace clustering mines clusters in arbitrary, possibly overlapping, subspaces. As the number of subspaces is exponential in the number of dimensions, existing algorithms trade-off efficiency for accuracy:

CLIQUE (Clustering In QUEst) uses a grid to discretize the search space and a bottom-up apriori-based algorithm [AGGR98]. Grids greatly reduce the computational complexity, yet clusters which spread across cells are missed and results are sensitive to the position of the grid.

SCHISM (Support and Chernoff-Hoeffding bound-based Interesting Subspace Miner) extends CLIQUE using a variable threshold adapted to the dimensionality of subspaces [SZ04]. As the apriori property does not hold for variable thresholds, heuristics and a grid-based discretization are used for pruning. Consequently, clusters are lost as well.

ENCLUS (Entropy-Based Subspace Clustering) and RIS (Ranking Interesting Subspaces) search for subspaces which might potentially contain clusters [CWZZ99, KKKW03]. Clusters are mined in a second step using any traditional full space clustering algorithm. As there is no connection between these two steps, they suffer from repeated cluster property computations in all subspaces which result in intractable runtimes and loss of subspace clusters in other projections. Redundant projections of clusters are not identified by both algorithms.

SUBCLU (density-connected SUBspace CLUstering) extends DBSCAN to subspace clustering via an apriori-based algorithm [KKK04]. Runtimes are better than for naive re-runs of DBSCAN, yet still not feasible for practical applications.

Recently some approaches have been developed to identify redundant subspace clusters. FIRES uses one-dimensional clusters to anticipate a set of potential high dimensional subspace cluster candidates [KKRW05]. This step tries to avoid redundancy as intermediate subspaces are not investigated. One problem of FIRES is that high dimensional clusters often do not induce one-dimensional clusters. Hence, high dimensional clusters may be missed which makes FIRES an approximative clustering algorithm.

Jinze et al. propose an ontology-driven subspace clusters method [LWY04]. Using the information from the ontology redundant clusters can be identified and removed from the result set. Since the redundancy definition is based on

an ontology this method is not applicable to settings without ontologies. In [AHLP06] Agarwal et al. developed a recommender system for research papers based on a subspace clustering method. Redundant clusters are removed from the result set in a post processing step.

In this work, we show how incorporating redundancy removal into the mining process allows for more efficient and accurate subspace clustering. Immediate removal of repetitive patterns is exploited for our novel SC-tree index. Existing approaches cannot benefit from indexing of high dimensional data for subspace clustering as apriori based algorithms are used, as noted also in [KKK04]. In apriori based bottom-up mining, redundant clusters can only be detected once the entire low dimensional projection has been processed. In contrast, our depth-first eDUSC technique is capable of immediate removal of these redundant clusters, allowing for both better pruning and the first indexing subspace clustering method to our knowledge.

3.3 Density-based subspace clusters

In the last chapter we proposed the DUSC subspace clustering model which mines dimensionality unbiased subspace clusters. The basic idea is to normalize the density in any subspace by the expected density of its dimensionality. In the DUSC subspace clustering model, an Epanechnikov kernel is used (depicted on the right in Figure 3.2), whereas the SUBCLU approach corresponds to the Rectangle kernel (depicted on the left in Figure 3.2). The algorithm proposed in this chapter works for any of these weighting functions. For simplicity of the presentation we use the rectangular weighting function in this section. We shortly present how to remove dimensionality bias for the rectangular density measure before we describe our algorithmic concept in the next chapter.

Definition 3.1 ***Rectangular density measure***

The rectangular density measure assigns each object within the respective neighborhood the same weight ($\mathcal{W}(t) = 1$)*. Based on this weighting function we define the Rectangular density measure for an area of influence given by* ε *as:*

$$\varphi\text{-}Rect^{\mathbf{S}}_{\varepsilon}(o) = \left|\mathcal{A}^{\mathbf{S}}_{\varepsilon}(o)\right|$$

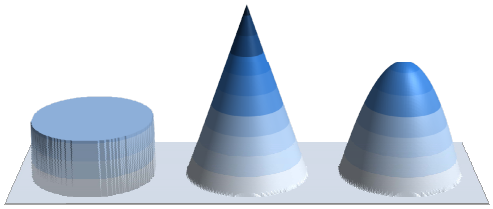

Figure 3.2: Rectangle, Triangular, and Epanechnikov kernels

The rectangular density measure specializes the generalized density measures as given in Definition 1.4. Since all objects within the area of influence are assigned the same weight the weighted sum over all objects simply corresponds to the number of objects contained in the area of influence.

To remove dimensionality bias the expected density of the density measure has to be taken into account. For the rectangular density measure the expected density is simply the average number of objects in the ε-neighborhood w.r.t. the dimensionality. This can be computed as the ratio of the volume of the ε-sphere to the volume of the subspace. Density in subspaces is thus normalized with respect to the dimensionality simply by adapting the threshold to the expected density.

Definition 3.2 ***Unbiased rectangular density measure.***
An object o is ***dense*** *in subspace $\mathbf{S}$ of dimensionality $|\mathbf{S}|$ if its ε neighborhood in $\mathbf{S}$, $\mathcal{A}_\varepsilon^{\mathbf{S}}(o) = \{p \in \mathbf{DB} \mid \|o-p\|^{\mathbf{S}} \leq \varepsilon\}$, contains more than $minPoints$ objects and exceeds the expected density in this subspace by a factor F:*

$$o \text{ dense in } \mathbf{S} :\Leftrightarrow \quad |\mathcal{A}_\varepsilon^{\mathbf{S}}(o)| \geq max\{minPoints, F \cdot expDen(\mathbf{S})\}$$

with expected density in the $\mathbf{S}$-dimensional subspace:

$$expDen(\mathbf{S}) = |\mathbf{DB}| \cdot \frac{c_{\mathbf{S}} \cdot \varepsilon^{|\mathbf{S}|}}{\mathbf{v}^{|\mathbf{S}|}}$$

$$\text{and } c_{\mathbf{S}} = \frac{\pi^{\frac{|\mathbf{S}|}{2}}}{\Gamma\left(\frac{|\mathbf{S}|}{2}+1\right)} \quad \text{volume of the unit sphere with:}$$

$\Gamma(n+1)=n\cdot\Gamma(n)$, $\Gamma(1)=1$, $\Gamma\left(\frac{1}{2}\right)=\sqrt{\pi}$ *the gamma function,*

and $\mathbf{v}^{|\mathbf{S}|}$ the volume of $|\mathbf{S}|$-dimensional subspace .

Please recall that the DUSC subspace clustering model defines subspace clusters as sets of density-connected objects. These sets are maximal, i.e. all density-connected objects are grouped together. Dimensionality bias is removed from density measures by taken the dimensionality of the subspace into account. Additionally, we do not consider one-dimensional clusters in this chapter as they are typically uninteresting for the user. These clusters reflect merely the distribution of individual attributes which is usually known or analyzed using standard analytic methods. As finding clusters in one dimension is not the goal of subspace clustering one dimensional clusters and clusters containing less than $minSize$ objects are excluded from the result set.

Another contribution of the DUSC subspace clustering model is to remove redundant clusters. Lower dimensional projections of subspace clusters typically do not differ much from their higher dimensional counterparts. Additional correlations are typically considered interesting by the user and hence lower dimensional projections which essentially cluster the same objects are irrelevant. As the number of projections is exponential, the number of redundant subspace clusters is overwhelming and hinders analysis of the mining result. Consequently, the output has to be filtered for user benefit.

We shortly recall the DUSC subspace clustering model (see also Definition 2.5) using the rectangular density measure:

Definition 3.3 ***DUSC subspace clusters using the rectangular density measure***

A set of objects $\mathbf{C} \subseteq \mathbf{DB}$ *in subspace* $\mathbf{S} \subseteq \mathbf{D}$ *is a subspace cluster if:*

- $\mathbf{C}$ *is* $\mathbf{S}$-***connected*** *and* ***maximal*** *(Definition 1.6)*
- $|\mathbf{C}|$ *is* ***dense****:* $\forall o \in \mathbf{C} : |\mathcal{A}_\varepsilon^{\mathbf{S}}(o)| \geq max\{minPoints, F \cdot expDen(\mathbf{S})\}$
- $|\mathbf{C}|$ *has a* ***minimum support****:* $|\mathbf{C}| \geq minSize \cdot |\mathbf{DB}|$
- $\neg\exists(\mathbf{C}', \mathbf{S}')$ *a subspace cluster with:* $(\mathbf{S} \subset \mathbf{S}') \ \wedge \ \mathbf{C} \subseteq \mathbf{C}' \ \wedge \ \left(\frac{|\mathbf{C}|}{|\mathbf{C}'|} \geq \delta\right)$ *(****not redundant****)*

In the next section, we discuss our depth-first algorithm for efficient dimensionality unbiased subspace clustering (**eDUSC**). Existing subspace clustering methods which follow a breadth-first apriori like methods suffer

from two major drawbacks: repeated density computations on potential subspace clustering regions and low pruning power as redundancy can only be removed after all lower dimensional projections have been processed.

Thus, in-process-removal of redundancy in depth-first algorithms is a key to efficient subspace clustering. Proceeding depth-first allows pruning of redundant subspace cluster if there is a higher dimensional subspace cluster that contains objects of the lower dimensional projection. Efficiency is not only hindered by lack of redundancy pruning. Furthermore, repeated density computations on potential subspace clustering regions are computational expensive. Using a filter-and-refinement architecture which prunes regions without computing the actual density of individual objects greatly improves efficiency. We additionally propose an index for subspace clusters which supports the depth-first filter architecture or our eDUSC approach.

3.4 Depth-first multistep subspace clustering

Detecting all subspace clusters in any subspace is a highly complex task. The number of possible subspaces is exponential in the number of attributes. Naively searching all of these subspaces is hence not feasible in most applications and pruning the search space is crucial.

Existing subspace clustering algorithms use an apriori-based approach to prune possible subspace clusters. These algorithms in general use a breadth-first search to identify sparse regions in low dimensional spaces. Those regions are then used to exclude higher dimensional projections of these regions from search. As discussed in the last chapter, to identify higher dimensional clusters the density threshold has to be decreased with respect to the expected density in higher dimensions. Thus in practical settings nearly no low dimensional subspace projection can be pruned as this threshold is almost always exceeded. Consequently, in breadth-first traversal, nearly all low dimensional subspaces are generated as candidate sets. This effect causes an extreme degeneration of the runtime behavior as the complete candidate sets of lower dimensions have to be held available in order to apply the apriori property (see also Section 2.6.2). Further on, redundancy pruning is not possible as information about higher dimensional clusters is only available after all low dimensional projections have been processed.

In Chapter 2.6.2 we introduced an apriori based algorithm for the DUSC

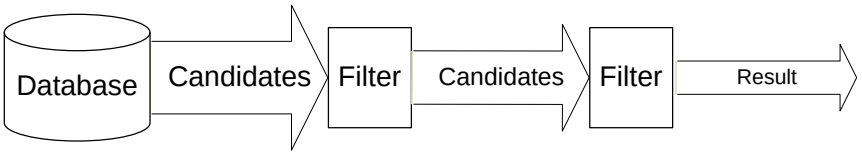

Figure 3.3: Multistep architecture

subspace clustering model. To identify clusters according to the unbiased DUSC density measure a filter technique based on weak dense subspace clusters is applied. Since the apriori method is applied on weak dense clusters the apriori based DUSC subspace clustering approach suffers from the same runtime problems as all apriori based algorithms. We tackle the runtime problem of subspace clustering algorithms by developing a new subspace clustering strategy.

To the best of our knowledge, we propose the first algorithm for subspace clustering that proceeds in depth-first order. A cluster mined in a low dimensional space is extended to the highest possible projection before the next low dimensional subspace cluster is analyzed. As not all clusters in all projections of the same dimensionality have to be determined simultaneously our algorithm overcomes the huge memory need of existing subspace clustering algorithms. Moreover, the depth-first approach makes redundancy pruning possible since information about high dimensional clusters is available before the next low dimensional subspace cluster is investigated. In Section 3.5 we will discuss an index structure which allows for efficient in-process-removal of redundant clusters.

A time consuming task in density-based subspace clustering is to identify density-connected regions. A neighborhood query is required to determine if an object is dense. Consequently, the complexity for clustering a subspace region in one subspace is quadratic in the number of data objects contained in that region [EKSX96]. The number of subspaces which has to be investigated is typically very high and, consequently, the number of regions which have to be investigated is huge. Thus the computational cost of density-based clustering is problematic for large high dimensional data sets.

We reduce the computational cost by a multistep algorithm for subspace clustering with a filter step which efficiently determines regions potentially containing density-connected clusters. Multistep query processing is an established efficiency enhancement method in database retrieval. Approxima-

tion filters are used to efficiently determine candidate sets. By ensuring that each approximative step is optimistic, no true result is dropped. This guarantees completeness of the result. In the last step, candidate sets are refined to obtain the final result. Thus, the accuracy is preserved [FRM94, SK98]. This architecture of multistep algorithms is illustrated in Figure 3.3.

We propose a filter step for subspace clustering based on a novel density conserving grid. Traditional grid based clustering methods lose accuracy as the result is influenced by the position and resolution of the grid [Sil86]. Moreover, dense regions might be cut apart. The density conserving grid does not lose any density-connected cluster but reduces the number of necessary scans for density-connected regions. Hence, the good runtime performance of grid based algorithms is preserved.

In our multistep depth-first search we combine the following three methods to an efficient Dimensionality Unbiased Subspace Clustering (**eDUSC**) algorithm:

(1) a density conserving grid which ensures completeness

(2) two effective filters for pruning the search space

(3) an in-process pruning of redundant subspace clusters

Next, we give the definition of a density conserving grid before we prove monotonicity properties for pruning the search space. We then discuss the eDSUC algorithm which efficiently mines density-connected subspace clusters based on the density conserving grid.

3.4.1 Density conserving grid

Existing grid-based subspace clustering methods are sensitive to position and resolution of the grid [AGGR98, NGC99, SZ04]. Subspace clusters might be cut apart, and thus not be detected. This is illustrated in Figure 3.4 (left): the highlighted cluster of eight objects is cut into two cells $\mathbf{C_{1,2}}$ and $\mathbf{C_{1,3}}$. By counting the number of objects per cell and comparing it against a minimum threshold of e.g. 5, neither cell would qualify. Consequently, the cluster is missed. Cell $\mathbf{C_{2,2}}$, on the other hand, contains no actual cluster, even though it contains more than 5 objects.

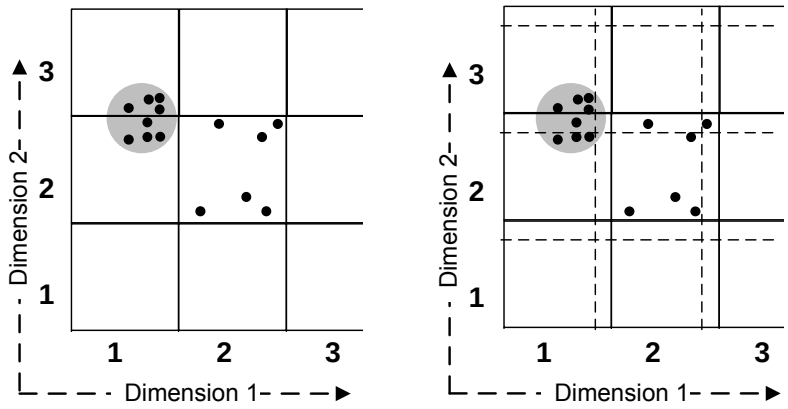

Figure 3.4: Traditional grid discretization (left), with connectivity borders (right)

Density-connected clusters might spread across multiple grid cells. To detect these clusters, we introduce a novel density conserving grid by devising connectivity borders. Subspace clusters which are density-connected across cells can be detected along the connectivity borders between cells. As the border between cells $\mathbf{C_{1,2}}$ and $\mathbf{C_{1,3}}$ in Figure 3.4 (right) contains objects, they should not be discarded to ensure that no potential cluster is missed. In the algorithm, these cells are merged until completely enclosing hypercubes for each subspace cluster are found.

Definition 3.4 ***Density-conserving grid.***
A ***density conserving grid*** *is a regular grid with connectivity borders:*

- *A* ***regular grid*** *is a partition of the attribute range* $\mathbf{v}$ *into* $g = \lceil \frac{\mathbf{v}}{w} \rceil$ *intervals* p_i *of equal width* w *and* $p_i = [w \cdot (i-1), w \cdot i),\ i = 1 \ldots g$.
- ***Connectivity-borders*** *are intervals of* ε*-width at the upper border of each cell in each dimension* $b_i = [w \cdot i - \varepsilon, w \cdot i),\ i = 1, \ldots, g$
- *A* s*-dimensional* ***subspace cell*** $\mathbf{C}_{\alpha_\mathbf{1},\ldots,\alpha_\mathbf{d}}$ *is given by an index vector* $\alpha_1, \ldots, \alpha_d$ *of the corresponding intervals* p_{α_j}, $\alpha_j \in \{1, \ldots, g, *\}$, *where* $(d-s)$ *stars denotes the unconstrained dimensions of the cell.*
- *A border* $\mathbf{B}_{\alpha_\mathbf{1},\ldots,\overline{\alpha_\mathbf{k}},\ldots,\alpha_\mathbf{d}}$ *in dimension* k *is given by the index vector* $\alpha_1, \ldots, \overline{\alpha_k}, \ldots, \alpha_d$ *of its cell* $\mathbf{C}_{\alpha_\mathbf{1},\ldots,\alpha_\mathbf{d}}$, *where the dash denotes the border dimension* k.

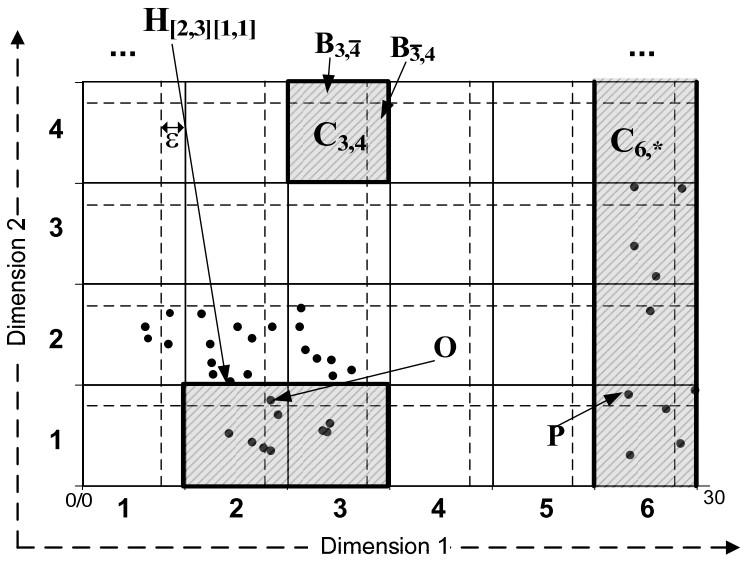

Figure 3.5: Cells, connectivity borders and hypercubes projected to dimension 1 and 2

In Figure 3.5 we illustrate our approach by an example of a two-dimensional space where each attribute range **v** is from 0 to 30. A **regular grid** partitions the data space into equiwidth intervals $p_i = [5\,(i-1)\,, 5i),\ i = 1 \ldots 6$. **Connectivity-borders** are areas of ε-width at the upper border of each cell in each dimension. The regular grid cells and the connectivity borders together form the **density conserving grid**. A **subspace cell** is given by a d-dimensional vector of the d interval numbers. A star $(*)$ denotes that the cell is not constrained in this dimension. We mark dimensions with "-" to denote connectivity borders for this dimension.

For example, cell $\mathbf{C_{3,4}}$ contains two connectivity borders $\mathbf{B_{\bar{3},4}}$ and $\mathbf{B_{3,\bar{4}}}$. In the example, $\mathbf{C_{3,4}}$ is a 2-dimensional cell which is restricted in interval 3 for dimension 1 and in interval 4 for dimension 2. $\mathbf{C_{6,*}}$ is a 1-dimensional cell restricted in interval 6 for dimension 1 and not constrained in dimension 2. Hypercubes consist of merged cells are given by interval ranges per dimension. $\mathbf{H_{[2,3][1,1]}}$ is a 2-dimensional example hypercube.

$|\mathbf{H}_{[a_1,b_1]\ldots[a_d,b_d]}|$ denotes the number of objects $o \in \mathbf{DB}$ contained in a hypercube. A full space hypercube $\mathbf{H}$ is projected to a subspace $\mathbf{H^S}$ with $\mathbf{S} = \{k_1, \ldots, k_s\}$ by removing the restriction in all other dimensions (i.e. setting the indices of $i \notin \mathbf{S}$ of $\mathbf{H}$ to "$*$"). For example $|\mathbf{H}_{[2,3][1,1]}|$ contains

9 objects, while its less constrained projection ($\mathbf{S} = \{1\}$) to dimension one $\mathbf{H}_{[\mathbf{2},\mathbf{3}][*]}$ contains 35 objects (all objects contained in the 2nd and 3rd interval).

3.4.2 Monotonicity properties

For good runtime performance it is important to prune regions from search space. Safely pruning without jeopardizing completeness requires a monotonicity on cluster properties (see also Section 2.6). Monotonicity ensures that if a region which is not a subspace cluster in a subspace $\mathbf{S}$ implies that this region cannot be a subspace cluster in any higher dimensional subspace $\mathbf{T}$ ($\mathbf{S} \subseteq \mathbf{T} \subseteq \mathbf{D}$), we may safely prune this region from further consideration.

We exploit two monotonicity properties in our eDUSC algorithm. The first monotonicity property states that a higher dimensional projection of any hypercube contains at most the same number of objects (e.g. $minSize$ objects).

$$(A)\ |\mathbf{H}^{\mathbf{S}}_{[a_1,b_1]\ldots[a_d,b_d]}| \leq n \ \Rightarrow\ |\mathbf{H}^{\mathbf{T}}_{[a_1,b_1]\ldots[a_d,b_d]}| \leq n$$

The monotonicity for hypercubes is used by grid-based subspace clustering algorithms to prune the search space. The proof for monotonicity (A) is given in [AGGR98]. Our eDUSC algorithm uses this monotonicity property in conjunction with the novel density conserving grid to prune spare regions.

The second monotonicity property is used by traditional density-based subspace clustering algorithms to remove objects which violate the density constraint. It states that the number of objects contained in a neighborhood is monotonously decreasing with increasing dimensionality (see proof in [KKK04]).

$$|N^{\mathbf{S}}_{\varepsilon}(o)| \leq minPoints \ \Rightarrow\ |N^{\mathbf{T}}_{\varepsilon}(o)| \leq minPoints$$

Monotonicity does not hold for $|N^{\mathbf{S}}_{\varepsilon}(o)|$ w.r.t. normalized density as the expected density $expDen$ decreases with growing dimensionality (see Definition 3.2). In Section 2.6 we already proofed a weaker monotonicity criterion for the Epanechnikov density measure (**weak density**). The same reasoning can be applied on the rectangular kernel as used in this chapter. Consequently we achieve the following monotonicity: if an object is not **weak dense** in $\mathbf{S}$ it is not **dense** in $\mathbf{T}$.

$$(B)\ \ |N_\varepsilon^{\mathbf{S}}(o)| \leq max\{minPoints, F \cdot expDen(\mathbf{D})\} \ \Rightarrow$$
$$|N_\varepsilon^{\mathbf{T}}(o)| \leq max\{minPoints, F \cdot expDen(\mathbf{T})\}$$

Based on monotonicity property (A) and (B) we derive the following two pruning criteria: (1) a hypercube $\mathbf{H}^{\mathbf{S}}_{[a_1,b_1]\ldots[a_d,b_d]}$ which completely encloses a connected region can be pruned if the region contains less than $minSize$ objects and (2) an object o can be pruned if it violates the weak density condition, as o is not dense in any higher dimensional subspace.

3.4.3 The eDUSC algorithm

In this chapter we propose our multistep eDSUC algorithm which uses two filters based on the pruning criteria described in the last chapter (see Figure 3.6). The first filter step (*hypercube approximation filter*) approximates each density-connected subspace cluster by a hypercube in the density conserving grid. Using the monotonicity criterion for hypercubes (A), a hypercube enclosing a density-connected set can be pruned if it contains less than $minSize$ objects. Due to the novel density conserving grid this pruning is performed without losing any cluster. Consequently a hypercube can be pruned without computing the density-connected sets contained in that region.

The second filter step (*weak density filter*) evaluates the hypercube according to weak density. Following Monotonicity criteria (B), a region can be pruned if the objects contained in that region violate the weak density condition. Thus the multistep approach of the eDUSC algorithm extends the pruning criterion of grid-based algorithms to density-based subspace clustering (see Figure 3.6). Pruning based on these two filter steps, is lossless in the sense that no cluster is wrongfully dropped from consideration.

As mentioned above, density-connected clusters might extend into several grid cells. The hypercube approximation filter thus merges adjacent cells until an enclosing hypercube for each subspace cluster is found. For efficiency reasons we process cells in lexicographical order to ensure that each cell has to be processed only once. We mark future merges w.r.t lexicographic order as *induced merges*; when these marked cells are actually processed, *performed merges* combine the cells. Lexicographically greater cells are processed in the future, hence we denote them as *future cells*, and lexicographically smaller cells as *past cells*. An enclosing hypercube is found if no more *induced merges* and *performed merges* are necessary from the current cell.

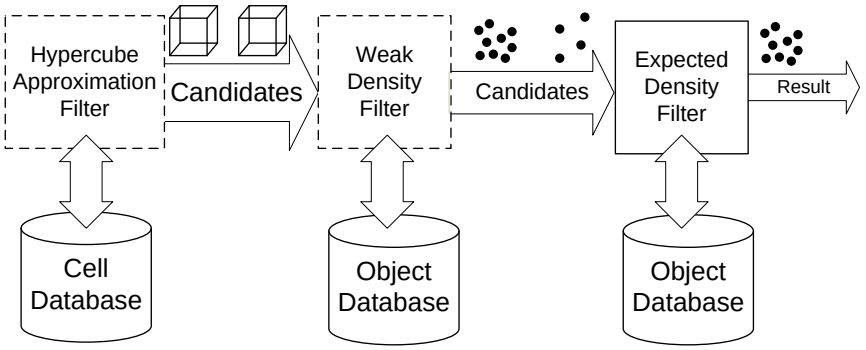

Figure 3.6: Multistep eDSUC algorithm

If a density-connected subspace cluster stretches from one cell into another, the connectivity border contains at least one object. Otherwise, as we set the border width exactly to the neighborhood range ε, objects in one cell cannot be in the ε-neighborhood of the other (see Definition 1.5). When a cell is processed, the connectivity borders of the cell are checked. For each border which contains an object a merge is *induced* into the corresponding adjacent cell. If both connectivity borders contain an object a cluster might also extend into the diagonal cell, i.e. adjacent to the intersection of both borders, ensuring an induced merge to this diagonal cell.

When processing a cell, first merges into future cells are *induced* and then merges with past cells are *performed*. Hence, a merge is always performed with a *past cell* which induced a merge into the cell. During a merge process the number of objects (the $objectCounter$) and the number of merges induced into future cells (the $induceCounter$) are aggregated. We use the concept of *induced* merges to indicate whether an enclosing hypercube for a density-connected region is found: if all induced merges of a hypercube are performed ($induceCounter = 0$) a MICH (*maximal induced cluster hypercube*) is found.

Using the method of induced and performed merges guarantees that a MICH does not cut a cluster into two parts and that each subspace cluster induces exactly one MICH (see Theorem 3.1). The hypercube approximation filter is an optimistic filter and hence a MICH is a conservative approximation which does not always contain a cluster.

Figure 3.7 illustrates the merge steps performed to identify a MICH enclosing the example cluster. We start by processing each cell of dimensionality two, beginning with cell $\mathbf{C_{1,1}}$. The number of objects is determined,

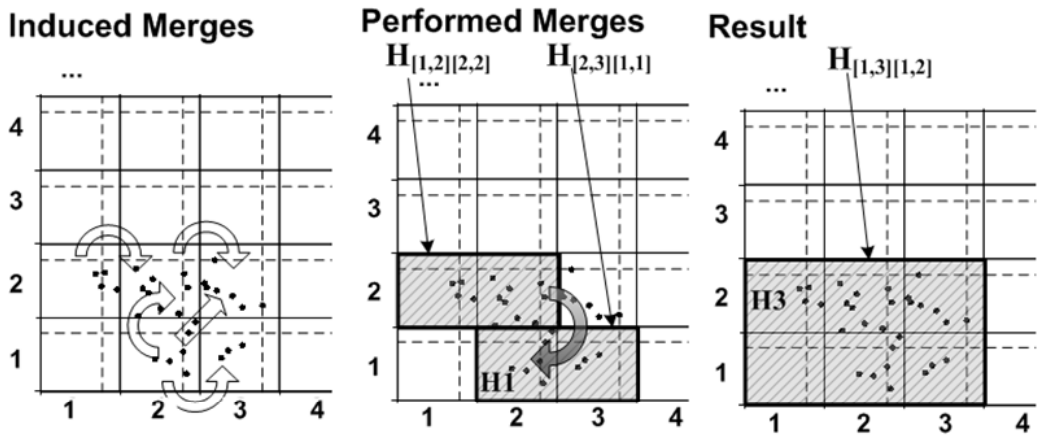

Figure 3.7: Induced and performed merges

and if the cell is not empty, its connectivity borders are tested. Cell $\mathbf{C_{1,1}}$ is empty and hence no merge is induced. Next, cell $\mathbf{C_{2,1}}$ is processed. As both connectivity borders contain an object, cell $\mathbf{C_{2,1}}$ induces a merge into $\mathbf{C_{3,1}}$, $\mathbf{C_{2,2}}$ and into $\mathbf{C_{3,2}}$. Next, cell $\mathbf{C_{3,1}}$ performs the merge with $\mathbf{C_{2,1}}$, creating $\mathbf{H_{[2,3][1,1]}}$. When $\mathbf{C_{2,2}}$ is processed, it is merged with $\mathbf{C_{1,2}}$ to $\mathbf{H_{[1,2][2,2]}}$. After this, the next merge induced into $\mathbf{C_{2,2}}$ is performed (second part of Figure 3.7). This merge step creates the hypercube $\mathbf{H_{[1,3][1,2]}}$. Finally, cell $\mathbf{C_{3,2}}$ is processed. As $\mathbf{C_{3,2}}$ was already merged with both cells, only the *induceCounter* is decremented. After this step, no more merges are necessary and $\mathbf{H_{[1,3][1,2]}}$ corresponds to a MICH.

After a MICH is found the eDUSC algorithm checks if the hypercube contains more than a number of *minSize* objects. If the region does not contain enough objects, the hypercube and all higher dimensional projections of that hypercube are pruned (monotonicity property (A)). Thus the regions identified by the density conserving grid are used to reduce the number of time consuming scans for density-connected clusters and to prune the search spaces.

Since hypercube approximation filter is a conservative approximation the next filter steps remove all "false alarms", i.e., all MICHs which do not contain a cluster. First a MICH is scanned for weak density-connected regions (using Algorithm 4). Based on monotonicity property (B) a MICH can be pruned from search space if the MICH does not contain a weak density-connected subspace cluster. To obtain the final result each weak dense cluster

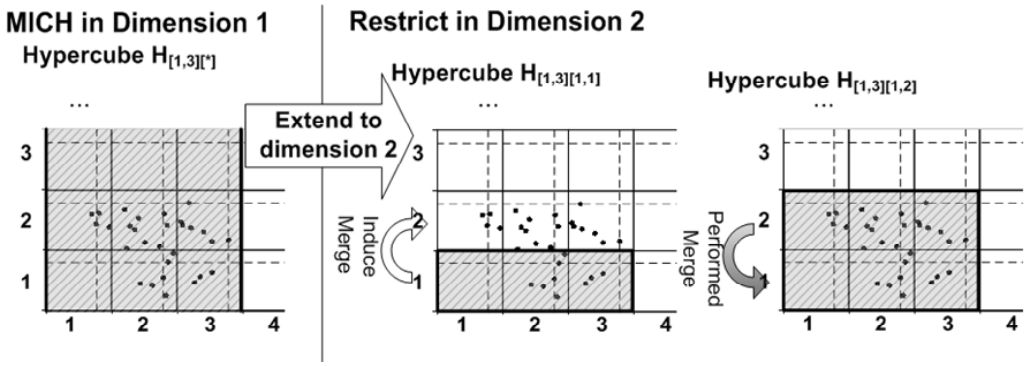

Figure 3.8: Extending a MICH to next dimension

is than rescanned w.r.t. the expected density of the subspace.

Working depth-first, our eDUSC analyzes only one MICH at a time by successively extending the MICH in all dimensions (see Section 3.4.1). Hence the eDUSC algorithm avoids generating more than one MICH at the same time. When a MICH is extended to a new dimension it is iteratively restricted to each grid cell. For example a two dimensional hypercube embedded in a four dimensional space $\mathbf{H}_{[\mathbf{1},\mathbf{3}][\mathbf{1},\mathbf{2}][*][*]}$ can be extended into dimensions three and four. We first extended it in dimension three and restrict it to cell one ($\mathbf{H}_{[\mathbf{1},\mathbf{3}][\mathbf{1},\mathbf{2}][\mathbf{1},\mathbf{1}][*]}$). Afterwards the merge procedure is applied recursively. Assume the merge step mines the MICH $\mathbf{H}_{[\mathbf{1},\mathbf{3}][\mathbf{1},\mathbf{2}][\mathbf{1},\mathbf{2}][*]}$. This MICH is next restricted in dimension four: $\mathbf{H}_{[\mathbf{1},\mathbf{3}][\mathbf{1},\mathbf{2}][\mathbf{1},\mathbf{2}][\mathbf{1},\mathbf{1}]}$. After this step the next hypercube ($\mathbf{H}_{[\mathbf{1},\mathbf{3}][\mathbf{1},\mathbf{2}][*][\mathbf{1},\mathbf{1}]}$) is analyzed and processed accordingly.

For simplicity, we demonstrate the extension step using a one-dimensional MICH $\mathbf{H}_{[\mathbf{1},\mathbf{3}][*]}$ (see Figure 3.8). After extending the hypercube $\mathbf{H}_{[\mathbf{1},\mathbf{3}][*]}$ to dimensions two and restricting it to the first grid cell $\mathbf{H}_{[\mathbf{1},\mathbf{3}][\mathbf{1},\mathbf{1}]}$ the merge procedure is applied again. As the corresponding connectivity border is not empty a merge is induced and performed directly afterwards, resulting in hypercube $\mathbf{H}_{[\mathbf{1},\mathbf{3}][\mathbf{1},\mathbf{2}]}$. $\mathbf{H}_{[\mathbf{1},\mathbf{3}][\mathbf{1},\mathbf{2}]}$ again corresponds to a MICH in subspace $\mathbf{S} = \{1, 2\}$ as its connectivity border is empty. Next the extension step is applied again, restricting the hypercube in the next dimension if available.

We now proof completeness by showing that each cluster induces exactly one MICH.

Theorem 3.1 ***eDUSC is complete:***
For each subspace cluster the enclosing MICH is mined.

We sketch the proof below:

Proof: Each cluster induces exactly one MICH:

- Start with the lexicographically first cell $\mathbf{C_1}$ of a cluster. As $\mathbf{C_1}$ is the first cell no *past cell* contains objects belonging to the cluster and hence no merge with a *past cell* is necessary.

- If the cluster is density-connected to a *future cell* $\mathbf{C_f}$, the corresponding connectivity border contains at least one object. Thus the cell induces a merge into $\mathbf{C_f}$.

- As $\mathbf{C_f}$ is processed in the future, the induced merge is performed. After the merge step, the hypercube encloses cell $\mathbf{C_f}$. Thus after processing a cell $\mathbf{C_f}$ no merge with a *past cell* is necessary.

- When the *induceCounter* is zero, the connectivity borders are all empty. Consequently, no more *future* merges are necessary. Thus, the entire MICH for the cluster is found. ◇

Figure 5 outlines the eDSUC algorithm. The first lines (Line 1-14) illustrate the **hypercube approximation filter**. This first filter step recursively calls the other filter steps of our multistep eDUSC algorithm. Recall that each MICH is a conservative approximation of a cluster (a cluster candidate). Using monotonicity property (A) a MICH and all higher projections can be pruned if the MICH does not contain at least *minSize* objects (see Line 16).

If a MICH is a cluster candidate, it is subsequently analyzed by the second filter step (**weak density filter**) of the eDUSC algorithm (see Line 17). The second filter step uses the second monotonicity property (B) to determine if a MICH contains a weak density-connected region analogously to [EKSX96]. If the hypercube does not contain any weak density-connected cluster the corresponding hypercube and all its higher dimensional hypercube projections can be pruned. This is the second step of our multistep algorithm. Following a depth-first approach redundant clusters can be pruned (see Line 19).

Algorithm 5: eDUSC($\mathbf{H}, \mathbf{S}, d_{first}$)

```
1  minPoints_weak = max{minPoints, F · expDen(d)}
2  S = ∅
3  for k = d_first ... d do                          // for each dimension and
4      S = S ∪ {k}  for i = 1 ... g do                                  // each cell
5          H = restrictHypercube(H, [k, i]);                        // restrict H
6          if |S| < 2 then // no testing for one-dimensional cells
7              eDUSC(H, S, k + 1) ;        // continue eDUSC recursion
8          else
9              if |S| == 2 then
10                 I = TestBorders(H) ;                  // test all borders
11             else if |S| > 2 then
12                 I = TestNewBorder(H, [k, i]);         // test new border
13             InduceMerge(H, I) ;          // for future neighbors of H
14             H = PerformMerges(H) ;  // for past neighbors of H
15             if induceCounter(H)==0 then                // MICH complete
16                 if objectCounter(H) ≥ minSize and
17                 isNotRedundant(H) then                    // 1st filter
18                     C = GetObjects(H)
19                     if DenseCluster(C, S, minPoints_weak) then
                       // 2nd filter
20                         eDUSC(H, S, k + 1); // further restriction
21                         if isNonRedundant(C, S) then
22                             minPoints_exp =
                               max{minPoints, F · expDen(|S|)}
                                                          // 3rd filter
23                             DenseCluster(C, S, minPoints_exp);
24                         else
25                             PruneCluster(C, S) ;    // C is redundant
26                     else
27                         Prune(C, S) ;  // C and higher dim.  proj.
28                 else
29                     Prune(C, S) ;      // C and higher dim.  proj.
```

The third step of the multistep algorithm can not be used for pruning but is necessary to remove false alarms (see Line 20). If a region contains a weak density-connected cluster it is still necessary to check if the region contains a cluster w.r.t. the dimensionality of the subspace (see Definition 3.2). Thus the method *ExpDensityScan* scans a region w.r.t. the expected density of a subspace. If the region finally contains a cluster the result is stored.

The hypercube approximation filter also uses an additional heuristic to avoid merging too many cells. Before large hypercubes (containing three or more cells) are merged we check if at least one object in the respective connectivity border is dense according to the weak density criterion. If not, the merge can be safely omitted. Further on, we use inverted lists to efficiently calculate the density of an object or to cluster a region (as in [KKK04]). To extract the actual subspace cluster from dense hypercubes, the original objects of the hypercube (plus an ε range) have to be investigated. Inverted lists can be used to quickly retrieve all of these objects.

Using the filter-and-refinement step of the eDUSC approach allows for effective filtering of too small and redundant regions without losing any density-connected subspace cluster. The second and third filter steps are using a traditional database scan to find dense regions. To avoid these time-consuming database scan the first filter identifies possible dense regions (MICHs) in each subspace. Depending on the distribution and the dimensionality of the number of MICHS is typically extremely high. Hence, an efficient implementation of the hypercube approximation filter is important to obtain a good runtime behavior for our eDUSC approach.

3.5 Indexing subspace clusters

In this section we demonstrate that indexing is indeed beneficial for a depth-first mining approach as eDUSC, and especially for in-process-removal of redundant subspace clusters. To index subspace regions as mined by the hypercube approximation filter, we utilize the transformation from the original space to the representation given by the density-conserving grid (see Chapter 3.4.1). In the next section, we shortly describe the transformation of points to cells and the basic idea underlying our index structure. This is followed by a detailed description of the index itself. We then describe how the SC-tree supports the individual steps of our eDUSC algorithm. Hence the SC-tree

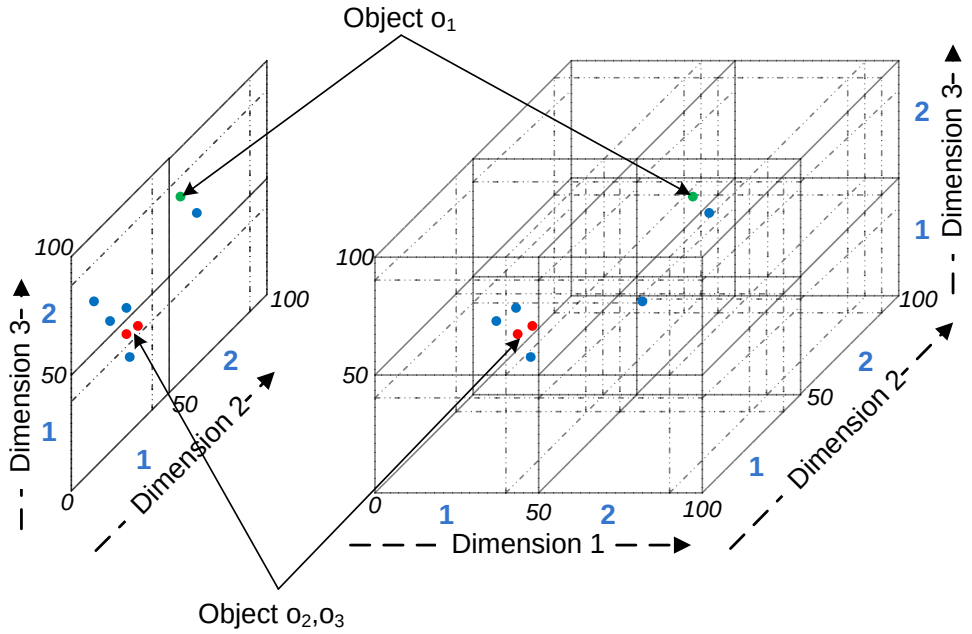

Figure 3.9: Example data set: transforming objects to cell representations

also supports an efficient way to implement the filter steps of the eDUSC algorithm.

3.5.1 Transformation

To efficiently mine maximal induced cluster hypercubes (MICHs) in a novel indexing structure, we utilize the compact representation of regions by cells. The density conserving grid as proposed in Chapter 3.4.1, allows for transformation of the data objects to cell memberships. Working on these cell memberships, we propose a novel index structure for depth-first subspace clustering in the next section.

Recall that density conserving grids contain additional connectivity borders to detect any subspace cluster. Subspace clusters which are density-connected across individual grid cells are detected along the connectivity borders between cells.

We use a three dimensional data set as a running example in this section. Figure 3.9 illustrates the three dimensional representation in the right part and a two dimensional projection of the data set to the subspace $\mathbf{S} = \{2, 3\}$ in the left part. The attribute range is partitioned into two equiwidth in-

tervals $p_i = [50(i-1), 50i],\ i = 1, 2$. Each subspace cell is identified by a two-dimensional vector of the interval numbers. In the illustration, interval numbers are illustrates using a blue color. We use the same notation for cells and hypercubes as in the last chapter (see Definition 3.4).

To detect clusters across grid cells, connectivity borders provide additional information. Connectivity-borders are areas of ε-width at the upper border of each cell in each dimension. If density-connected subspace clusters stretch across the connectivity border, at least one object has to be contained inside the border area. Otherwise, as the neighborhood range ε is exactly the same as the border width, objects in either cell cannot be in the neighborhood of each other (see Definition 3.3). Recall that we mark dimensions with "-" to denote connectivity borders for this dimension.

Based on the regular grid cells and the connectivity borders we define the following transformation: any object o contained in a grid cell (or border) $\mathbf{C}_{\alpha_\mathbf{1},\ldots,\alpha_\mathbf{d}}$ is transformed from the original object representation to a set of corresponding indexes per dimension.

Reconsidering Figure 3.9, we see that object o_2 and o_3 (illustrated by a red color) are both positioned in cell $\mathbf{C_{1,1,1}}$ with connectivity border $B_{1,\bar{1},1}$. Hence, o_2, o_3 can be compactly represented as: $\{(1,1),(2,1),(3,1)\}$, that is grid cell interval one in dimension 1, interval two in dimension two and interval one in dimension three. Since object o_2 and o_3 are contained in one border, they are additionally represented as $\{(1,1),(2,\bar{1}),(3,1)\}$. Note that objects in the same cell are transformed to the same representation: o_2, o_3 are both transformed to {(1,1),(2,1),(3,1)}. Another example is given for object o_1 (colored green) which is positioned in cell $\mathbf{C_{1,2,2}}$ and border $B_{\bar{1},2,2}$. Consequently o_1 is transformed to the cell representation $\{(1,1),(2,2),(3,2)\}$ and the border representation $\{\overline{(1,1)},(2,2),(3,2)\}$. In this example the objects (o_1, o_2, o_3) are the only objects positioned in a connectivity border. Thus all other objects are only transformed to one cell representation.

Definition 3.5 ***Transformation.***

*A **transformation** according to the density conserving grid (seen Definition 3.4) of a continuous object $o = (o_1, \ldots, o_d)$ contained in cell $\mathbf{C}_{\alpha_\mathbf{1},\ldots,\alpha_\mathbf{d}}$ is defined as a mapping of o to a set of descriptors*

$$o \mapsto \{descriptor_1, \ldots, descriptor_d\} \text{ with } descriptor_i = (i, \alpha_i),$$

where α_i denotes the interval in dimension i. Similarly, for each border j that o is contained in $(\mathbf{B}_{\alpha_1,\ldots,\overline{\alpha_{i_j}},\ldots,\alpha_d})$, o is mapped to a set of descriptors

$$o \mapsto \{descriptor_1, \ldots, descriptor_d\} \text{ with } descriptor_i = (i, \alpha_i), i \neq i_j$$

$$descriptor_{i_j} = (\overline{i_j, \alpha_{i_j}})$$

Decomposition into interval indexes per dimension is a useful transformation for subspace clustering, where different combinations of dimensions have to be mined. That is, each object is transformed to exactly one cell descriptor, and to zero or at most d border descriptors, depending on the number of borders it is contained in. This transformation from feature space to grid cell memberships allows for compact indexing of subspace clustering information. Subspace clustering on the original database is reduced to mining of membership counts in discrete cell intervals. We therefore propose building a compact index on cells and borders and their respective object counts.

3.5.2 SC-tree structure

Using a compact index structure we avoid repeated database scans. In a single scan on the database, we retrieve all information necessary to compute density information about grid cells and borders. This information is stored in the tree to allow efficient mining without additional database scans. As described in the last section each database object is transformed into a set of descriptors $\{descriptor_1, \ldots, descriptor_d\}$ and possibly more sets containing information for each border an object is positioned in. Each descriptor set representing a cell or border is next inserted into the SC-tree.

The nodes of a SC-tree are labeled with a tuple (*descriptor*, *count*). To insert a descriptor set into the SC-tree, the descriptor set is ordered in a lexicographical way. The descriptors $(dimension, interval)$ are first ordered according to their dimensions and then according to their interval number. Beginning with the lexicographical first descriptor the SC-tree creates a path for each descriptor set beginning from the root. If a corresponding node for a descriptor already exists the SC-tree increments the count information for this node (a newly created node always has a count 1). Hence, the first inserted dimension is distinguished at the root and the last one at the leaf

level. For each descriptor set which contains a border the count information is not increased (new created nodes get a count of 0).

After all objects have been added to the SC-tree the initial SC-tree is completed. One path in the initial SC-tree describes a region of the discretized data space. The count value in a node specifies the number of objects in the region described by the path from the root to this node. Since all descriptors have the same length (the dimensionality of the data space), all paths from the root node to a leaf have the same length. As each object increases the count of one node of each level by one, the sum of all counts of one level corresponds to the size of the database (please note again, that border descriptors do not increment any count information). Hence, the initial SC-tree contains all information about the discretized regions but also about the borders, allowing a very compact representation of the total information needed by the hypercube approximation filter of our eDUSC approach.

Figure 3.10 shows the initial SC-tree for our three dimensional example database. For a better visualization the dimension part of a descriptor is shown on each level and not in the nodes. As all objects have been transformed to sets with one descriptor per dimension, every path from the root to a leaf has length three. The border descriptors are represented by a dark shaded node.

For example the green colored object is transformed to the descriptor sets $\{(1,1),(2,2),(3,2)\}$. The green marked path of the SC-tree depicted in Figure 3.10 represents this descriptor set. The count information annotated at the leaf node specifies that only one object is contained in cell $\mathbf{C_{1,2,2}}$. Similar the two red colored objects (descriptor set $\{(1,1),(2,1),(3,1)\}$) are represented by the red path. As one additional object is contained in cell ($\mathbf{C_{1,1,1}}$) the count information 3 is annotated at the leaf node. In this example we colored inner nodes using the color of all objects represented by the node (e.g. node 1 in dimension 1 is colored red and green). As mentioned before border descriptors are illustrated by dark gray elliptical-shaped nodes, e.g. the two red objects positioned in border 1 of dimension 1 are represented by the by the red-gray shaded nodes. The green object contained in border 1 of dimension 3 creates one single path (represented by the green-gray shaded nodes). This path has no count information, because it is only an indicator that there is an object in the respective border region. Summing up the counts on one level we see that the example database consists of 8 objects.

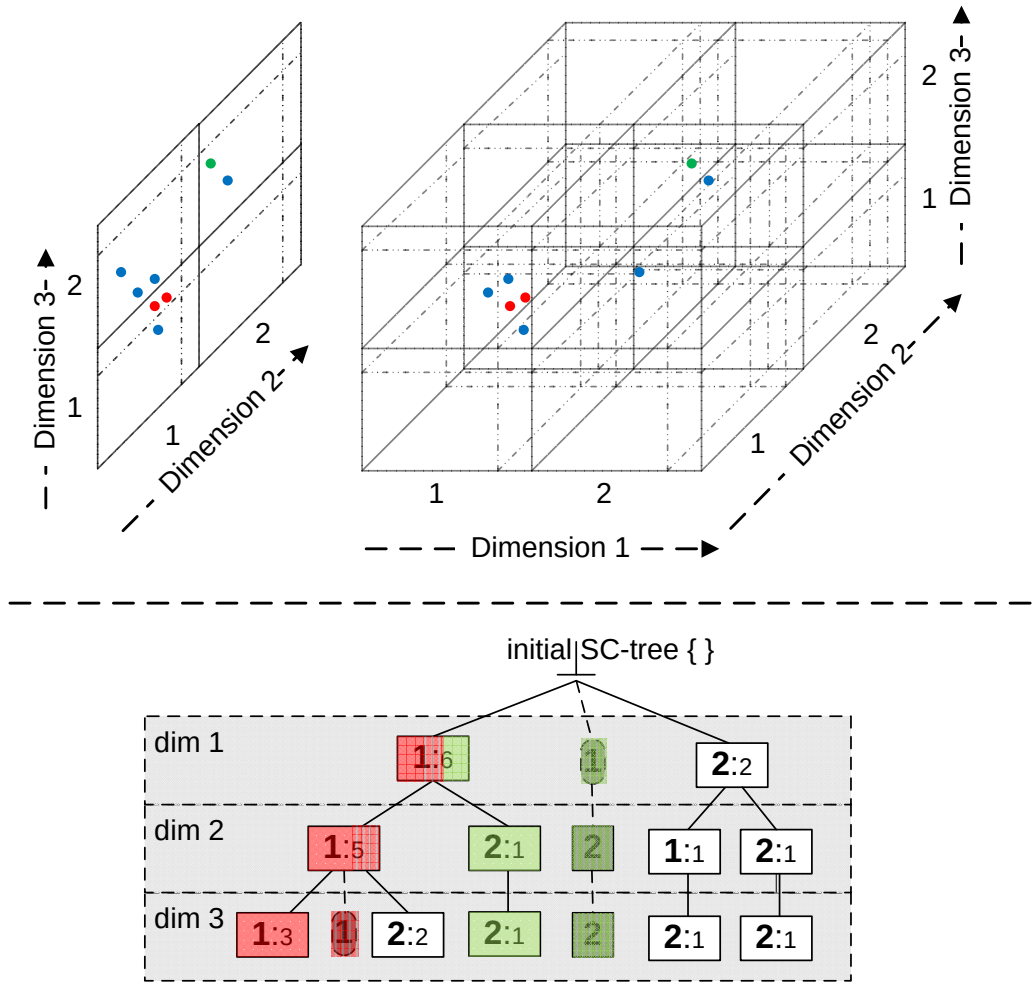

Figure 3.10: Creating the initial SC-tree for the example data set

3.5.3 Mining subspace clusters using the SC-tree

In this section we present how the SC-tree can be used for an efficient implementation of the *eDUSC* approach. Especially the hypercube approximation filter benefits from using the SC-tree. Further more, the SC-tree makes in-process pruning of regions during the mining procedure possible.

Algorithm 6 illustrates the integration of the SC-tree into the *eDSUC* approach. The algorithm shortly sketches the main steps of the filter-and-refinement architecture of *eDUSC* based on the SC-tree (see also Algorithm 5). Please note that as from this point of the chapter *eDUSC* is meant as the integration of the SC-tree into the *eDUSC* algorithm.

Following the $eDUSC$ approach the initial SC-tree representing the entire database is recursively restricted using each descriptor (each interval in each dimension) (Line 2). This step corresponds to the $restrictHypercube$ step of the $eDUSC$ algorithm presented in Chapter 3.4. The restricted SC-tree only contains descriptor sets which contain the respective descriptor.

Next merges are detected and induced into future cells by restricting the SC-tree to the corresponding border (Line 3). If the border descriptor exists the connectivity border contains at least one object (see $TestBorders$ of $eDUSC$) and hence a merge is induced into the respective cell. In this case the restricted SC-tree is stored for further use (Line 4). To ensure that no clusters are cut apart by the used grid representation merges are performed if necessary (Line 5). Next the monotonicity properties presented in Chapter 3.4.2 are used for pruning regions (Line 6). If a restricted region cannot be pruned, the region is restricted in a further dimension through the recursive calls (Line 7) of the $eDUSC$ procedure. To ensure non-redundant clustering in-process-removal of redundant clusters is performed in Line 8. Having mined all higher dimensional subspaces by returning from the recursive call, the redundancy properties are checked for the current SC-tree. This step improves efficiency as pruning redundant SC-trees is possible without costly access to the database and redundant clusters can be pruned during the mining process. Only if a region cannot be pruned the density based filter discussed in the last section computes the final result.

After having given this brief description of the SC-tree we next describe the important parts of the mining method in detail. Therefore, for each step an abstract description will be given followed by an example application of the step on a running example. As parameters for the example we use $minSize = 3; \delta = 50\%$.

Restricting the SC-tree

Starting from the initial SC-tree representing the entire database with no restricted dimensions, the given SC-tree is restricted in further dimensions through recursive calls. Each call restricts the current SC-tree for each descriptor. The restrictions are done in lexicographical order of the descriptors (dim_i, int_k) starting from the leaf nodes of the tree.

Restricting a SC-tree to a descriptor (dim_i, int_k) means that all paths which do not contain the respective descriptor are removed from the SC-

Algorithm 6: *eDSUC*(SC_tree, result)

```
1  foreach descriptor in SC_tree do
       // restrict given SC-Tree in a further dimension
2      SC_tree_restrict := restrict(SC_tree, descriptor);
       // restrict given SC-Tree to corresponding border
3      SC_tree_border := restrict(SC_tree, borderOf(descriptor));
       // induce merges into future cells
4      induce(SC_tree_border, SC_tree_restrict);
       // merge restricted tree with neighboring trees
5      merge(SC_tree_restrict);
       // pruning based on monotonicities
6      pruneRecursion(SC_tree_restrict);
       // recursive call of DUSC
7      eDUSC(SC_tree_restrict, result);
       // prune redundant clusters
8      pruneClustering(SC_tree_restrict);
       // get objects from tree
9      C = getObjects(SC_tree_restrict);
       // get subspace from tree
10     S = getSubspace(SC_tree_restrict);
       // density-based filtering
11     result := DensityBased_Filter(C, S) ∪ result;
```

tree. Hence, after the restriction only those objects that are positioned in interval k of dimension i are represented by the restricted tree. Finally, the subpaths which only represent the descriptor are removed from the SC-tree. The restriction thereby can be interpreted as a projection of the data to the descriptor (dim_i, int_k). As during recursion, one restricted SC-tree is restricted in more and more dimensions, these restrictions are stored for each SC-tree as $\{descriptor_1 \times \ldots \times descriptor_n\}$ with $n \leq d$. For a multiply restricted SC-tree this means that it can be obtained starting from the initial SC-tree and performing restrictions for the n descriptors it is restricted to. Please note, that paths representing a border remain if they contain the descriptor. This method can also be applied in the same way for border descriptors. For efficiency considerations, all nodes labeled with the same

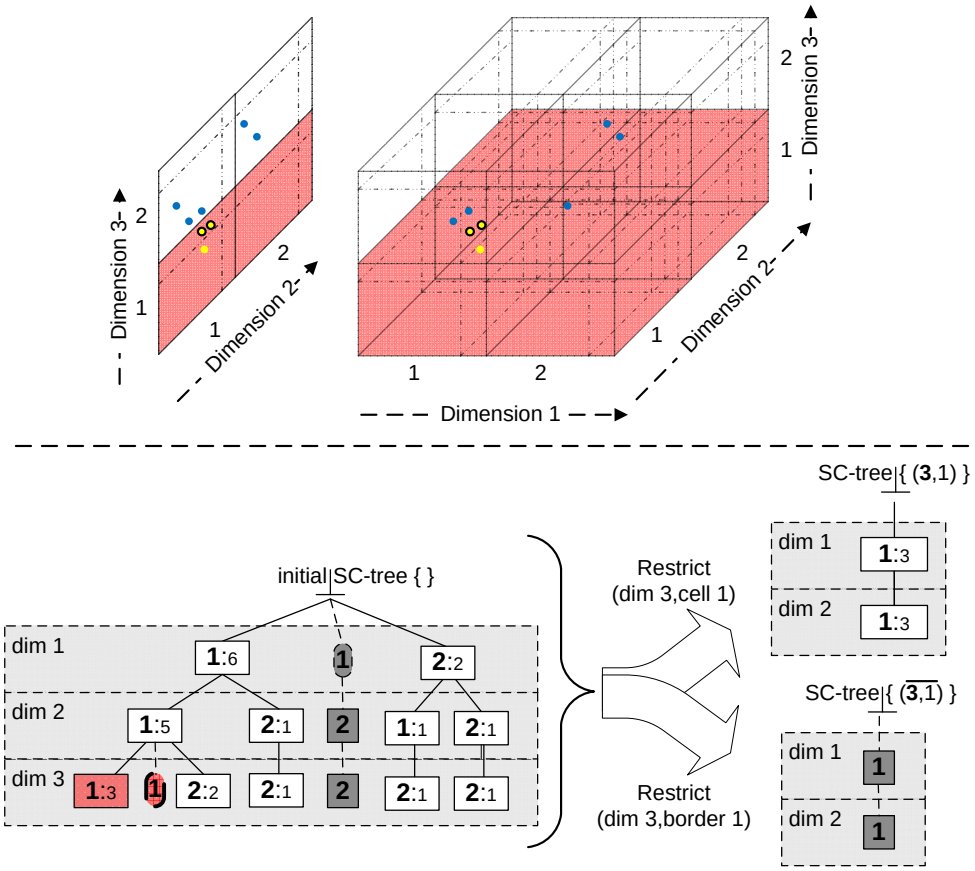

Figure 3.11: Restricting the SC-tree

descriptor are part of a linked list. This step is similar in spirit to conditional FP-growth steps in association rule mining that builds conditional subtrees for frequency counts [HPY00].

Example: Figure 3.11 illustrates the restriction step for the descriptor $(3, 1)$ and the border descriptor $\overline{(3, 1)}$. For each node labeled with this descriptor the paths to the root node are copied to the restricted SC-tree $\{(3, 1)\}$ but labeled with the count information of the restricted node. In our example the only node corresponding to the descriptor $\overline{(3, 1)}$ is the first node from the left annotated with 3 (see Figure 3.10). The corresponding 3 objects that are contained in interval 1 of dimension 3 are yellow highlighted in the corresponding graphical representation (upper part of Figure 3.11). The new SC-tree therefore only contains one path labeled with count 3. Intuitively, this restricted tree represents the three objects in the $(3, 1)$ region and could

also be achieved by inserting the descriptor-transformation of these three objects $\{(1,1),(2,1)\}$ into an empty SC-tree. As discussed above the same method can be used to restrict a SC-tree to a border. Since the count information is not stored for border descriptors the resulting SC-tree does not contain any count information (see also Figure 3.11). The objects contained in the border are black encircled in the graphical representation.

Inducing merges

Following the eDUSC approach a merge is induced into a future cell if the corresponding connectivity border contains at least one object. The SC-tree includes special nodes to represent the connectivity borders. Hence, testing if a border contains an object can be implemented by restricting the SC-tree to the respective border descriptor. If the tree is not empty the connectivity border contains an object. In this case a merge is induced as described in the last section.

Example: Using our running example again, the SC-tree is first restricted in a second dimension since one-dimensional cells do not induce merges (see Algorithm 5). The obtained SC-tree which is restricted to the descriptors $\{(3,1),(2,1)\}$ is presented in Part A of Figure 3.12. In the upper part of Figure 3.12 the area corresponding to the SC-tree is marked red and the contained objects are colored yellow.

To detect necessary merges the two borders $\{\overline{(3,1)},(2,1)\}$, $\{(3,1),\overline{(2,1)}\}$ have to be checked. The SC-tree for $\{(3,1),\overline{(2,1)}\}$ can be computed from the SC-tree $\{(3,1)\}$. As the obtained SC-tree is empty no merge is induced into the corresponding neighboring cell (Part B of Figure 3.12). The second border is checked by restricting the SC-tree $\{\overline{(3,1)}\}$ to the cell descriptor $\{(2,1)\}$ (Part C of Figure 3.12). Since the SC-tree $\{(3,1),\overline{(2,1)}\}$ exists a merge is induced into the upper cell. The two objects contained in the connectivity border are black encircled in the graphical representation (upper part of Figure 3.12). To perform merges the corresponding SC-tree is needed and hence we store the SC-tree $\{\overline{(3,1)},(2,1)\}$ for further use.

Performing Merges

Merging two regions can be implemented by merging the SC-trees representing the regions. Merging means that each path of one tree is added to the other tree. New nodes are created if parts of a path do not exist and the

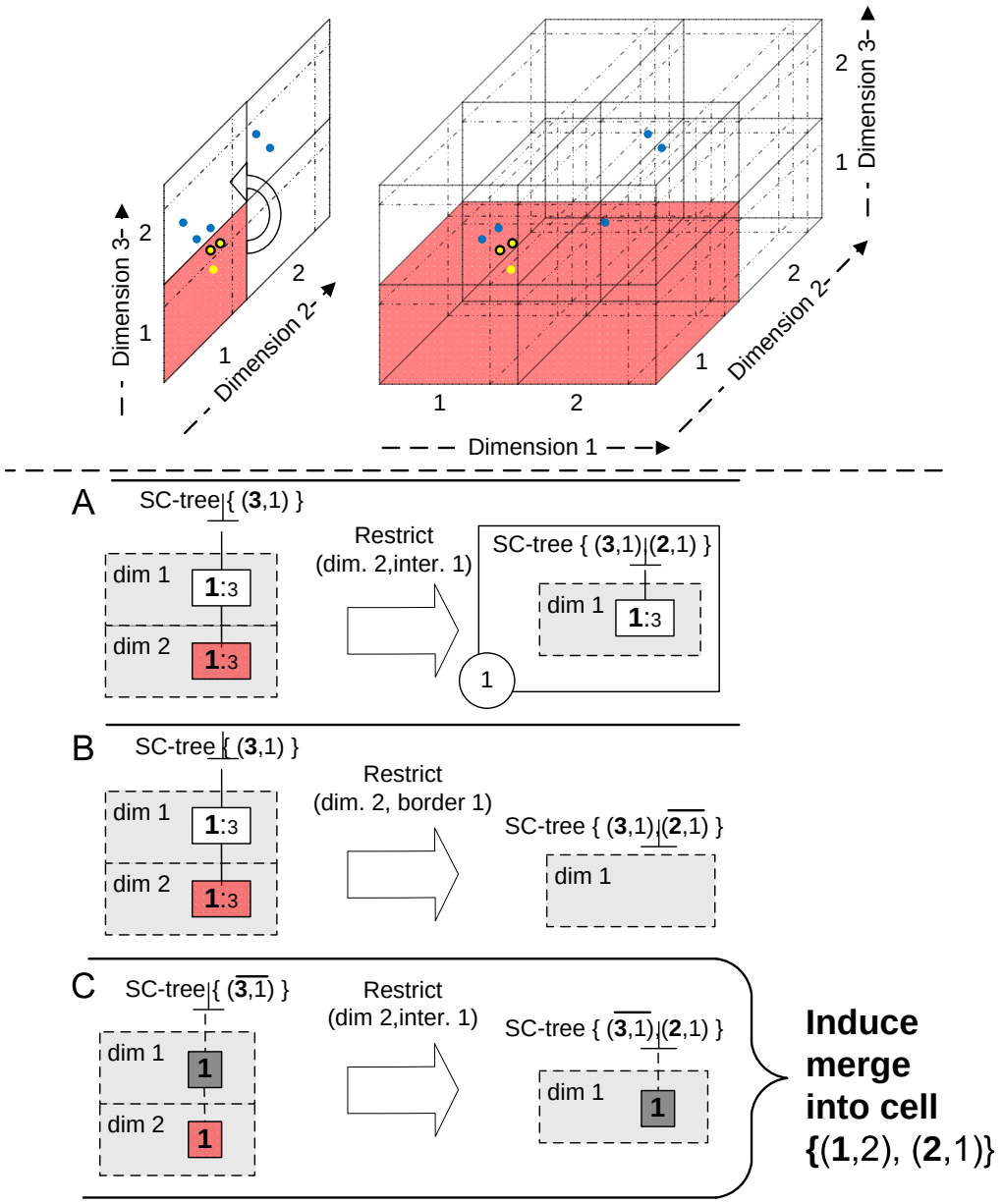

Figure 3.12: Inducing merges using the SC-tree

annotation of existing nodes is aggregated. The resulting SC-tree describes all objects contained in the combined region and additionally contains the information about the dimensions which are not restricted so far. Hence, mining higher dimensional subspace cluster based on the merged SC-tree is possible.

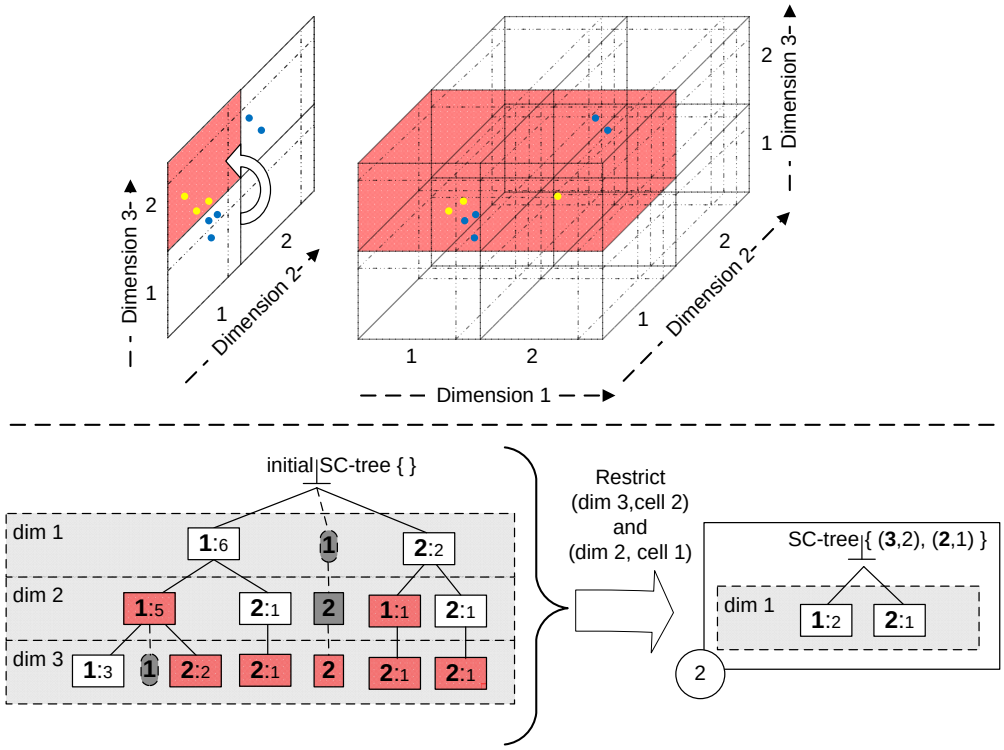

Figure 3.13: Processing the next cell using the SC-tree

Example: As mentioned above a merge is performed if the corresponding region is processed. In Figure 3.13 the restriction of the initial SC-tree to the region $\{(3,2),(2,1)\}$ is illustrated. After the two restriction steps the SC-tree still contains two nodes describing the two cells in dimension one. Next the merge with cell $\{(3,1),(2,1)\}$ is performed which is illustrated in Figure 3.14. In this example the second SC-tree already contains the nodes of the merged region. Hence, the merged tree corresponds to the second tree added the count information of the first tree. As depicted in the upper part of Figure 3.14, the resulting SC-tree describes the objects contained in the merged red marked region.

Pruning recursive calls

Two monotonicity properties have been introduced in the last section to prune a region from further consideration. Pruning property (A) can directly be implemented using the SC-tree. If a MICH is found by a SC-tree

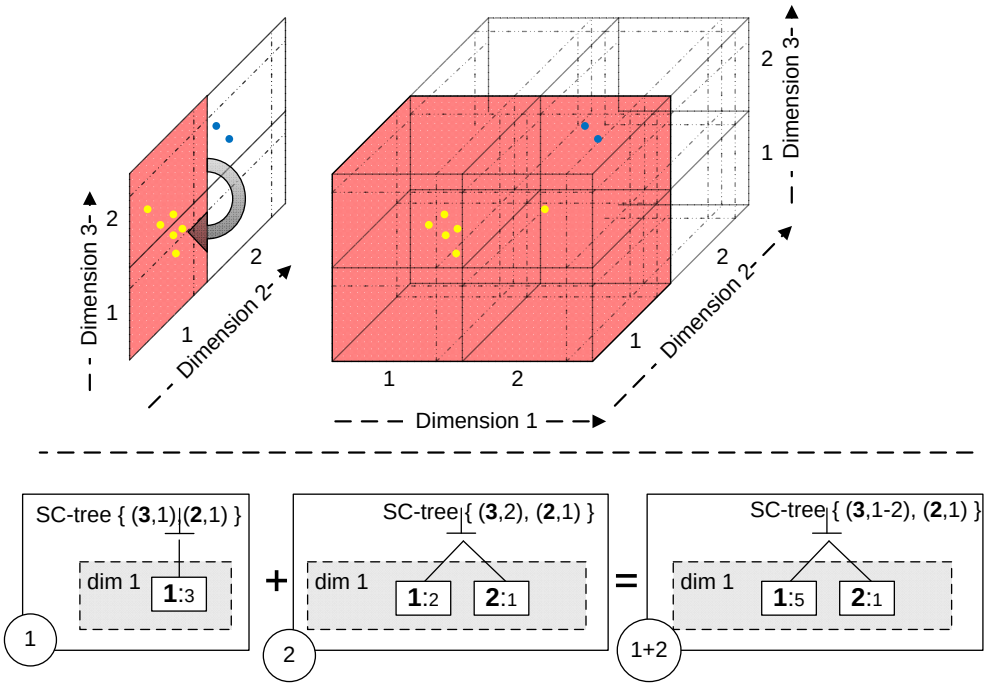

Figure 3.14: Merging cells using the SC-tree

(no more merges have been induced) the SC-tree completely encloses a possible density-connected region. According to monotonicity property (A) this region can be pruned and hence the recursion of $eDUSC$ is stopped if the SC-tree contains less than $minSize$ objects.

Example: Pruning according to the $minSize$ parameter is illustrated in Figure 3.15. The merged SC-tree contains 6 objects and hence can not be pruned. Following the depth-first approach the SC-tree is next restricted to interval 1 in dimension 1. The corresponding node (colored light blue) contains 5 objects and can not be pruned, either. Since no object is contained in the respective connectivity border the SC-tree is next restricted to descriptor $(1, 2)$. The SC-tree representing the cell $\{(3, 1\text{-}2), (2, 1), (1, 2)\}$ only contains 1 object (green node in the SC-tree 1+2 depicted in Figure 3.15). The count information of 1 means that only one object is contained in the corresponding cell (green area in the upper part of Figure 3.15). Consequently the region can be pruned and the time consuming database scan of this region is not necessary.

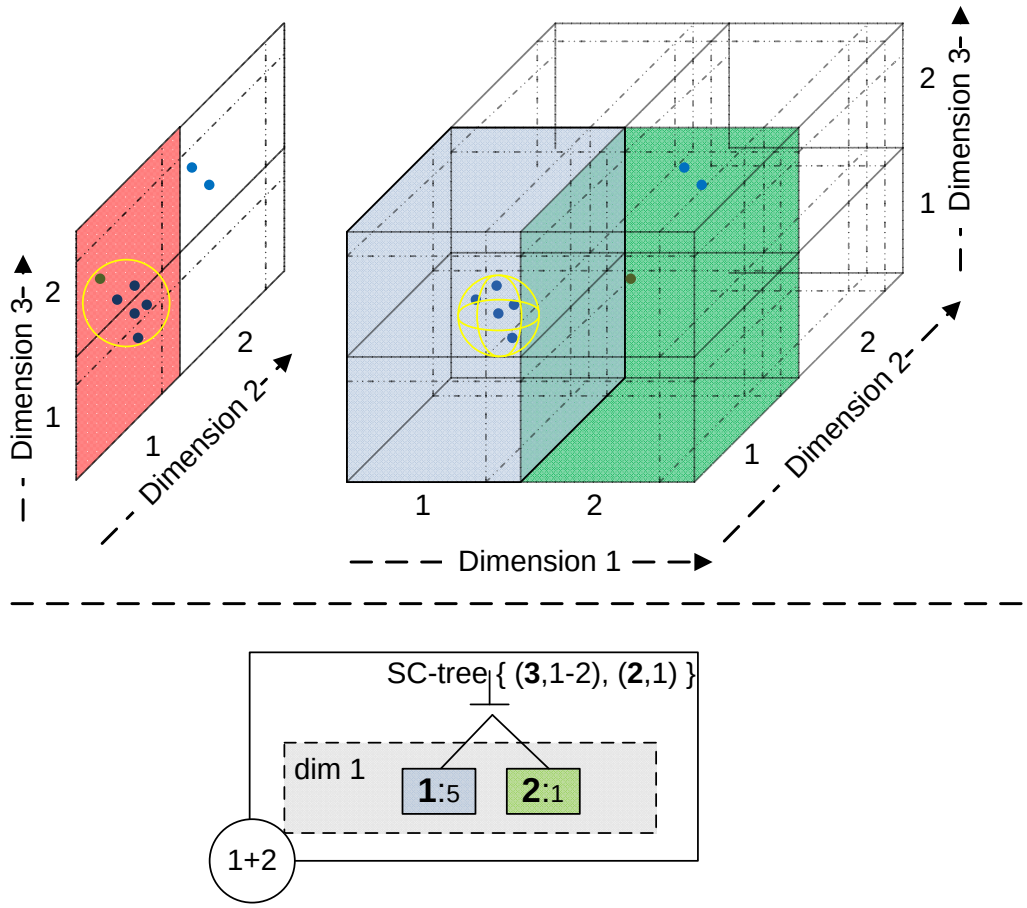

Figure 3.15: Pruning based on the SC-tree

In-process redundancy pruning

If a region can not be pruned the last step in identifying a cluster is to find density-based clusters. This step requires a database scan to identify the area of influence for the objects contained in the restricted region (see Definition 3.3).

As mentioned in previous sections high-dimensional clusters typically show up in lower-dimensional projections as well. These projections form redundant subspace clusters which contain essentially the same information. Before performing a density-based clustering the degree of redundancy for a region is checked by *eDUSC* without accessing the original database. This step greatly improves efficiency as time consuming clustering redundant regions is avoided. To check if a region is redundant the method

pruneClustering determines if a higher-dimensional cluster exists which contains objects of the current region. While stepping back in recursion, only SC-trees containing potentially non-redundant clusters (with respect to the redundancy parameter δ) are mined with *DenseCluster*.
Example: By restricting the merged SC-tree $\{(3, 1\text{-}2), (2, 1)\}$ to the first cell in dimension 1 to $\{(3, 1\text{-}2), (2, 1), (1, 1)\}$ (illustrated by the light blue node in Figure 3.15) we obtain an SC-tree which contains 5 objects (more than *minSize*). As the highest dimensionality is reached the region can not be redundant and hence is clustered. In this example a dense cluster containing all 5 objects is found (see also Figure 3.15).

Stepping back in the recursion, we already found one higher-dimensional cluster. Redundancy pruning can be performed by checking the SC-tree $\{(3, 1\text{-}2), (2, 1)\}$. This SC-tree contains 6 objects, the 5 objects belonging to the 3-dimensional cluster and one additional object (the dark green object contained in the red region of Figure 3.15). Using a redundancy parameter of $\delta = 0.75$ this cluster must at least have $\frac{5}{0.5} = 10$ objects to be non-redundant. Thus, *DenseCluster* is not performed on the current region, because these objects are totally redundant to the cluster found in the 3-dimensional subspace.

Density-based scan are also not necessary for the other two regions as the SC-tree for $\{(3, 1), (2, 2)\}$ is empty and the SC-tree for $\{(3, 2), (2, 2)\}$ only contains two objects (which is less than *minSize*). Similarly, the other regions can be pruned. Hence, *eDUSC* only needs one database scan on a region containing 5 objects to cluster the example data set in all subspaces.

In-time-availability of SC-trees

In the last section we have discussed efficiency in terms of pruning the search space and reducing the number of necessary database scans. Since we use the SC-tree as an index structure to ensure faster mining, it is important that SC-trees are are available if a restriction or merge step is necessary. Hence, in this section we discuss frequently used operations like restrictions and merges in more detail. We show that the process order of *eDUSC* approach ensures that each SC-tree needed for the next step is already available. This can be guaranteed as they have been created in an earlier step.

Recall the merge step explained in the example in Section 3.5.3. Starting from a restricted SC-tree $\{(3, 1)\}$, one border which was checked is obtained

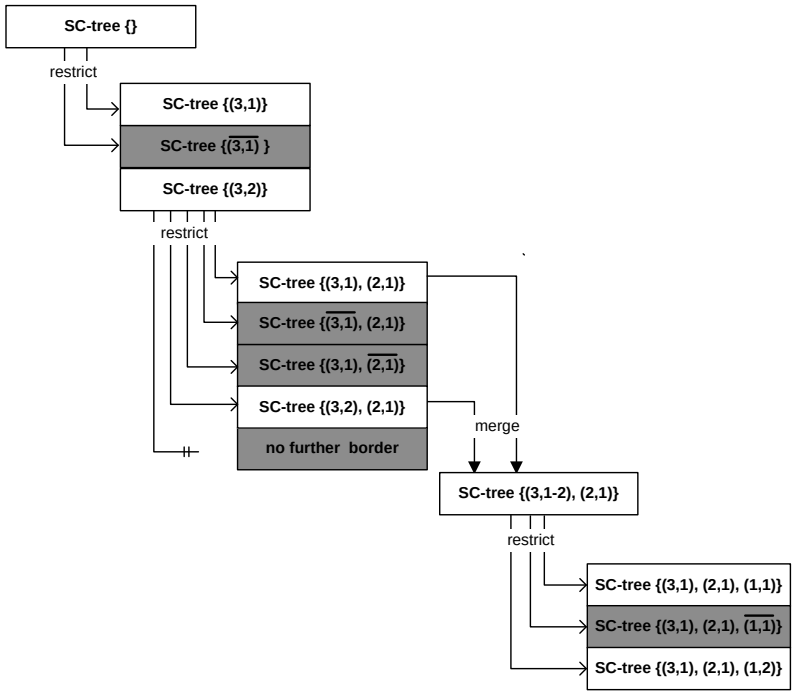

Figure 3.16: SC-tree availability during mining

by restricting the tree to the border $\{\overline{(2,1)}\}$. The restricting is possibler by using the SC-tree created in the first step. As a merge was induced, the SC-tree is stored and merged with the SC-tree $\{(3,2),(2,1)\}$. The merged SC-tree $\{(3,1-2),(2,1)\}$ was then used for further restrictions. Figure 3.16 represents these steps by displaying the names of the used SC-trees. Obviously all restrictions are performed in a hierarchical way on the given initial SC-tree. Also, the detection of necessary merges is performed on these trees. Only the resulting merged SC-tree is then needed for the recursive call of *eDUSC* and which is then restricted in further dimensions.

As the index structure has been tuned with respect to the subspace mining process, the information needed for the current and following steps is available. Moreover, there is no irrelevant information for other subspaces thatnare not mined in the current recursion path. This is automatically realized by the restriction phase where only the information concerning the current descriptor is transferred into the restricted SC-tree. This reflects the general idea of the restricted index storing only the necessary information. The initial SC-tree realizes this as well, so during mining, not only costly

database scans, but also unnecessarily large trees are avoided.

Correctness of in-process removal

eDUSC removes redundant clusters in-process which implies that all clusters necessary to identify a redundant cluster have been mined before. We proof the correctness of this step by showing that first, all non-redundant clusters are in the resulting list and second, that there are only non-redundant clusters in the list. Bear in mind that a cluster is non-redundant if the objects have not been found in a higher dimensional cluster with respect to the redundancy parameter δ in Definition 3.3. Thus the first aspect is clear as an SC-tree is only pruned if the region can only contain redundant clusters with respect to the clusters already contained in the result list. Next we have to prove the following theorem:

Theorem 3.2 *The result set of eDUSC only contains non-redundant clusters.*

Proofing this theorem we mainly based on the fact that *eDUSC* mines in a depth-first manner and hence higher dimensional clusters are mined first.
Proof: Assume *eDUSC* has a redundant cluster (C_l, S_l) in its output. As it is redundant a higher dimensional cluster (C_h, S_h) exists with:

$$C_h \subseteq C_l \wedge S_l \subset S_h \wedge |C_h| \geq \delta \cdot |C_l|.$$

We show that this assumption cannot be true in two cases:

1 (C_h is mined before C_l)
Before inserting C_l in the result list, the redundancy pruning (Line 8 of the *eDUSC* algorithm) has to be performed. As the inequation above is true, no *DenseCluster* is performed and thus C_l cannot have been inserted into the result list.

2 (C_h is mined after C_l)
As C_h is a higher dimensional cluster of C_l, the subspace S_l consists of all dimensions that are considered relevant for cluster C_l: $S_l \supset S_h$. Due to the processing order of depth-first search, *eDUSC* had to further restrict the subspace S_l to S_h before processing S_l with *DenseCluster*. Thus case 2 cannot occur.

As we showed, *eDSUC* mines all non-redundant subspace clusters with respect to Definition 3.3. Furthermore, we show in Section 3.6 that the removal of redundancy yields not just better runtimes but also better quality.

Using sort heuristics for efficient SC-tree construction

To further improve the efficiency of the *eDUSC* approach we developed different heuristic to construct the SC-tree. The order of the descriptors is decisive for the creation of the initial SC-tree and the subsequent mining procedure. For the *eDUSC* algorithm described in Section 3.5.3 the given order of dimensions and intervals was used. This is the intuitive way of sorting the descriptors as the first restriction is done on the leaf level using the first dimension and so on (see example in Section 3.5.3). Aiming a more efficient processing, this order can be changed by the heuristic discussed next.

Based on a heuristic used in the FP-tree [HPY00] from association rule mining the size of a tree structure can be reduced by re-sorting the inserted itemsets. In association rule mining FP-trees are only used for frequent itemset mining, so FP-trees do not need to be merged to gain completeness and the most expensive operation is the projection of the data to gain projected FP-trees. For this kind of algorithm the size of the tree structure is the limiting factor for efficient processing. Thus the sorting heuristic for the FP-tree aims at reducing the tree size by sorting the items by their frequency. By inserting the items first that occur very frequent these items create paths that are most probably also used for other itemsets. This possible re-usage of existing paths should lead to fewer nodes and thus to a more compact tree representation of the data.

Adapting such a heuristic to the subspace clustering problem is not appropriate. One main efficiency gain is achieved by the composite order on the descriptors (dimension, interval) first ordered by the dimension and then by the interval number. The mining process is directed by this composite order, so for the SC-tree this order cannot be discarded. The main point for the ordering as discussed previously is that each level of the SC-tree represents one dimension so that in one restriction step all irrelevant intervals of the current dimension can be removed. Also, the border entries in the given SC-tree can indicate merges with the successive interval on the same level.

It is clear that, the SC-tree needs a composite ordering of the descriptors (dimension, interval) but there is no constrain given for the ordering of the

two parts inside the composite order. As the ordering of the intervals is naturally given by the starting point of the interval and a different order could cause complications for region growing in the merge step, we focus our sorting heuristic on re-sorting the dimensions. Using the same aim to reduce the size of the tree, we sort dimensions with respect to their scattering. A dimension in which the data is distributed over a wide range should not be inserted first in the tree. Instead, a dimension in which the data is clustered in one region can lead to many common paths in the tree when inserted first.

A second and more interesting aim for mining is the reduction of runtime, which in our case is limited to the *DenseCluster* with database access and to the merge operations that have to be performed on SC-trees. *DenseCluster* is already pruned due to in-process-removal of redundancy. Merges cannot be pruned as they are necessary to ensure the completeness of the *eDSUC* approach. Yet many merges can be avoided with a different processing order. Mining scattered dimensions with noisy data at first forces many merges on big SC-trees as no other dimensions have been restricted yet. These merges probably do not lead to subspace clusters at all. By mining the scattered dimensions last, *eDSUC* needs to perform merges only on very small SC-trees and this can be beneficial for the overall runtime.

For the SC-tree, aiming to avoid merges in the first steps of mining process contradicts to the previous order with the scattering dimensions on the leaf level. The dimensions that contain clustered data have to be inserted last, because restriction starts with the leaves. Starting at the leaf-level, *eDSUC* has the possibility to keep all other dimensions for further restrictions and thus mine the highest dimensional clusters first.

Entropy is an information theoretic indicator for the homogeneity of the data and can be used to measure the distribution in each dimension. Given the discretized data space for each dimension and thus also the percentage f_i of objects in each interval $i = 1 \dots g$ (intervals positioned as described in Definition 3.4) entropy is calculated by:

$$E(f_1, \dots, f_g) = -\sum_{i=1}^{g} f_i \cdot \log(f_i)$$

Using entropy for each dimension, both aims discussed above can be achieved. Inserting dimensions with low entropy values first (ascending order) realizes the first sorting heuristic for small trees, while inserting dimensions

with high entropy values (descending order) realizes the second heuristic for fast subspace mining. Both ascending and descending order will be analyzed in experiments in the following section.

3.6 Experiments

In thorough experiments we demonstrate the performance of the *eDUSC* algorithm based on the SC-tree in terms of efficiency as well as in terms of successful removal of redundant subspace clusters. *eDUSC* as the first depth-first approach is compared to the apriori based breadth-first algorithm SUBCLU [KKK04]. SUBCLU is the most recent approach to detect density-based subspace clustering without approximation. To enhance neighborhood queries and density computations SUBCLU uses inverted files on individual dimensions. As discussed in the original paper, other indexing techniques are not beneficial for breadth-first subspace clustering. To the best of our knowledge, there is no other indexing approach for subspace clustering. Thus, we additionally evaluate the overall runtime performance in comparison with an approximative grid-based approach, SCHISM [SZ04]. SCHISM works in a bottom-up manner and is tuned for efficiency at the cost of accuracy. Since SCHISM loses clusters in grid discretization and pruning of potentially irrelevant grid cells, it is only included in the general performance experiments as a baseline comparison. Its subspace clustering result set, however, is not comparable. Experiments were run on Pentium 4 machines with 2.4 Ghz and 1 GB main memory.

3.6.1 Synthetic data set

Based on properties of real world data sets we generate synthetic data for scalability experiments. We extend a method proposed in SUBCLU [KKK04] to generate density-based clusters in arbitrary subspaces. Given the subspace and the number of objects for each cluster, dense regions separated by noisy regions are created. Objects can belong to multiple subspace clusters, just as in most real world data sets. We generate data of different dimensionalities and hide subspace clusters with a maximal dimensionality of 80% of the data dimensionality. Four subspace clusters are hidden in the synthetic data with two of them overlapping in 10% of their objects. The subspace clusters were

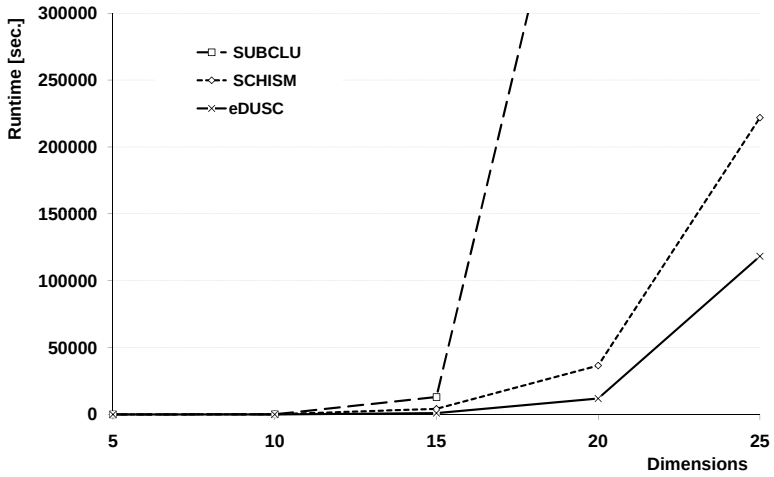

Figure 3.17: Scalability vs. dimensions

generated in a normalized data space with an attribute range of 0 to 100 in each dimension.

Scalability w.r.t. dimensionality

Scalability in terms of dimensionality is important, as subspace clustering aims at clustering in high dimensional data where "full-space" clustering is typically infeasible. As the number of possible subspace clusters depends exponentially on the dimensionality of the subspace, clustering high dimensional data sets is challenging for many subspace clustering algorithms.

As depicted in Figure 3.17, SUBCLU does not scale w.r.t. dimensionality. For the 20-dimensional data set the algorithm did not even finish after six days. The reason is the tremendous amount of 1 and 2-dimensional subspaces that have to be clustered before processing any higher dimensional subspace.

SCHISM as a grid-based approach has better runtimes as SUBCLU but also suffers from apriori-based candidate generation overhead. Surprisingly at first glance, eDUSC scales better than the approximative grid based algorithm SCHISM in terms of dimensionality. This is due to the fact that SCHISM relies on a fixed pruning threshold u for lower dimensional subspaces. In high dimensional subspaces, as the fixed threshold loses its power, runtime of SCHISM goes up.

The eDUSC algorithm with its depth-first approach overcomes this prob-

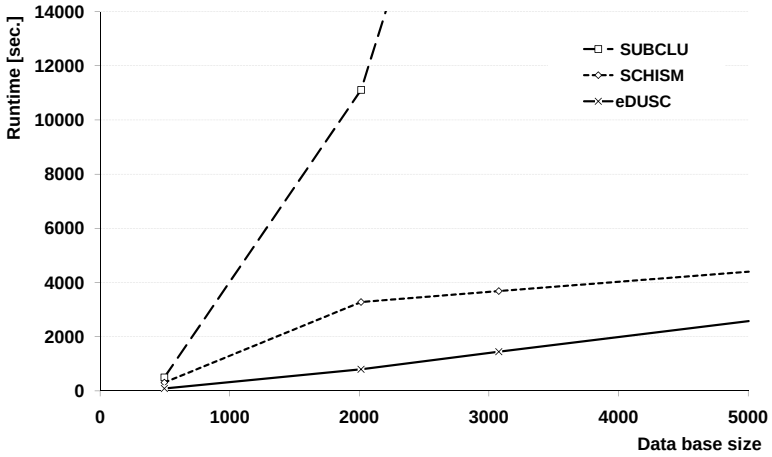

Figure 3.18: Scalability vs. database size

lem. eDUSC uses an efficient depth-first strategy and therefore scales remarkably well. Further on, in contrast to SCHISM eDUSC is lossless because of its novel density conserving grid.

Scalability w.r.t. database size

In the next experiment we generate 15-dimensional data sets with different numbers of objects. As we can see in Figure 3.18, eDUSC and SCHISM scale well w.r.t. the number of objects. However eDUSC again outperforms SCHISM due to the new depth-first approach without candidate generation. SUBCLU also does not scale with increasing number of objects because of the time consuming database scans needed for density-based clustering. In contrast eDUSC uses the density conserving grid to avoid these scans and thus is nearly not influenced by the database size.

Selectivity

To illustrate the efficiency of the hypercube approximation filter we analyze its selectivity compared to the number of database scans in SUBCLU. Recall that density-based clustering has quadratic complexity w.r.t. the number of objects. Thus avoiding database scans contributes significantly to the performance gains of eDUSC. Figure 3.19 shows the number of required density-based clustering computations for the data from our first experiment with

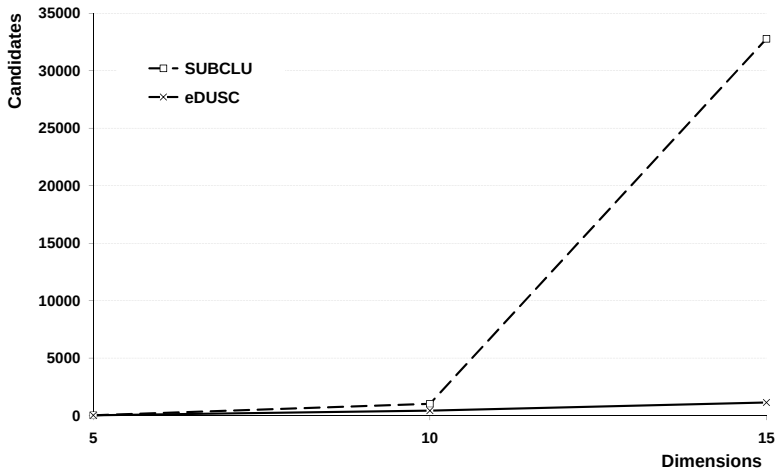

Figure 3.19: Selectivity vs. dimensions

varying dimensionality. The highest dimensionality shown is 15-dimensional because SUBCLU does not scale to higher dimensional data. We see that indeed the main drawback of SUBCLU is the large number of regions that have to be clustered. Multistep filtering in eDUSC substantially lowers the number of density-based clustering computations. Many regions that have to be processed by SUBCLU are pruned by eDSUC. Pruning is possible based on the information of the grid and border cells without any database scans.

Parameters

Detecting clusters in arbitrary subspaces may require tedious parameter tuning in some algorithms. We evaluate parameter setting for SUBCLU, SCHISM and eDUSC to study their parameter sensitivity and demonstrate that the eDUSC algorithm is quite robust in terms of parameter settings.

Fixed density thresholds. As already mentioned SUBCLU is dimensionality biased due to a fixed density threshold $MinPoints$. For different dimensional subspaces the measured density is not comparable. To find high dimensional subspace clusters one has to use a lower value for $MinPoints$ in SUBCLU. In eDUSC this parameter is robust because of the normalized density approach as defined in Section 3.3. Figure 3.20 illustrates that with decreasing $MinPoints$ the runtime of SUBCLU increases. Finding high dimensional subspace clusters is virtually infeasible because already for

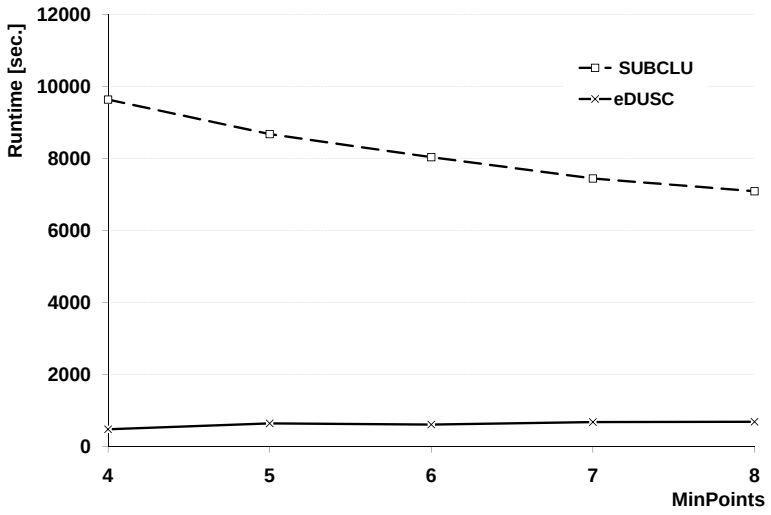

Figure 3.20: MinPoints vs. runtime

$MinPoints = 8$ SUBCLU did not scale in the previous experiments.

Variable density thresholds

As eDUSC uses a variable threshold its density measure is comparable for clusters in different subspaces. In Figure 3.20 we see that the choice of $MinPoints$ has almost no effect on the runtime of the eDUSC algorithm. As we can see in Figure 3.21 the second parameter F has almost constant runtimes, too. Using a normalized density measure which automatically adapts to the dimensionality of the subspace, the variable density threshold is simply set with the fixed parameter F. F reflects the factor by which the variable expected density should be exceeded. Thus eDUSC is easily parameterized by a constant and thus robust factor, yet adapts to the dimensionality of the subspace.

Gridsize

Grid-based algorithms like SCHISM suffer from sensitivity to the grid resolution. In Figure 3.22, runtime of SCHISM and eDUSC for varying gridsizes are shown. The performance of SCHISM depends largely on the grid structure not only in terms of runtime as shown in Figure 3.22, but also in terms of accuracy as density assessment is for individual grid cells (see Chapter

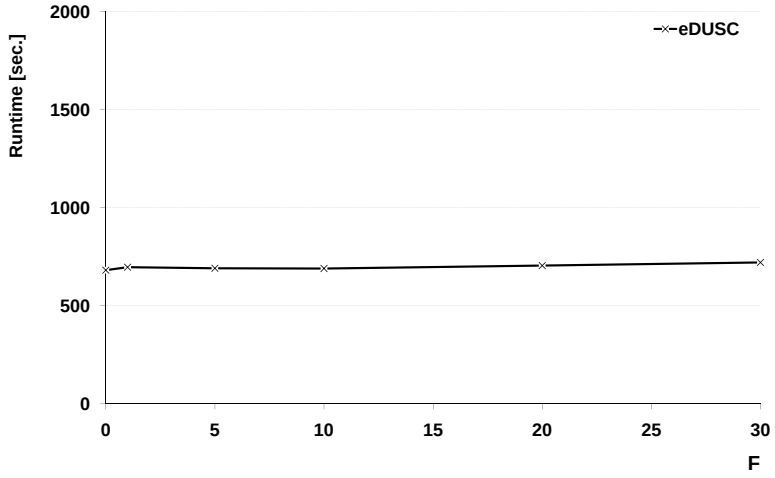

Figure 3.21: F parameter vs. runtime

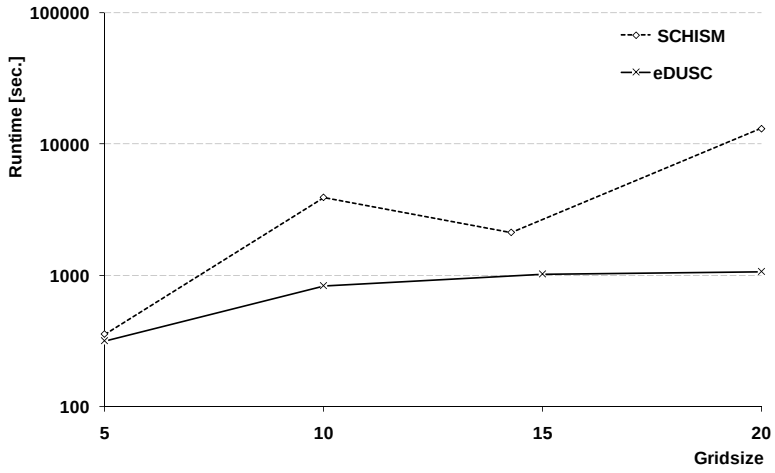

Figure 3.22: Gridsize vs. runtime

2.7). eDUSC is robust to positioning and resolution of the grid through its grid cells merges.

Weighting functions

eDUSC uses a weighted density measure for density assessment in the neighborhood of an object (see Section 3.3). We show that its runtime is insensitive to the choice of kernels for weighting. Figure 3.23 illustrates the runtimes for SUBCLU, SCHISM and eDUSC with the Rectangle (REC)

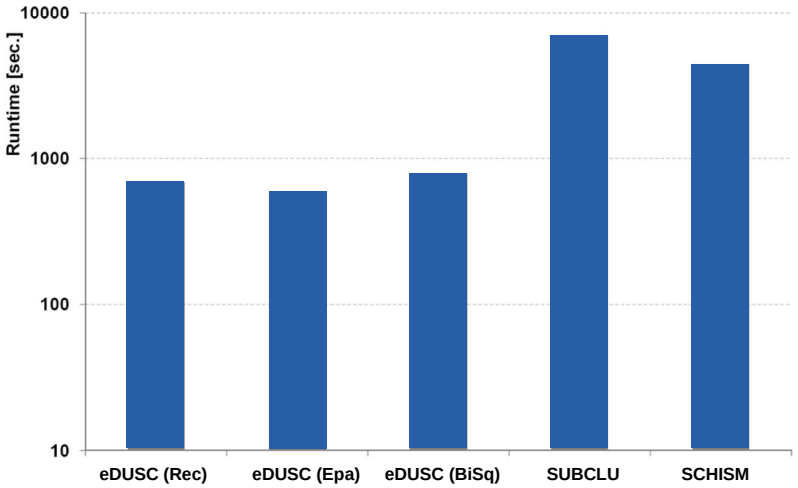

Figure 3.23: Three kernels vs. SUBCLU and SCHISM

$W(x) = 1$, Epanechnikov (EPA) $W(x) = 1 - x^2$ and Bisquare kernels (BIS) $W(x) = (1 - x^2)^2$, respectively.

Sorting heuristics

We next evaluate the effect of different heuristics for constructing the SC-tree. As mentioned, the SC-tree construction mainly depends on the order of the dimensions. Two possible heuristics have been proposed in the last section: creating compact trees using an ascending order of the dimensions w.r.t. their entropy or avoiding early merges by sorting the dimensions in descending order w.r.t. their entropy. We first evaluate the effect of the different sorting heuristics on the runtime (see Figure 3.24). Clearly, avoiding early merges improves the overall runtime of $eDSUC$. Sorting in descending order almost halves the runtime.

On the other hand, by sorting in ascending order the size of the initial SC-tree is reduced as presented in Figure 3.25. This shows that even with a smaller initial SC-tree the mining algorithm cannot reach the good runtimes of the descending order heuristic. For efficient mining it is thus essential to have as few merges as possible in the first restrictions. This is achieved by the descending order heuristic.

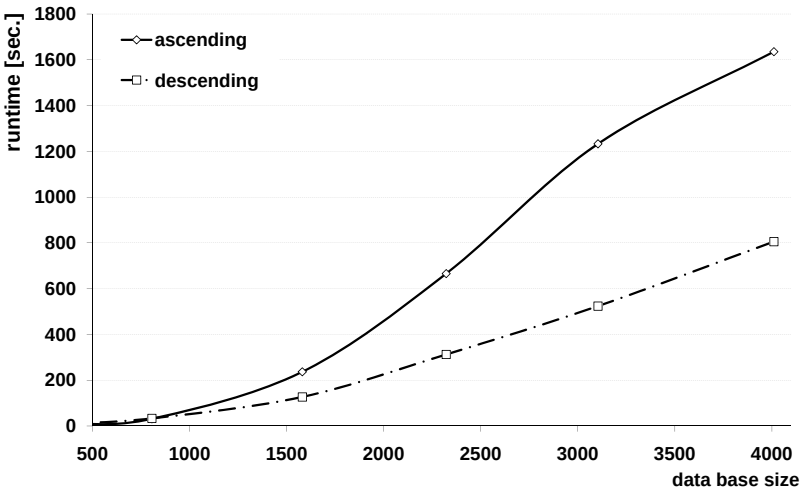

Figure 3.24: Runtimes for sorting heuristic

Redundancy removal

We now demonstrate the effects of in-process-removal of redundancy. Figure 3.26 illustrates the effect of varying the degree of redundancy δ for subspace clusters (see our subspace cluster Definition 3.3) for different database sizes. The higher δ, the more redundant subspace clusters are included in the result set. As we can see, performance depends largely on the degree of redundancy. No redundancy, i.e. $\delta = 0\%$ shows runtimes far lower than all other variants. Even as little as $\delta = 20\%$ redundancy in the result set leads to four times slower runtimes (depending on the database size). Including all redundant subspace clusters, i.e. $\delta = 100\%$ shows exponential runtime behavior that is only slightly better than SUBCLU. Thus, redundancy removal is clearly important for runtime performance. As shown in the next experiment, redundancy removal is also highly beneficial for quality of the result.

The $F1$-value is an improved measure in evaluating the quality of a clustering if the hidden clusters are known (see also Chapter 2.7). For each hidden cluster precision and recall are evaluated. Clusters hidden in high dimensional real world data sets typically spread over several subspaces. Thus, for each hidden cluster H_i ($i = 1...h$), the found clusters F_i are determined by joining all subspace clusters C_k detected that mainly represent H_i. Precision measures the accuracy of detection of a hidden cluster, while recall measures the coverage of the detection. The $F1$-value for one hidden cluster is then defined by the harmonic mean of precision and recall. Finally the $F1$ value

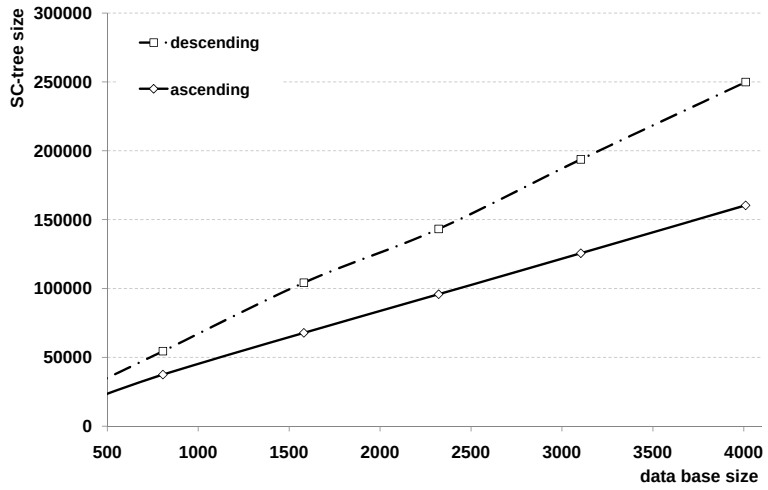

Figure 3.25: Tree size for sorting heuristic

for the complete clustering is determined by averaging all $F1$-values for all hidden clusters [MSE06].

$$
\begin{aligned}
F_i &:= \{o \mid \exists k : o \in C_k \wedge i = \text{argmax}_j |C_k \cap H_j|\} \\
Precision &:= \frac{F_i \cap H_i}{F_i} \\
Recall &:= \frac{F_i \cap H_i}{H_i} \\
F1 &:= \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}
\end{aligned}
$$

Figure 3.27 illustrates $F1$ quality measurements and the size of the output for the same experiment as before. As we can see from the dark gray bars, the number of subspace clusters increases exponentially (note that the scale to the right is logarithmic). Thus, users would be overwhelmed by the output size of traditional algorithms. The light gray bars indicate the $F1$ quality measurements. Clearly, removing redundancy leads to far better $F1$ values. Redundant subspace clusters thus obscure the information in maximal dimensionality subspace clusters. Thus, removing redundancy yields better efficiency and effectiveness in subspace clustering. If no redundancy

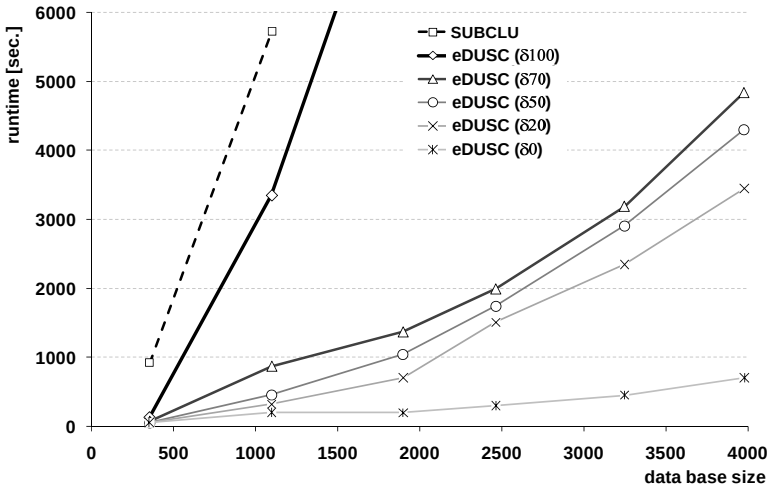

Figure 3.26: Redundancy

is removed like in SUBCLU the number of subspace clusters is tremendous. The SUBCLU algorithm did not finish after more than 10 days. Hence we were not able to evaluate the $F1$ value for SUBCLU.

3.6.2 Quality and runtimes on real world data

In Chapter 2.7 we already analyzed the quality of our DUSC subspace clustering method. In this experiment we evaluate the effects of removing redundancy on the runtime of $eDSUC$ w.r.t. the quality of the result. Since $eDSUC$ is capable of removing redundant cluster during the mining process removing redundancy greatly improves the runtime. As in the previous experiment in Chapter 2.7 we use four real world data sets (see Table 3.1). To measure $F1$ values on real world data where no ground truth on the number and size of subspace clusters is known, we use class label information in the data as ground truth.

For each data set $F1$ quality measurements are given in Figure 3.28 for SUBCLU and for $eDSUC$ with three different redundancy settings, $\delta = 0\%, 2\%, 5\%$. The bars nicely illustrate that removing redundancy is important for all four real world data sets. This result is similar to the results presented in Chapter 2.7. Hence, the redundancy definition given in Definition 3.3 indeed removes those clusters which are blurred by noise. If clusters which contain noise are removed from the result set, the precision is improved. For

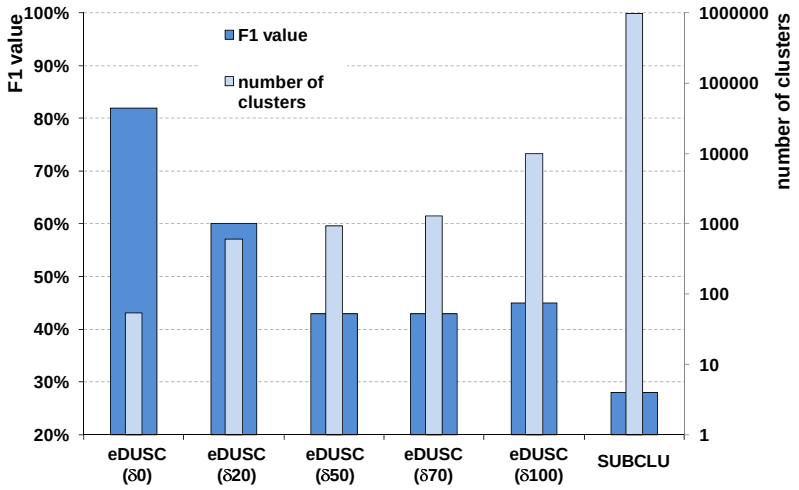

Figure 3.27: Redundancy Quality

	Objects	Dimensions	Classes	Source
Pendigits	7494	16	10	[AN07]
Vowel	990	10	11	[AN07]
Glass	214	9	6	[AN07]
Shapes	160	17	9	[KWX+06]

Table 3.1: Real world data sets

example using $\delta = 0\%$ results in a precision of 30% for the Vowel data while using $\delta = 5\%$ yields a precision of 24%. Removing all redundant subspace clusters might miss a few cluster objects resulting in slightly lower recall. As illustrated by Figure 3.28 allowing a very small amount of redundant clusters is a good compromise between a high precision and recall. Consequently, removing redundancy from real world subspace clusterings is an important step not only for allowing users to inspect the result but also in improving the quality.

Runtimes for all data sets are given in Figure 3.29. Comparing the runtime and quality results we can see that the enormous efficiency gain of the *eDUSC* algorithm is achieved for an even more effective density-based model that outperforms competing approaches. As discussed, our proposed *eDSUC* algorithm is the first algorithm which is able to speed up subspace clustering

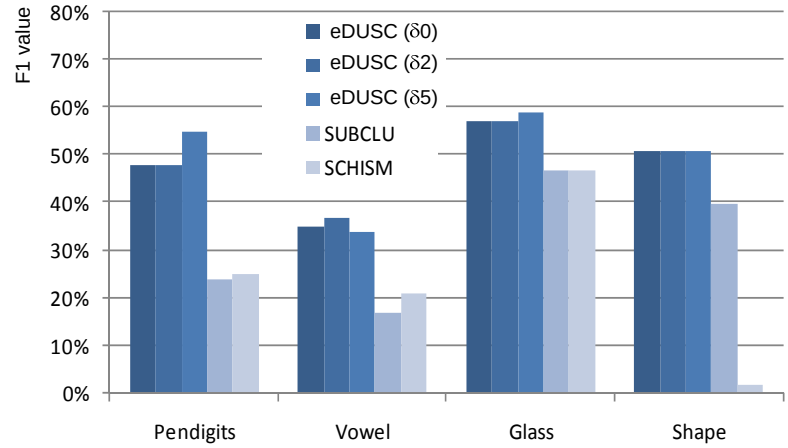

Figure 3.28: Quality on real data

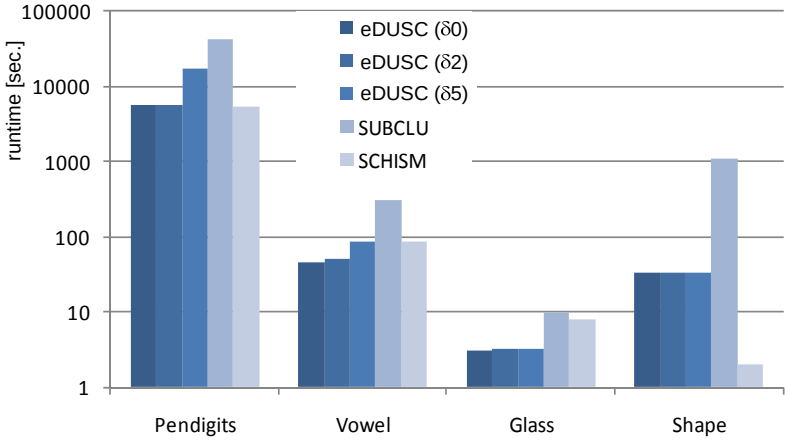

Figure 3.29: Runtime on real data

by in-process-removal of redundant clusters. The results demonstrate that *eDSUC* outperforms SUBCLU, especially for settings which exploit the full potential of redundancy pruning ($\delta = 0\%$). Thus, *eDUSC* is an algorithm that shows significantly better runtimes for a subspace clustering model of very high accuracy, in two datasets even better runtimes than the approximate SCHISM approach.

Summing up, *eDSUC* indeed outperforms existing subspace clustering approaches on both synthetic and real world data. The filter-and-refinement

architecture allows efficient in-process-removal of redundant subspace clusters. The SC-tree support the *eDSUC* algorithm by indexing subspace regions which yields superior runtime performance. Removing redundancy actually improving the quality of the result and reduces the output to a manageable size.

3.7 Conclusion

We introduced eDUSC, an efficient unbiased density-based subspace clustering algorithm. eDUSC uses a database inspired multistep approach which allows powerful pruning of the search space by exploiting two monotonicity properties. Based on a novel density-conserving grid a conservative approximation of density-connected regions by hypercubes ensures completeness. A systematic processing order of the grid cells yields high efficiency.

Efficient implementation of the filter steps is achieved by using the SC-tree. The SC-tree supports the depth-first approach of eDUSC and stores all information necessary for in-process-pruning of clusters. Efficiency is clearly achieved without jeopardizing accuracy, as eDUSC is capable of losslessly detecting subspace clusters with respect to recent models incorporating normalized density and different kernel weighting functions.

Our experiments on large and high dimensional synthetic and real world data sets show that eDUSC outperforms recent subspace clustering algorithms by orders of magnitude. Further on the depth-first approach of eDUSC makes redundancy pruning and indexing of subspace regions possible. We analyzed how in-process removal of redundancy in a depth-first approach permits powerful pruning of the subspace cluster search space.

Chapter 4

Visualization of subspace clusters

Subspace clustering as described in the last chapters is capable to extract interesting patterns from high dimensional data. To generate knowledge from these patterns and benefit from human cognitive abilities, meaningful visualization of patterns is crucial. Clustering is a data mining technique that aims at grouping data to patterns based on mutual (dis-)similarity. For high dimensional data, subspace clustering searches patterns in any subspace of the attributes as patterns are typically obscured by many irrelevant attributes in the full space. For visual analysis of subspace clusters, their comparability has to be ensured. Existing subspace clustering approaches, however, lack interactive visualization and show bias with respect to the dimensionality of subspaces.

In this chapter, we propose a novel distance function for subspace clusterings. We suggest two visualization techniques that allow users to browse the entire subspace clustering, to zoom into individual objects, and to analyze subspace cluster characteristics in-depth. Bracketing of different parameter settings enable users to immediately see the effect of parameters on their data and hence to choose the best clustering result for further analysis. Usage of user analysis for feedback to the subspace clustering algorithm directly improves the subspace clustering. We demonstrate our visualization techniques on real world data and confirm results through additional accuracy measurements and comparison with existing subspace clustering algorithms.

4.1 Introduction

While computers efficiently support statistical and associative analysis of large amounts of data, they do not reach the high cognitive abilities of human users. Human experts are able to quickly identify both correlations and irregularities if data or results are appropriately visualized [Kei02]. As "visual animals", humans excel at expressing and analyzing visual entities [Luc94]. Tasks that are inherently difficult for computers such as parametrization, or are subjective by their very nature such as redundancy or relevance to a given task, can be effectively supported by human computer interaction. Visualization plays a key role in interaction as it builds an interface between an automated output and the human user. Visual data analysis allows combining the efficiency of computers with the cognitive strength of human users [Kei02]. For data analysis or mining tasks, visualization is the central step from patterns to knowledge in the knowledge discovery process [HK01].

Subspace clustering, as introduced in the last chapters, aims at identifying clusters in any possible attribute combination. So far we solved the efficiency problems of subspace clustering algorithms and removed redundant clusters to obtain reasonable result sets. As the identified patterns of subspace clustering algorithms are represented in different subspaces visualizing subspace clusterings is a challenging problem, as well.

For full-space clustering, different visualization techniques exist [FGW02, SD02, FdOL03, KS04, KMSZ06]. In all these approaches the same fixed set of dimensions is encoded in the visual model. These methods are thus as limited as full-space clustering: clusters are detected only if visible in the chosen mappings. As typically clusters are hidden in different subspace dimensions they cannot be detected by globally defined projections and encodings.

Meaningful subspace cluster visualization requires aggregation of similar results and occlusion of redundant information. Users need compact representation of different subspace clusterings for visual analysis of relevant aspects and feedback to the algorithm. Visualization thus requires a measure of similarity for the output, as well as techniques to reduce the overwhelming number of (redundant) results common in subspace clustering. Exploiting user feedback, subspace clustering algorithms may be focused to relevant parts or subspaces of the data through explicit guidance.

In this chapter, we propose a novel distance measure for subspace clusters that reflects their inherent connections or differences, respectively. Based on

subspace clustering distance models, VISA (visual subspace clustering analysis) is able to produce expressive visual diagrams that allow for meaningful user interaction.

The key properties of our density-based subspace clustering approach DUSC (dimensionality unbiased subspace clustering) make reasonable browsing of subspace clusterings possible. As DUSC takes the dimensionality of clusterings into account and removes redundant clusters DUSC leads to comparable clustering results between different subspaces. This allows for meaningful analysis of visualized subspace clusters following our novel VISA method.

In this chapter, we propose visualization techniques for subspace clustering. Our contributions include:

- unbiased, comparable results based on a statistically sound subspace clustering model
- powerful yet compact visualization capable of dealing with clusters in many different projections
- meaningful ranking of the most "interesting" results to guide analysis of the output
- user interaction for parameter setting, exploration and feedback
- handling of redundant output

This chapter is structured as follows: we review related work on visualization techniques in the following section. Novel visualization techniques for subspace clustering results, including discussion of how to rate (dis-)similarity and redundancy of the output, are presented in Section 4.3, before we conclude in Chapter 4.4.

4.2 Related work

Traditional clustering does not scale to high dimensional spaces. As clusters do not show across all attributes, they are hidden by irrelevant attributes [BGRS99]. Global dimensionality reduction techniques such as principal component analysis are typically not appropriate, as relevance is not globally uniform [DHS01].

Visual data analysis techniques benefit from human cognitive abilities in data mining through user interaction. For traditional clustering in general, various visualization techniques have been proposed [FGW02, SD02, FdOL03, KS04, KMSZ06]. In all these cases, the same dimensions of the data space are encoded by the same projections or parameters in the visual model. The methods thus behave as limited as full-space clustering does: Clusters are only detected if they are visible in the respective chosen mappings. As typically clusters are hidden in different subspace dimensions they cannot be detected by globally defined projections and encodings.

Additionally, subspace clustering visualization has to deal with the exponential number of subspace projections and the typically enormous redundancy of the result. As different projections contain different clusters, visualization should provide an overall overview as well as means for in-depth analysis and interaction. Special cases like mosaic encodings in gene-expression analysis [ESBB98] order clusters by biological properties like position of a gene on the chromosomes. This underlying ordering does not extend to other application domains. As there is no inherent ordering in general subspace clustering applications, lack of (dis-)similarity measures and poor comparability of results are major hindrances for visualization.

4.3 VISA

In the last sections we analyzed the result of the DUSC subspace clustering model defined in Section 2.5 analytically. To gain knowledge from the resulting patterns the clusters have to be analyzed by users. This analysis is still challenging as there are many different subspace projections and possibly redundant subspace clusters. We define the set of subspace clusters that have to be analyzed:

Definition 4.1 ***Subspace Clustering***

A subspace clustering is a set of clusters in their corresponding subspaces $\{(\mathbf{C}_1, \mathbf{S}_1), \ldots, (\mathbf{C}_n, \mathbf{S}_n)\}$*, where* $\mathbf{C}_i$ *is a subspace cluster in subspace* $\mathbf{S}_i$ *as specified in Definition 2.5.*

For user benefit, it is necessary to visualize subspace clusterings such that the entire output can be browsed even for clusters in different subspace

projections, and that detailed views into individual subspace clusters are possible. Moreover, feedback for subsequent guiding of subspace clustering runs should be provided. This requires visual support for analysis of parameter settings as well as a focus on the most "interesting" results.

We therefore define a set of criteria that should be included in visualization:

Definition 4.2 ***Visual analysis criteria***

The following properties characterize a subspace clustering:

- ***space overlap****, i.e. the number of common subspaces for any two cluster subspaces* $\mathbf{S}_i$ *and* $\mathbf{S}_j$*:* $\mathcal{S} = |\mathbf{S}_i \cap \mathbf{S}_j|$
- ***object overlap****, i.e. the number of common objects in any two subspace clusters* $\mathbf{C}_i$ *and* $\mathbf{C}_j$*:* $\mathcal{O} = |\mathbf{C}_i \cap \mathbf{C}_j|$
- ***interestingness****, i.e. the factor* $\mathcal{I}$ *by which the expected density is exceeded on average for any subspace cluster* $\mathbf{C}_i$ *in subspace* $\mathbf{S}_i$*:*

$$\mathcal{I}(\mathbf{C}_i, \mathbf{S}_i) = \frac{\sum_{o \in \mathbf{C}_i} \varphi^{\mathbf{S_i}}(o)}{|\mathbf{C}_i| \cdot \alpha(\mathbf{S}_i)}$$

The above aspects are crucial for our VISA approach, since they are necessary for in-depth analysis of subspace clusterings, where large result sizes easily occlude the most interesting, novel patterns. Next, we show how compact representation and detail information on subspace clusters can be combined for browsing. After this, we focus on bracketing and in-depth analysis.

4.3.1 Browsing subspace clusterings

To give a general overview over the entire subspace clustering, basic structural information of subspace clusters should be compactly represented. Interactively, details should be provided for individual clusters during browsing.

The major challenge for any overview visualization of subspace clusterings lies in the comparability of subspace clusters. As subspace clustering algorithms may identify patterns in completely different or overlapping subspaces, we have to define (dis-)similarity of subspace clusters on a general

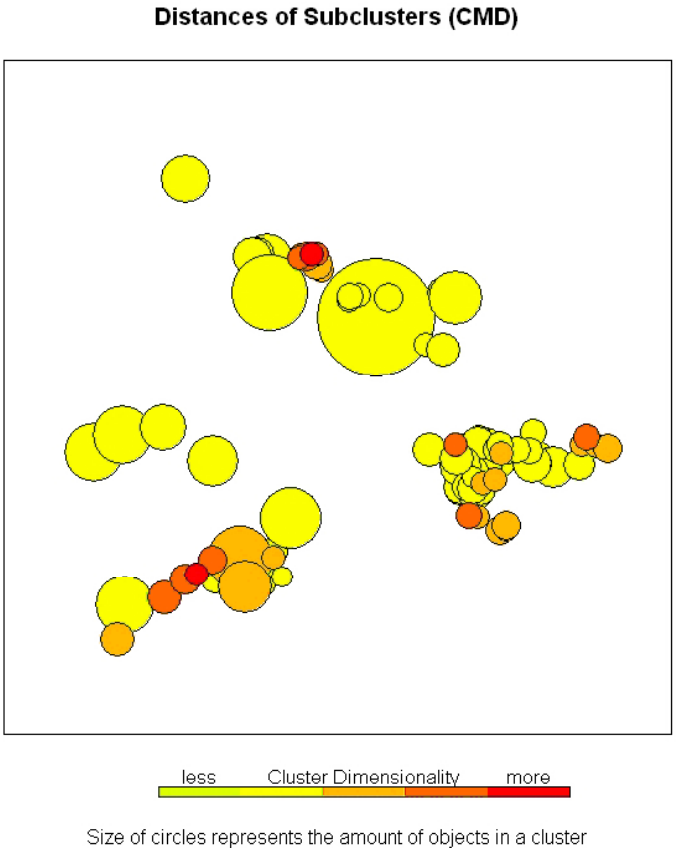

Figure 4.1: Subspace clustering overview: the diameter depicts the number of objects in the cluster, the color denotes the dimensionality (the darker, the more dimensions)

scale. We propose a distance function for comparing any two subspace clusters that takes both the subspace overlap and the object overlap into account. It is defined as a convex sum of the difference in subspaces and the difference in cluster objects:

Definition 4.3 ***Subspace Cluster Distance***

The distance between two subspace clusters $\mathbf{C}_i, \mathbf{C}_j$ *in subspaces* $\mathbf{S}_i, \mathbf{S}_j$*, respectively, is defined as the convex sum of subspace distance and object distance:*

$$\beta \left(1 - \frac{|\mathbf{S}_i \cap \mathbf{S}_j|}{|\mathbf{S}_i \cup \mathbf{S}_j|}\right) + (1-\beta)\left(1 - \frac{|\mathbf{C}_i \cap \mathbf{C}_j|}{\min\{|\mathbf{C}_i|, |\mathbf{C}_j|\}}\right)$$

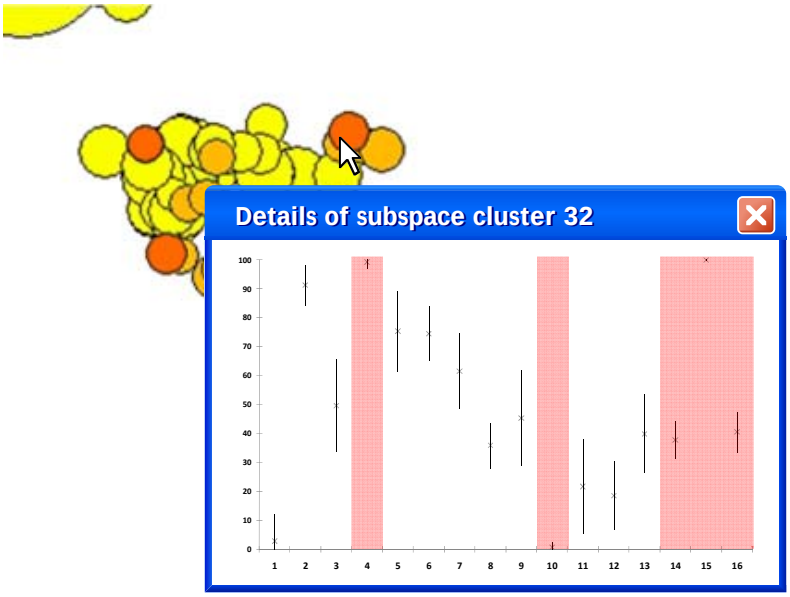

Figure 4.2: Detailed view for one subspace cluster: mean and variance for each dimension of the subspace cluster

This distance function thus allows comparing subspace clusters for a general overview. Independent of any application specific properties of the data, similarity can be measured by incorporating the criteria presented in Definition 4.2.

Normalization of both subspace and object distance is to a range of zero to one, respectively. We use two different normalizations for object and subspace distances.

1.) Object distance: a fully redundant subspace cluster is a lower dimensional projection of an interesting subspace cluster (object overlap). For visualization, interesting subspace clusters should be stacked upon their redundant counterparts, i.e. the distance should be zero. This is achieved by normalizing object distance by minimum cluster size. For redundant subspace clusters, the minimum in the denominator is the same as the intersection in the numerator, and object distance is indeed zero.

2.) For subspace distances, clusters in subsets of dimensions may very well differ in all objects. They share subspace projections, yet do not actually overlap. Their subspace distance should therefore still reflect the difference in the remaining dimensions, and not be zero. To this end, normalization is by the union of subspaces. For visualization, this means that sub-projections

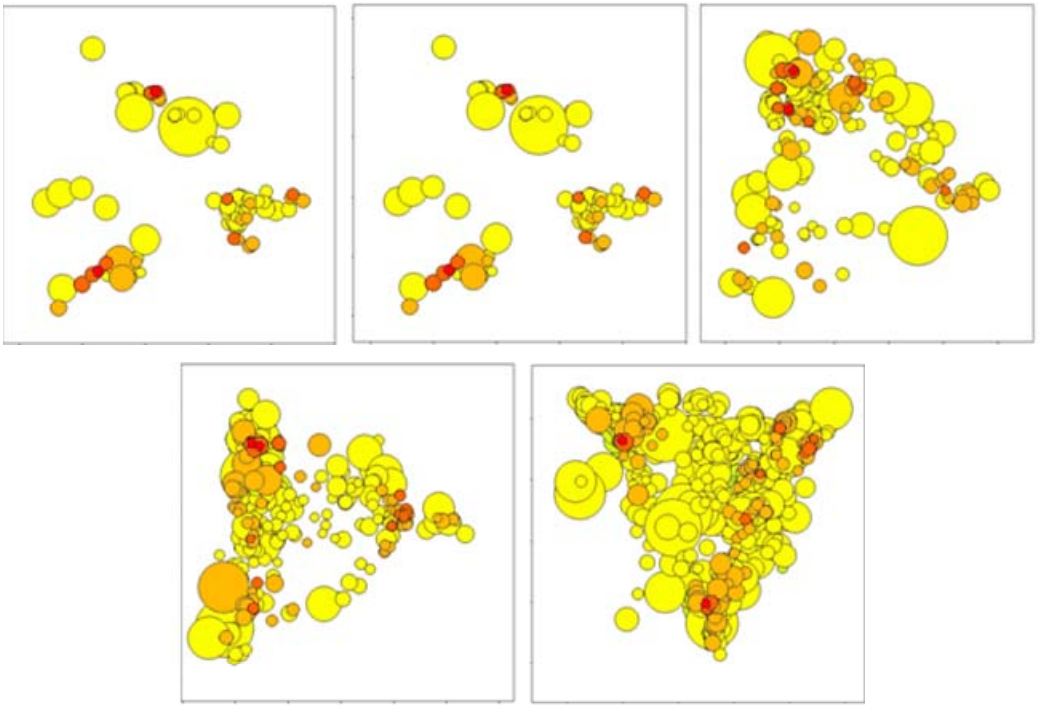

Figure 4.3: Bracketing redundancy in MDS images (from left to right $\delta = 0\%, 2\%, 5\%, 7\%, 10\%$): for each parameter setting, the output is illustrated, allowing users to pick appropriate settings from the series of images easily

are located close to each other.

Object overlap can be further favored by choosing β smaller than 0.5 to give more weight to object distance.

An overview over the distances of all subspace clusters can be achieved by multidimensional scaling (MDS) [Kru64]. Simply put, it allows for non-linear projection of objects from their original distance space to a 2D or 3D visualization space, preserving mutual distances as much as possible. In our MDS image of the subspace clustering result, the distance information is enriched by information on the size and the dimensionality of a subspace clusters. For a general overview size and dimensionality of subspace clusters can be visualized by the radius of the circles and their color, respectively (see Figure 4.1). Consequently, browsing the entire output is possible. However, this enriched MDS image by itself does not provide sufficient information for user analysis. To get a better understanding of why subspace clusters are similar or different and for browsing the actual objects in subspace clusters,

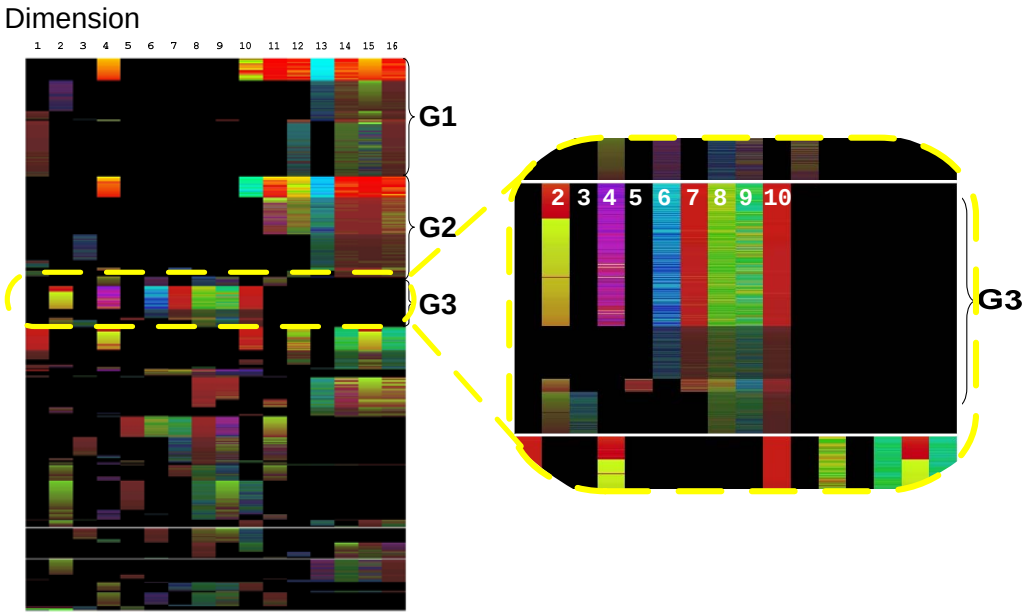

Figure 4.4: Matrix of subspace clusters groups. Rows represent the cluster and columns the dimensions, groups are separated by white lines (see right part). Color map HSV: hues represent the value in each dimension of the subspace cluster; saturation and value represent the factor $\mathcal{I}$ of interestingness.

detailed information should be available upon click on subspace cluster representations. As individual objects in subspace clusters are typically more than two-dimensional, we represent the distribution of objects in a subspace cluster by a mean and variance plot for all dimensions. Additionally, we highlight those dimensions that are relevant for the subspace cluster (see Figure 4.2).

4.3.2 Bracketing

Bracketing refers to a technique originally from photography. Several different camera settings are used to take a series of pictures of the same subject. Photographers then pick the best setting among the resulting pictures. It has been discussed in human computer interaction in [Rob04]. We propose bracketing for subspace clustering visualization as a useful technique that demonstrates the effects of any parameterization. It provides not just a sin-

gle subspace clustering output, but a series to choose from. This allows users to analyze the effect of different parameters at a single view.

As we have seen already in Figure 4.1, the clustering shows groups of subspace clusters. In their center we have the desired high dimensional (red) subspace clusters containing few objects (small circle). These high dimensional clusters in lower projections again form clusters, which are depicted as large yellow circles. This effect can be reduced by our redundancy parameter δ, as we can see in Figure 4.3. The overwhelming result for $\delta = 10\%$, i.e. redundancy of 10% is permitted, gives us evidence of poor cluster quality as in large low dimensional clusters not only hidden clusters are found but also noise. As we discussed in Chapter 2.5 removing such redundant clusters leads to a significant quality improvement of the overall clustering result.

4.3.3 In-depth analysis

As illustrated in Section 4.3.1, subspace clusterings visualized in MDS images typically form groups of similar subspace clusters. For in-depth analysis, visualizing characteristics of each of these groups of subspace clusters in a compact way is important. Characteristics are joint and distinguishing properties of groups of subspace clusters. Properties of importance are the interestingness $\mathcal{I}$ of a subspace cluster as well as the subspace and object overlap. Visualizing these properties is necessary for in-depth analysis of the overall subspace clustering.

Subspace clusters are grouped if they share dimensions or if they have objects in common, as defined by our subspace cluster distance function. To visualize these groupings such that similarities show up as clear visual patterns, we propose to plot all groups of subspace clusters and the objects contained in each group. Each group is defined as the set of all subspace clusters similar to a subspace cluster of high interest (see Def. 4.2). We call the most interesting subspace cluster of a group the *anchor* of the group. A *group* is then defined as all subspace clusters of at most ϑ distance to the anchor.

To visualize a group of subspace clusters we use a matrix representation. We start with the subspace cluster having the highest interest. Hence this cluster is the anchor of the first group. Starting with the anchor each subspace cluster of a group is represented by its objects. Each object is depicted by one row whereas the columns illustrate the dimensions of the object. To

allow an in-depth analysis we use different color codes to visualize all characteristics of an object, that is its interestingness as well as the values in each dimension. If a dimension is not part of a cluster the column is black.

To highlight subspace clusters of high interestingness we use the factor $\mathcal{I}$ as saturation and value in HSV color space [FvDFH96]. Hence interesting subspace clusters can be easily identified as the active dimensions are represented by bright and intensive colors. To represent the value for each

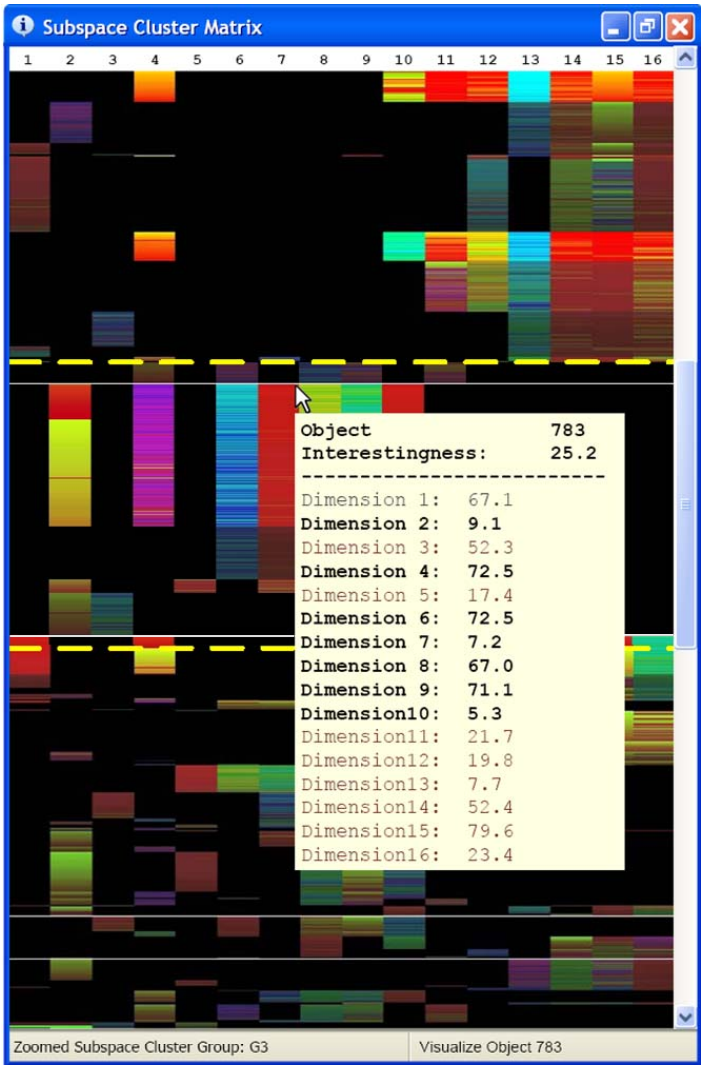

Figure 4.5: User Interface for a Subspace Clusters Matrix: By pointing on individual objects additional information about the cluster is visualized

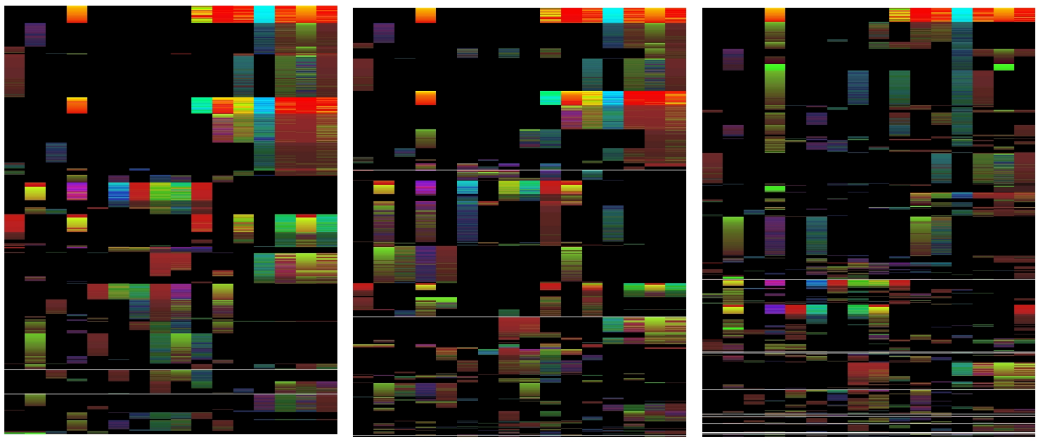

Figure 4.6: Bracketing Matrix for $\delta = 0\%, 5\%, 10\%$: for each parameter setting, the output is illustrated, allowing users to pick appropriate settings from the series of images easily

dimension we use the hue value according to the HSV space. By using all possible hue values the complete color range is used to encode the value of an object but other mappings can be used as well. Further on, objects are ordered with descending interest. Redundant objects that are already visualized in other clusters are omitted in order to obtain a compact and descriptive representation of the complete clustering. Once all objects of a subspace cluster have been presented the next subspace cluster with respect to the next anchor is displayed.

Figure 4.4 illustrates the subspace cluster groups for the Pendigits data set using $\delta = 0\%$. As discussed, a group of subspace clusters typically has some dimensions in common (the *core dimension* of a group). Our proposed subspace clustering matrix allows for visual recognition of core dimensions. Consider the group $G3$ zoomed in on the right side of Figure 4.4, dimensions 2, 4 and 6 to 10 can be easily identified as core dimensions. By comparing the different groups illustrated in Figure 4.4 we can see that many subspace clusters identified correlations in dimension 12 to 16 (e.g. $G1$, $G2$, etc.), while some other groups also have a core containing the dimensions 6 to 10 (e.g. $G3$). Further on, a density-connected fullspace cluster (a cluster considering all dimensions) is not found (no completely colored row contained in Figure 4.4). Additionally, detailed information about an object can be obtained by moving the mouse over the object. Figure 4.5 illustrates a user interface for

visualizing a subspace cluster matrix. Information about the interestingness and the values of an object can be presented in a tool tip. Hence, details about the *center of a group* (i.e. the objects of highest interest) can be easily obtained by the user by moving the mouse over the first objects of a group. For precise selection of individual objects the scroll bar can be used for an interactive zoom of specific areas of the subspace clustering matrix (illustrated in Figure 4.5 by the area surrounded by the yellow lines). The area above and below the yellow lines shows the original compact representation as depicted in Figure 4.4. In-between the lines, rows are enlarged by a specified factor. Scrolling up or down, the zoom area can be varied.

Bracketing as shown in Figure 4.6 illustrates the subspace cluster groups for different redundancy setting (left part $\delta = 0\%$, middle $\delta = 5\%$ and right part $\delta = 10\%$). As can be easily seen, a clear clustering structure is obtained if redundancy is removed (leftmost matrix). Small subspace groups are removed if less redundancy is allowed. Hence, removing redundancy improves the clustering structure. The visual result corresponds to our empirical result presented in the last chapter.

4.4 Conclusion

We introduced the first subspace clustering visualization to the best of our knowledge. Based on comparable results, i.e. unbiased subspace clustering, browsing of the result is possible through a novel distance function that reflects the subspace and the object overlap, respectively. Subspace clustering interestingness is incorporated to show the most relevant results. Interaction is possible through zooming in to objects in subspace clusters and through choice of adequate subspace clustering settings and feedback.

Chapter 5

Efficient subspace clustering for multidimensional sequence databases

Many environmental, scientific, technical or medical database applications require effective and efficient mining of time series, sequences or trajectories of measurements taken at different time points and positions forming large temporal or spatial databases. Particularly the analysis of concurrent and multidimensional time series poses new challenges in finding clusters of arbitrary length and varying number of attributes.

We present a novel algorithm capable of finding subsequence clusters in different subspaces and demonstrate our results for temporal and spatial applications. Our analysis of structural quality parameters in rivers is successfully used by hydrologists to develop measures for river quality improvements.

5.1 Introduction

Environmental sensors produce data streams at successive time points which are often archived for further analysis. Applications like stock market analysis or weather stations gather ordered sequences of different values in large temporal databases. Weather stations for example use multiple sensors to measure e.g. barometric pressure, temperature, humidity, rainfall. Many other scientific research fields like observatories and seismographic stations

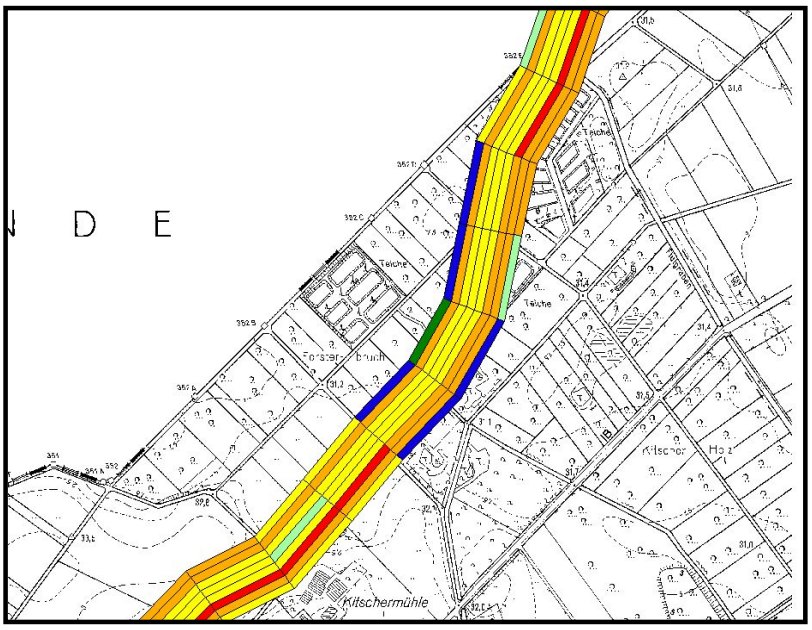

Figure 5.1: GIS illustration of eight out of 19 river attributes. Each attribute is depicted as one of the parallel river sequences. Colors denote the categorical values in each attribute in the respective subsequence, from blue for "one" (best quality) to red for "seven" (worst quality).

archive similar spatial or spatial-temporal sequences.

As one application example, we focus on hydrological data. In a current project of the European Union on renaturation of rivers, the structural quality of river segments is analyzed. For a spatial database of German rivers, about 120.000 one-hundred-meter segments were evaluated according to 19 different structural criteria, e.g. quality of the riverbed [LUA03]. They were mapped to quality categories, where a value of "one" indicates perfect quality, while a value of "seven" indicates most severe damages. Figure 5.1 illustrates 8 of the 19 attributes of a sample river segment in GIS (geographic information system) representation. The sequence order of the segments is given by the flowing direction of the rivers.

As the project aims at a quality improvement over the next decades, packages of measures have been suggested for different structural damages. They have been formalized in rules specifying the attribute value constraints and the attributes influenced positively by execution of the respective measure.

An example constraint might be that a certain segment has good quality (categories one to three) riverbed and riverbanks and poor river bending (categories five to seven). This could be improved by measures like adding deadwood to positively influence river bending.

Finding and analyzing these patterns helps hydrologists summarize the overall state of rivers, give compact representations of typical situations and review the extent to which these situations are covered by measures envisioned. They can identify those patterns which are problematic, i.e. have low quality ratings, but are not yet covered by measures. In a follow-up step, these measures are annotated by time and cost information. This is used to generate an overview over the state of rivers as it might be in the near future if the measures are put into action.

From a computer science point of view, finding the intrinsic structure of these multidimensional sequences is a two-fold task:

- detect frequent patterns within sequences for all possible subsequence lengths (note that we cannot know the pattern length a priori),
- then detect subspaces where patterns occur in combination.

Patterns are ranges of values (which correspond to several categories of river quality structure) found in several (sub-)sequences. Pattern analysis has to take into account that the data is subjective and fuzzy, because structural quality of rivers was mapped by different individuals. Our approach extends density-based clustering based on weighting functions to subsequences. This approach effectively detects fuzzy subsequence patterns. These patterns are clustered efficiently for arbitrary lengths using monotonicity properties. We transform these subsequence pattern clusters into a cluster position space such that mining parallel patterns in any subspace can be reduced to efficient FP-tree frequent itemset mining.

This chapter is organized as follows: we review related work on sequence mining in Section 5.2. Basic definitions in Section 5.3 lay a sound foundation for the proposed clustering method in Section 5.4. We describe and discuss our algorithmic concept in Section 5.5. The experimental evaluation in Section 5.6 demonstrates the effectiveness of our approach on real world and synthetic datasets. Efficiency is shown and parametrization is evaluated. We conclude this chapter in Section 5.7, summarizing our results and anticipating future work.

5.2 Related work

As discussed in Chapter 1.3 numerous clustering methods have been proposed in the literature, including partitioning clustering, e.g. the well-known k-means algorithm [Mac67]. These algorithms require the specification of the number of clusters to be found and can only detect convex cluster regions. Categorical clustering methods work well for categorical data where the notion of neighborhood is not meaningful [GRS99, ZPAS05].

We focus on density-based clustering as density-based methods are robust to noise since it clusters only those points or sequences above some noise threshold as discussed in [Den04, AKMS07b]. Moreover, it naturally incorporates neighboring objects into its cluster definition.

The analysis of sequence data has recently gained a lot of attention. Recordings of data at successive time points have been studied in time series mining; e.g. [FRM94, KCMP01]. Most of these approaches, however, aim at finding patterns of values which do not have to directly follow one another, but may have other values in-between, as in sequential frequent itemset mining [AS95, AFGY02].

Motif mining searches those patterns which have the highest count of similar subsequences [PKLL02]. Matching within some range is used to determine the frequency. However, neighbors are not weighted and the range is fixed for all sequences. Moreover, parallel patterns are not discussed since the application targeted is one-dimensional time series. While noise is removed in motif discovery as well, we found density-based clustering to be more useful in handling fuzzy data, because density-based clusters automatically adapt to different ranges of fuzziness.

5.3 Subspace subsequence cluster model

In this section we formalize our notion of patterns in subsequences. We define a suitable cluster notion which reflects that our database consists of ordinal-valued sequences with some degree of noise. Since we are dealing with clusters of sequences in this chapter we do not use the definitions for objects and clusters given in Section 1.4.

Density-based clustering, which tries to separate dense areas ("clusters") from sparse ones ("noise") reflects the requirements of applications such as

the river data scenario in that arbitrary cluster shapes may be found, the number of clusters does not need to be fixed a-priori and noise is labeled as such, i.e. it does not have to be assigned to any cluster [EKSX96, KKK04].

5.3.1 Subsequence patterns and clusters in one attribute

As noted before, it is crucial to determine subsequence clusters of arbitrary sequence lengths. We define sequence patterns of arbitrary length in single attributes and subsequently model subspace clusters by detecting parallelism between these clusters.

Definition 5.1 ***Sequence pattern:***

- *A tuple* $S = (s_1, \ldots, s_k)$ *of* k *subsequent values at positions* 1 *through* k *is called a* ***sequence*** *of length* k *in one attribute.*
- *A database* $\mathbf{DB}$ *is a set of sequences* $\{S^1, ..., S^z\}$.
- *We denote a* ***subsequence*** *of* S *from position* i *to* j *by* $S[i, j] = (s_i, \ldots, s_j)$.
- *Whenever we are not interested in the concrete positions of a sequence, but merely in its values, we call this a* ***pattern*** $\mathbf{P} = \langle p_1, \ldots, p_k \rangle$ *in one attribute.*
- *A pattern* $\mathbf{P}$ *occurs in a database if there is a sequence* $S \in \mathbf{DB}$ *and a position* $i \in \mathbb{N}$ *with* $\mathbf{P} = S[i, i+k\text{-}1]$.
- *The* ***support*** *of a pattern* $\mathbf{P}$ *is the number of its occurrences in the sequences of the database* $\mathbf{DB}$*:* $sup(\mathbf{P}) = |\{i \in \mathbb{N}$ *and* $S \in \mathbf{DB}$*, where* $\mathbf{P} = S[i, i+k\text{-}1]\}|$

When searching for prevailing patterns, it is important to notice that merely counting of sequence patterns is not sufficient in many scenarios. It is crucial to account for two factors: first, mapping of river structures may be blurred by people's subjective decisions on unclear category boundaries. Second, measures and their constraints may be applicable over several categories and cannot always be fitted exactly to these categorical boundaries.

Moreover, small deviations in few segments may be tolerable for the application of a certain measure if this results in longer river courses treatable by a single measure. Hydrologists are thus interested in including "similar" sequences in frequency notions.

Density-based clustering tries to separate dense areas ("clusters") from sparse ones ("noise"). The density of a pattern is determined by evaluating its neighborhood according to some distance function.

Any L_p-Norm ($L_p(\mathbf{Q}, \mathbf{Q}') = \sqrt[p]{\sum_{i=1}^{k}(q_i - q_i')^p}$) between patterns $\mathbf{Q}$ and $\mathbf{Q}'$ can be used, yet the Manhattan norm (L_1) has shown to work well in preliminary experiments. We use a weighting function to ensure that with greater distance to the pattern evaluated, the influence of neighbors decreases. As discussed in Chapter 1.4 any series of monotonously decreasing values can be used as a weighting function, since distances between nominal sequences are always discrete. All kernel-estimators known from statistics [HK98] are constantly falling functions. Experiments have shown that weighting functions based on Gaussian kernels ($W_\sigma^{\mathbf{P}}(\mathbf{Q}) = \mathsf{exp}\ (-\mathsf{dist}(\mathbf{P}, \mathbf{Q})^2/2\sigma^2)$) perform well for in many categorical applications. Using weighting functions provides us with a natural way of defining the set of similar sequences to be included in the density evaluation. Whenever the weights assigned drop below a certain significance threshold τ, these sequences should not be considered in the density estimation. For example, Gaussian kernels assign (possibly very small) density values to all patterns in the database, which can be cut off below some tiny value, such as $\tau = 0.01$ or less. This way, excess density computations can be avoided.

Definition 5.2 ***Neighborhood and Density****:*

- *The* τ-***neighborhood*** *of a pattern in one attribute* $\mathbf{P}$*,* $N_\tau(\mathbf{P})$*, is defined as the set of all patterns* $\mathbf{Q}$ *which at least have the influence* τ *on the pattern* $\mathbf{P}$ *evaluated:* $N_\tau(\mathbf{P}) = \{\mathbf{Q}, \textit{where } W_\sigma^{\mathbf{P}}(\mathbf{Q}) \geq \tau\}$*.*

- *The* **density** *of* $\mathbf{P}$ *is the weighted sum of patterns in the neighborhood*

$$\mathsf{density}(\mathbf{P}) = \sum_{\mathbf{Q} \in N_\tau(\mathbf{P})} W_\sigma^{\mathbf{P}}(\mathbf{Q}) * sup(\mathbf{Q})$$

- $\mathbf{P}$ *is* ***dense*** *with respect to a density threshold* ρ *iff* $\mathsf{density}(\mathbf{P}) \geq \rho$*.*

We are now ready to formalize our cluster notion. As mentioned before, clusters should consist of similar, dense sequences of the same length. Sequences within one cluster should therefore be within certain parameterizable boundaries.

Definition 5.3 ***Cluster****:*
A set $\mathbf{C} = \{\mathbf{P}_1, \ldots \mathbf{P}_m\}$ *of* m *patterns* $\mathbf{P_i}$ *is a cluster of length* k *with respect to a density threshold* ρ *and a compactness parameter* γ *iff:*

- *for all patterns* $\mathbf{P}_i \in \mathbf{C}$: $\mathsf{density}(\mathbf{P}_i) \geq \rho$. ***(Density)***
- *for any patterns* $\mathbf{P}_i, \mathbf{P}_j \in \mathbf{C}$ *there is a chain of patterns* $(\mathbf{Q}_1, \ldots, \mathbf{Q}_v) \in \mathbf{C}$ *such that* $\mathbf{Q}_1 = P_i, \mathbf{Q}_v = P_j$ *and* $\forall\ r\ \mathsf{dist}(\mathbf{Q}_r, \mathbf{Q}_{r+1}) \leq \gamma$. ***(Compactness)***
- *for all patterns* $\mathbf{Q}$ *of length* k *with* $\ni \mathbf{Q} \ni \mathbf{C}$*:* $\mathbf{C} \cup \{\mathbf{Q}\}$ *is not a cluster.* ***(Maximality)***

Put informally, we are thus looking for clusters of patterns which are as large as possible (maximality), whose elements are all dense (density) and at most γ apart from each other (compactness). We set γ to one in categorical settings.

Example.

Figure 5.2 illustrates the definition of density and gives an example for a density calculation of pattern $\mathbf{P} = \langle 1, 2 \rangle$. The upper part visualizes two sequences S and T with two exemplary attributes, the riverbed and the bank. In the lower part of the figure the density-value for a pattern $\langle 1, 2 \rangle$ of the riverbed attribute is calculated. We assume a significance threshold of $\tau = 0.2$. The Gaussian weighting function drops below $\tau = 0.2$ for sequences having a distance higher than 1.8 from the point evaluated. Thus in our ordinal setting only subsequences with a distances of or less 1 have significant influence: $\mathsf{exp}(-1^2/2) \approx 0.6 > \tau$ and $\mathsf{exp}(-2^2/2) \approx 0.13 < \tau$. In our example the τ-neighborhood of pattern $\langle 1, 2 \rangle$ contains the patterns $\langle 1, 2 \rangle$ itself starting at two positions S_1 and S_6 (distance zero) and $\langle 1, 3 \rangle$ starting at four positions S_3, S_8 and T_2, T_5 (distance one). Thus the density-value for $\langle 1, 2 \rangle$ is $2 * W_\sigma^{\langle 1,2 \rangle}(\langle 1, 2 \rangle) + 4 * W_\sigma^{\langle 1,2 \rangle}(\langle 1, 3 \rangle) = 2 * 1 + 4 * 0.6 = 4.4$. For a density-threshold of e.g. $\rho = 3$ the pattern $\langle 1, 2 \rangle$ is considered dense.

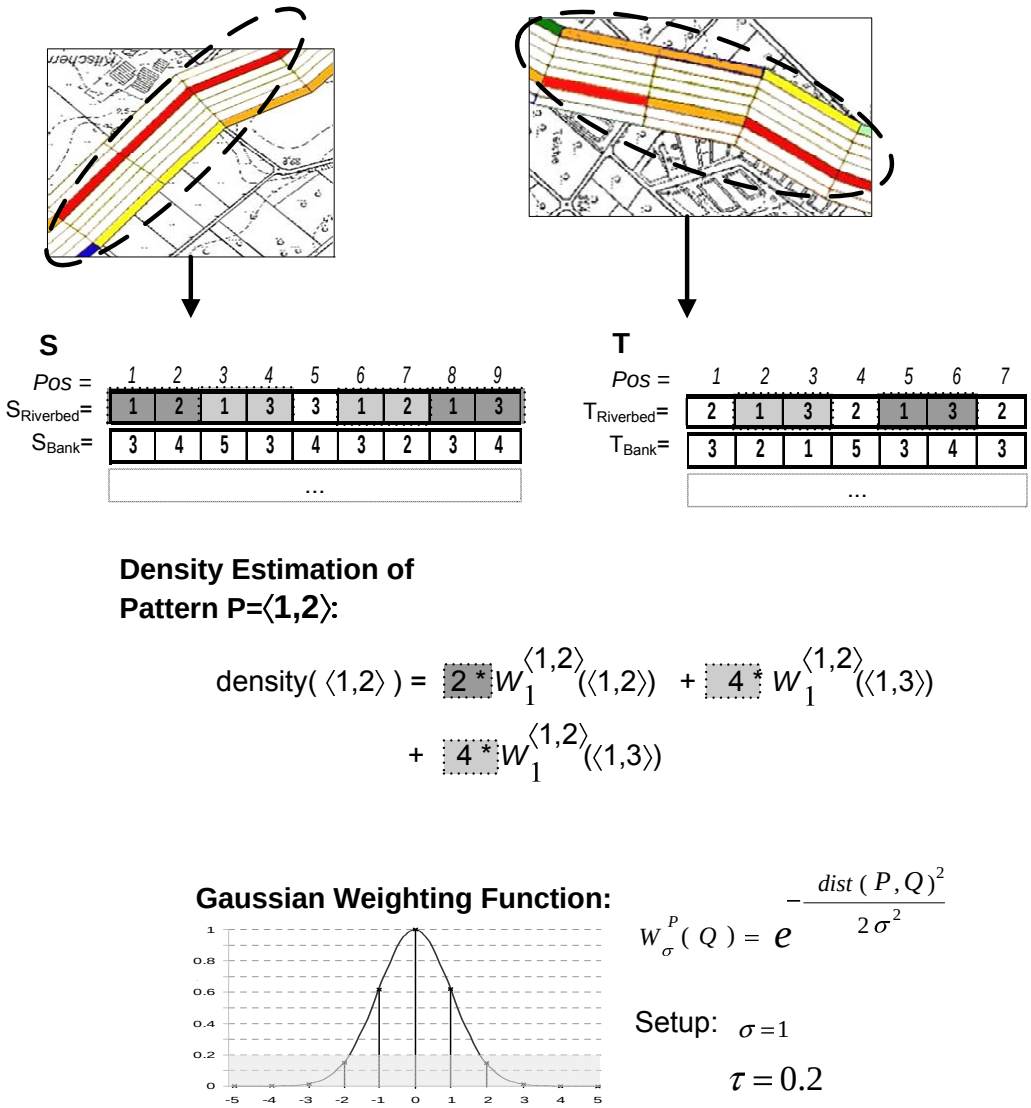

Figure 5.2: Example density calculation for a pattern contained in the river database

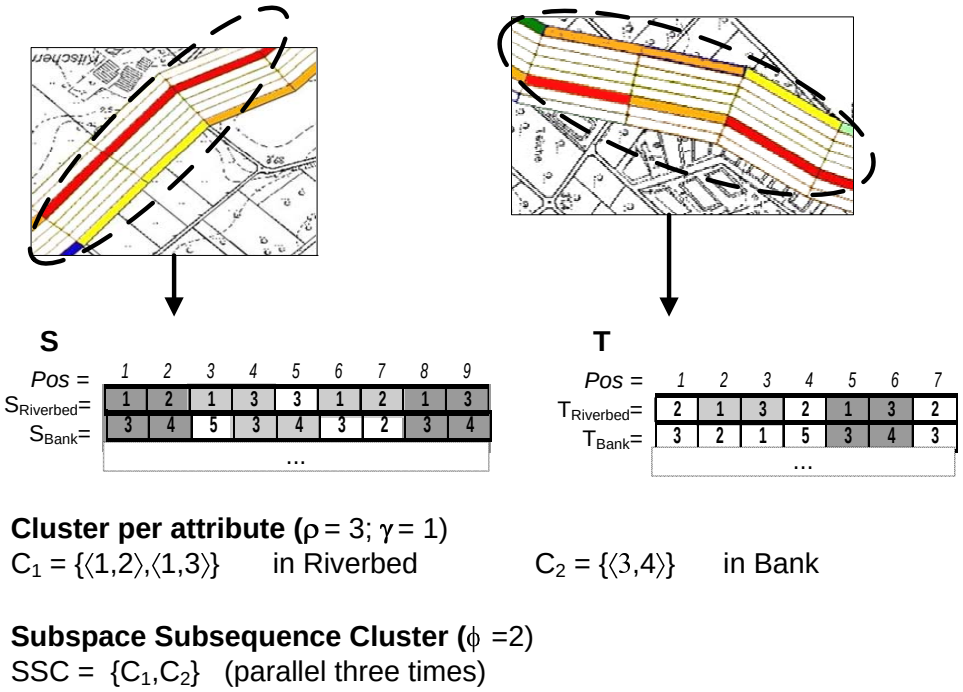

Figure 5.3: Example for a subspace subsequence cluster

5.3.2 Subspace subsequence clusters

Single sequence patterns give insight into the inherent structure in individual attributes. In multidimensional sequence databases these clusters of patterns have to be extended to subspace clusters (i.e. parallel patterns). River measures may affect several structural properties, e.g. the river bank on the left and the right as well as the river bending. Similarly, constraints are often formulated for several attributes as well. Likewise, in other applications, situations which require specific measures are typically described via several sensor values.

We define subspace clusters of patterns as frequent parallel occurrences of single attribute clusters. Frequency is measured in terms of simultaneous occurrences, i.e. the number of positions in any sequence, where clusters are detected in different attributes. We formalize our notion of parallel clusters as follows:

Definition 5.4 ***Subspace subsequence clusters:***
A set of sequence clusters $\mathbf{C_1}, \ldots, \mathbf{C_n}$ *of length* k *from* n *different attributes is a subspace subsequence cluster* **SSC** *iff:*

- $\mathbf{C_1}, \ldots, \mathbf{C_n}$ *are* ***parallel*** *at some position* i*,*
  i.e. $\forall j \in \{1 \ldots n\}$ *a pattern* $P_j \in \mathbf{C_j}$ *occurs at position* i
- $\mathbf{C_1} \ldots \mathbf{C_n}$ *occur* ***frequently*** *together,*
  i.e. $|\{\mathrm{i} \in \mathbb{N},\ \mathbf{C_1}, \ldots, \mathbf{C_n}$ *are parallel at position* $\mathrm{i}\}| \geq \phi$*.*

Put informally, we are thus looking for those positions which show a pattern contained in each attribute (subspace) and which have a count equal to or greater than the threshold given (Frequency).

The frequency threshold ϕ reflects the number of positions where the subspace cluster is detected. It can be set in relation to the database size, i.e. the overall number of multidimensional sequence segments (see Section 5.6). Hence, ϕ corresponds to the relative frequency of a subspace subsequence cluster (the support) and is a key parameter depending on the application. In general, increasing ϕ decreases the number of identified subspace clusters, as more occurrences of the pattern are required. Subspace clusters detected using a higher ϕ are far more typical for the data set. Our experiments suggest that an initial setting of about 1% of the data set serves as a good starting point for analysis. As more or less patterns are desired, the threshold is adapted accordingly.

Example.
In Figure 5.3 we present a subspace subsequence cluster in two attributes. The subspace subsequence cluster SSC consists of two clusters $C_1 = \{\langle 1,2\rangle, \langle 1,3\rangle\}$ in the riverbed attribute and $C_2 = \{\langle 3,4\rangle\}$ in the bank attribute. They occur in positions S_1, S_8, T_5, thus three times. Assuming a frequency threshold of $\phi = 2$, SSC is a subspace subsequence cluster.

5.4 Efficient subsequence cluster mining

To detect subsequence clusters of arbitrary length, a naive approach might be to simply re-run a density-based clustering algorithm for each length value to detect all possible clusterings. Obviously, this leaves room for efficiency improvement.

5.4.1 Monotonicity

We avoid excess clustering steps by exploiting monotonicity properties of clusters. We show that patterns which are not dense, cannot be part of longer dense patterns. We may safely prune them from consideration in longer pattern clustering. We formalize this monotonicity first for patterns and then prove that this property holds for clusters themselves.

Theorem 5.1 ***Density Monotonicity:***
For any two patterns $\mathbf{P}, \mathbf{Q}$ *of length* k *and their respective prefix/suffix* $\mathbf{P}', \mathbf{Q}'$ *of length* $k-1$ *holds:*

$$(1) \quad \mathbf{Q} \in N_\tau(\mathbf{P}) \Rightarrow \mathbf{Q}' \in N_\tau(\mathbf{P}')$$

$$(2) \quad \mathsf{density}(\mathbf{P}) \leq \mathsf{density}(\mathbf{P}')$$

For (1) we note that an L_p norm, $p \geq 1$ is the p-root of sum of absolute differences in sequence values. This means that a reduced sum of $k-1$ of these differences is necessarily smaller than or at most equal to a sum of all k differences.

Proof: For a prefix/suffix $\mathbf{Q}', \mathbf{P}'$ of $\mathbf{Q}, \mathbf{P}$ we first show: $\mathsf{dist}(\mathbf{P}, \mathbf{Q}) \geq \mathsf{dist}(\mathbf{P}', \mathbf{Q}')$. Dropping the first or last summand in our distance sum for $\mathbf{Q}, \mathbf{P}$, we obtain for the prefixes or suffixes $\mathbf{Q}, \mathbf{P}$:

$$\sqrt[p]{\sum_{i=1}^{k} |p_i - q_i|^p} \geq \sqrt[p]{\sum_{i=1}^{k-1} |p_i - q_i|^p} \quad \text{dropping the last summand: prefix}$$

$$\sqrt[p]{\sum_{i=1}^{k} |p_i - q_i|^p} \geq \sqrt[p]{\sum_{i=2}^{k} |p_i - q_i|^p} \quad \text{dropping the first summand: suffix}$$

$\Rightarrow \mathsf{dist}(\mathbf{P}, \mathbf{Q}) \geq \mathsf{dist}(\mathbf{P}', \mathbf{Q}')$

Thus the distance between two patterns is greater than the distance between the respective prefix or suffix. Using this fact we can prove (1):

$\mathsf{dist}(\mathbf{P}, \mathbf{Q}) \geq \mathsf{dist}(\mathbf{P}', \mathbf{Q}')$

$\Rightarrow \quad W_\sigma^{\mathbf{P}}(\mathbf{Q}) \leq W_\sigma^{\mathbf{P}'}(\mathbf{Q}')$

(weighting functions are increasing with decreasing distance)

$\Rightarrow \quad (W_\sigma^{\mathbf{P}}(\mathbf{Q}) \geq \tau \Rightarrow W_\sigma^{\mathbf{P}'}(\mathbf{Q}') \geq \tau)$

(using Definition of N_τ)

$\Rightarrow \quad (\mathbf{Q} \in N_\tau(\mathbf{P}) \Rightarrow \mathbf{Q}' \in N_\tau(\mathbf{P}'))$.

Part(2) is proven using a similar argument: the density is defined as a weighted sum of the support of patterns. Their shorter counterparts, prefix or suffix, are assigned smaller distance values (part 1) and therefore larger weights.

$$\mathsf{density}(\mathbf{P}) = \sum_{\mathbf{Q} \in N_\tau(\mathbf{P})} W_\sigma^{\mathbf{P}}(\mathbf{Q}) * sup(\mathbf{Q}) \text{ (Definition of density)}$$

$$\leq \sum_{\mathbf{Q}' \in N_\tau(\mathbf{P}')} W_\sigma^{\mathbf{P}}(\mathbf{Q}) * sup(\mathbf{Q}')$$

(Part 1: $sup(\mathbf{Q}) \leq sup(\mathbf{Q}')$ as a subsequence for $\mathbf{Q}$ also matches $\mathbf{Q}'$)

$$\leq \sum_{\mathbf{Q}' \in N_\tau(\mathbf{P}')} W_\sigma^{\mathbf{P}'}(\mathbf{Q}') * sup(\mathbf{Q}')$$

(Part 1: distance of shorter patterns is smaller, weights are larger)

$= \mathsf{density}(\mathbf{P}')$ (Definition of density) $\quad \diamond$

Since density and neighborship are monotonous, we can conclude that clusters are monotonous as well:

Theorem 5.2 ***Cluster Monotonicity****:*
For any cluster $\mathbf{C}$ *of length* k*, there is a cluster* $\mathbf{C}'$ *of length* $k-1$ *such that for any pattern* $\mathbf{P}$ *and its prefix/suffix* $\mathbf{P}'$ *holds:*

$$\mathbf{P} \in \mathbf{C} \Rightarrow \mathbf{P}' \in \mathbf{C}'$$

Proof: For any $\mathbf{P}$ in $\mathbf{C}$, we have that $\mathbf{P}'$ is dense (Lemma 5.1), and since the distance of shorter patterns is smaller (Proof of Lemma 5.1), there exists

a chain of prefix/suffix patterns which guarantees compactness. This means that all prefixes/suffixes are part of a cluster (which might contain additional patterns due to the third requirement, completeness). ◇

Summing up, we know that any pattern or cluster which does not satisfy the density criterion, can be safely pruned from the search for longer patterns or clusters.

5.4.2 Indexing subsequence patterns and clusters

In order to efficiently calculate the density value for a pattern it is crucial to quickly retrieve the neighborhood of a subsequence. We introduce an index structure for patterns which supports neighborhood queries and density estimators.

Since existing index structures do not work for patterns of different length, the hierarchical index structure illustrated in Figure 5.4 was developed. The index structure is tailored to the SSC algorithm (subspace subsequence clustering algorithm) in that the bottom-up approach is supported: queried patterns always grow in length.

The index is constructed by scanning once over each sequence. Each pattern of a fixed starting length determined is added to the index structure. A pattern is represented by a path of labeled nodes from the root to a leaf. To later determine the density value of a pattern the support is annotated at each corresponding leaf node. Further on the index structure stores the position list (all starting points) for each pattern. For scalability purposes the position list is stored on hard disk. In addition, a cluster identifier (e.g. $C1$) or the flag *unclassified* (UC) is stored at each leaf node (Fig. 5.4). The SSC algorithm uses this flag to efficiently calculate the transitive closure (compactness in Def. 5.3) for a dense pattern. The density value for a pattern can be calculated by summing up and weighting the support of all patterns in the corresponding neighborhood. The index structure efficiently supports neighborhood queries by selecting the appropriate node ranges while descending the tree.

Example.

In Figure 5.4 we assume a parameter setting of $\tau = 0.2$ with a weight of 1 for patterns having distance zero and a weight of 0.3 for patterns having distance one. The density value for pattern $\langle 1, 3 \rangle$ can then be calculated in

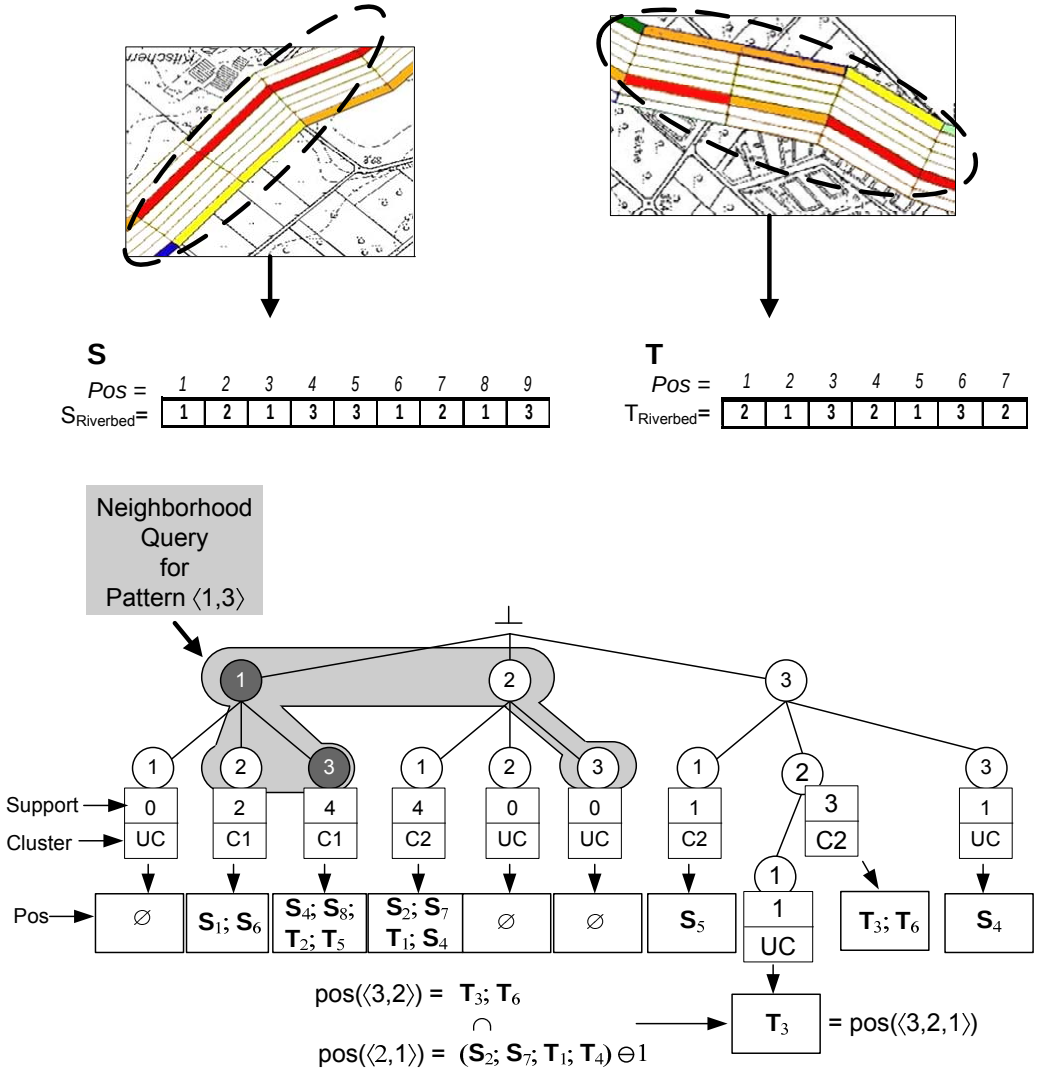

Figure 5.4: Hierarchical index structure for finding subsequence clusters

the following way: Starting at the root node the node range ("1″-″2″) must be considered. When processing node "2", for instance, all patterns containing this node have a distance of at least one (distance from $\langle 1,3\rangle$ to $\langle 2,*\rangle$ is greater than or equal to one). Since the maximal distance according to τ is bounded by 1, the only pattern which must be considered below node "2" is the pattern $\langle 2,3\rangle$. In the same way all other patterns within the specified range can be determined (in this example $\langle 1,2\rangle$, $\langle 1,3\rangle$ and $\langle 2,3\rangle$). Finally the density value for the pattern $\langle 1,3\rangle$ can be calculated by summing up the support of the leafs for each weighted subsequence:

$$\begin{aligned} density(\langle 1,3\rangle) &= W^{\langle 1,3\rangle}(\langle 1,2\rangle) * sup(\langle 1,2\rangle) + W^{\langle 1,3\rangle}(\langle 1,3\rangle) * sup(\langle 1,3\rangle) + \\ &\quad W^{\langle 1,3\rangle}(\langle 2,3\rangle) * sup(\langle 2,3\rangle) \\ &= 0.3 * 2 + 1.0 * 4 + 0.3 * 0 = 4.6. \end{aligned}$$

5.4.3 Clusters of arbitrary length

To identify clusters of arbitrary length the SSC algorithm works bottom-up by first identifying short clusters and then successively searching for longer clusters. This section proposes a method to generate new longer cluster candidates by combining appropriate shorter clusters.

The SSC algorithm elongates the investigated patterns by one in each step. Thus the index structure must be able to calculate the position list and support for constantly growing patterns. This is achieved by the algorithm: the position list for a pattern $\mathbf{P}$ of length k can be calculated by using its $k-1$ prefix and suffix. Intuitively, the pattern $\mathbf{P}$ can only occur in a sequence if its prefix starts at exactly its starting position and its suffix ends where $\mathbf{P}$ ends. This means that no extra database scans are necessary to determine its occurrence - we simply take a look at all the starting positions of its prefix and determine its intersection with its suffix shifted by one. Since both are shorter by one, they must have been processed earlier.

Example.

Figure 5.4 illustrates this algorithm for the pattern $\mathbf{P} = \langle 3,2,1\rangle$. Its prefix is the pattern $\langle 3,2\rangle$, its suffix $\langle 2,1\rangle$. We can compute the positions of pattern $\mathbf{P}$ simply by shifting the position list of the suffix one back and intersecting it with the position list of its prefix. The prefix $\langle 3,2\rangle$ occurs in positions

T_3, T_6, while its suffix $\langle 2, 1\rangle$ is found at starting positions S_2, S_7, T_1, T_4. Any sequence elongated by one which contains this suffix has to start one position earlier to end with this suffix. Any occurrence of $\langle 3, 2, 1\rangle$ will thus start at $\{T_3, T_6\} \cap \{S_1, S_6, T_0, T_3\} = \{T_3\}$. And indeed $\langle 3, 2, 1\rangle$ is found at this position as we can verify in the illustration of $\mathbf{T}$.

As we can see, patterns are efficiently extended on the fly when a longer pattern is accessed for the first time.

5.5 A subspace subsequence clustering algorithm

Our SSC algorithm exploits both monotonicity properties and density computation on the index. Working in two steps, each monotonicity property on subsequence patterns and clusters are used. The first step searches for clusters of arbitrary length while the second step combines parallel clusters.

5.5.1 Step one: Subsequence clustering

The first step of the SSC algorithm is presented in Figure 5.5. First, the index structure is created using a parameter $length_{start}$ (can be set to one to mine all patterns where desired). After the construction step the index structure contains one entry for each pattern of length $length_{start}$. Next the first cluster candidate is generated. The first candidate contains all patterns, since all of them might be dense and hence could belong to a cluster. A depth-first search on the index structure efficiently retrieves all different patterns stored in the database (method *queryAllEntries*).

Next the clustering loop starts which discovers all clusters from $length_{start}$ to $length_{max}$ (may be set to *infinity*). For every length all patterns of all cluster candidates are tested if they satisfy the density property. Each unclassified dense pattern is then expanded to a cluster by the method *ExpandToCluster*. This method creates a new cluster and assigns each dense pattern within the transitive closure (w.r.t. γ) to this new cluster.

Neighborhood queries are necessary to determine the density value (method *isDense*) and the transitive closure of a subsequence (method *ExpandToCluster*). Since all patterns of shorter length are no longer necessary after each

```
ClusterSet SSC-Phase1(Database db, int length_start, int length_max)
   Index index(db);
   index.initialCreate(length_start);          // create the index using all length_start sequences
   ClusterSet candidateSet    := { index.queryAllEntries() }; // all sequences are candidates
   ClusterSet clustSetResult  := ∅;        // result set of subsequence clusters
   foreach length from length_start to length_max do
       ClusterSet clustSet  := ∅;          // store new cluster sets

       /* Generate new clusters based on each candidate cluster */
       foreach clustCand in candidateSet do
           markUnclassified(clustCand);    // marks all sequences unclassified
           foreach seq in clustCand do

               /* Check if sequence is dense and expand to Cluster */
               if isUnclassified(seq) and isDense(seq, index) then
                    clustSet := clustSet ∪ ExpandToCluster(seq, clustCand, index);
               end if;

           end foreach;
       end foreach;

       index.prune(length); // discard unnecessary paths and position lists

       clustSetResult := clustSetResult ∪ clustSet; // store cluster

       /* Create new candidate clusters using monotonicity property */
       candidateSet := CreateCandidateCluster(clustSet, length+1, index);
   end foreach;

   return clustSetResult;
```

Figure 5.5: Subsequence clustering algorithm (Phase 1 of SCC)

```
ClusterSet CreateCandidateCluster(ClusterSet clustSet, int length, Index index)
  ClusterSet resultSet := ∅;
  foreach clust in clustSet do
    foreach seq in clust do

      /* query all sequences from index which start with Suffix(seq) */
      SequenceSet extSeqSet := index.prefixQuery( Suffix(seq) );
      ClusterSet  clustCand := ∅;

      foreach extSeq in extSeqSet do

        /* if extension is dense then put the extended sequence into the result set */
        if isDense(extSeq, index)) then
          clustCand := clustCand ∪ { seq[1]•extSeq };
        end if;
      end foreach;

      resultSet := resultSet ∪ clustCand;
  end foreach;

  return resultSet ;
```

Figure 5.6: Creating Candidate Clusters

clustering step the position lists and flags stored for these patterns can then be released (method *prune*).

After a set of clusters is discovered for a specific length the cluster set is stored in the result set *clustSetResult* and new cluster candidates are determined (*candidateSet*) by using the old cluster set (*clustSet*). Figure 5.4 also illustrates the generation of a cluster candidate. This step utilizes the monotonicity property of subsequence patterns in Lemma 5.2: Only subsequence patterns whose prefix and suffix are member of an already discovered cluster (and hence are dense) must be considered as members of a new cluster candidate.

The algorithm to calculate the corresponding cluster candidates is based on the discovered clusters of the previous step (Fig. 5.6). The method *CreateCandidateCluster* loops over all patterns of all clusters and tries to extend each pattern. For this purpose the suffix of $length - 1$ is extracted from each pattern and all patterns which start with the extracted suffix are queried from the index (*prefixQuery*). Each queried pattern which satisfies the density property is then used to create an elongated pattern by concatenating the appropriate prefix and suffix. These queries are also supported by the proposed index structure. The elongated patterns are finally assigned to a new cluster candidate.

Example.

Consider a cluster containing only one pattern $\mathbf{P} = \langle 5, 1, 3, 7 \rangle$. This cluster of length 4 is extended to a cluster candidate of length 5 in the following way: First a query using the suffix of $\mathbf{P}$, $\langle 1, 3, 7 \rangle$, is performed. We assume that the intersection with all possible patterns of length 4 results in a set $\{\langle 1, 3, 7, 3 \rangle, \langle 1, 3, 7, 9 \rangle\}$. Next the patterns are checked whether they are dense or not. This can be achieved by simply evaluating the annotated cluster flag (a dense pattern must belong to a cluster). Let's assume $\langle 1, 3, 7, 9 \rangle$ is dense and $\langle 1, 3, 7, 3 \rangle$ is not dense. In this case the two patterns $\langle 5, 1, 3, 7 \rangle$ and $\langle 1, 3, 7, 9 \rangle$ are used to create the elongated pattern $\langle 5, 1, 3, 7, 9 \rangle$ of length 5. This pattern is assigned to a new cluster candidate and returned to the SSC clustering algorithm.

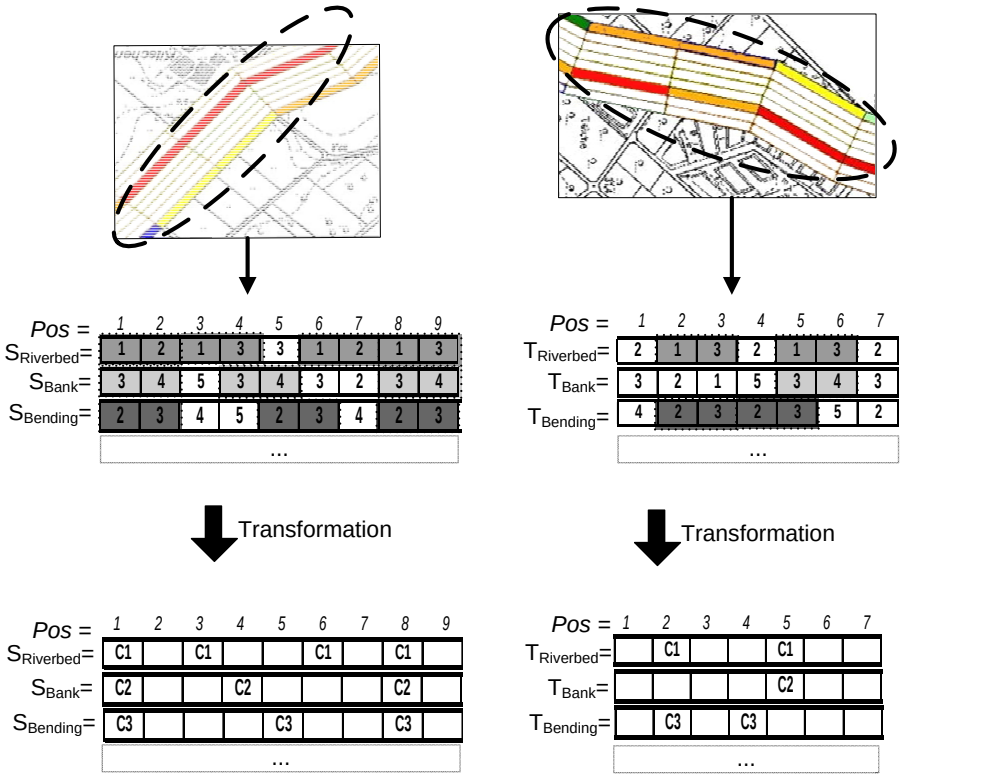

Figure 5.7: Transformation example

5.5.2 Step two: Subspace subsequence clustering

Having discovered dense patterns and combined them to clusters, we identify those clusters which occur parallel in a subspace of the dataset (see Def. 5.4). Since for any cluster of length k all corresponding clusters of length $k-1$ have previously been detected, we use the monotonicity property presented in Lemma 5.2.

An important property of subspace subsequence clusters is that a subsequence of a specific length may belong to no more than one cluster. Hence any position in a sequence is the starting point for at most one density-based cluster of a fixed length. This property is used to transform the sequence database from a value representation to a cluster representation. After this transformation it is possible to discover subspace clusters by efficient frequent itemset mining techniques.

We use the following representation to apply the frequent itemset mining: clusters starting at the same position in different attributes are combined to one itemset. The clusters are identified by their cluster-ids. A frequent itemset contains often occurring subspace clusters in which each cluster belongs to a different attribute.

Example.

Figure 5.7 illustrates the transformation process. In the upper part, three attributes of sequence **S** are shown, with clusters highlighted in gray colors. Assumed are the following sequence clusters: $C_1 = \{\langle 1,2\rangle, \langle 1,3\rangle\}$ in S_1, $C_2 = \{\langle 3,4\rangle\}$ in S_2, $C_3 = \{\langle 2,3\rangle\}$ in S_3. They are transformed according to their positions as follows: Position 1 in the sequence **S** would yield a transaction $\{1,2,3\}$ since a cluster occurs in each dimension. Position two has no starting clusters, position three only cluster C_1 and so on (lower part of Fig. 5.7). We can therefore immediately derive the frequency of itemsets from the cluster information: the itemset $\{C_1\}$ occurs six times, $\{C_2\}$ four times, ..., and finally $\{C_1, C_2, C_3\}$ two times.

We use the Frequent Pattern (FP) growth algorithm proposed by Han et al. for the extraction of frequent itemsets [HPY00]. The tree representation is very suitable for our task since database scans are avoided. Recall that the FP-tree builds a path annotated by support values for all frequent

items. To obtain frequent itemsets conditional FP-trees are constructed iteratively for each frequent item. As mentioned above we use the following representation to apply the FP-tree: one-dimensional subsequence clusters starting at a certain position are itemsets containing cluster-ids (see Fig. 5.7). The FP-tree supports efficiently determining frequent patterns w.r.t. ϕ. A frequent itemset contains often occurring correlated clusters in which each subsequence cluster belongs to a different dimension. Thus the result of the FP-tree algorithm contains subspace subsequence clusters as described in Def. 5.4. In order to find subspace subsequence clusters of any length the FP-tree algorithm is started for all different lengths for which clusters have been found. Since the FP-tree works extremely fast, clustering arbitrary lengths is efficient.

5.5.3 Analytical evaluation

Subspace clustering is a highly complex task, as the number of subspaces is exponential in the number of attributes. Further on, detecting subspace subsequence clusters of arbitrary length is quadratic in the sequence length. Consequently, subspace subsequence clusters detection is a challenging problem. To ensure efficiency, our algorithmic approach therefore bears on each of these aspects.

First, concerning arbitrary lengths of possible patterns, our index structure avoids re-building potentially frequent patterns from scratch. Only frequent patterns, not the actual sequences nor the infrequent patterns, are actually processed when elongating clusters. For each of these patterns, compact representations are stored and position lists are kept on hard disk to reduce main memory usage. Second, only subspaces which contain clusters in single attributes are mined for subspace subsequence clusters. This greatly reduces the number of potential combinations. Moreover, by mapping sequences to cluster ids, the FP-growth algorithm allows for fast detection of subspace subsequence clusters. Memory requirements could be further reduced by applying a secondary storage extension of the FP-tree method as suggested e.g. in [GZ04].

We validate this analysis in the experiments which show that our algorithm scales almost linearly in terms of both length and numbers of attributes on different data sets.

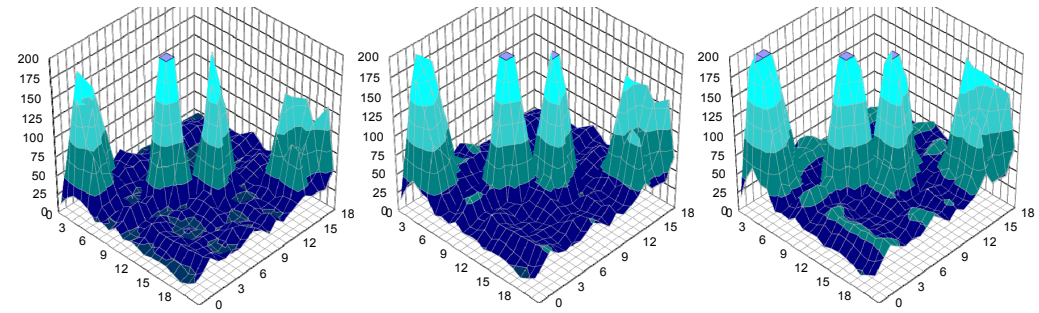

Figure 5.8: Influence of σ on density ($\sigma = 0.75, 0.9, 1.0$ from left to right)

5.6 Experimental evaluation

We ran experiments on synthetic and real world data to evaluate the quality and performance. Synthetic data is used to evaluate different cluster parameters and to develop guidelines and heuristics for supporting users in setting up parameters. It also evidences the algorithm's scalability as well as its quality in detecting all generated clusters. Real world data demonstrates the usefulness of the results for domain experts. Our implementation is in Java, and experiments were run on Pentium 4 machines with 2.4 Ghz and 1 GB main memory.

5.6.1 Parameter Setup

Several parameters can be used to tune our algorithm to domain specific needs. We therefore develop appropriate guidelines and heuristics for choosing reasonable parameter settings. To study the effect of different parameter setups on the clustering result we generated different synthetic data sets. For each data set we varied the number of clusters per sequence and subspace clusters contained in the data.

As suggested in Section 5.3 we use a Gaussian kernel as a weighting function. Thus the first parameter to set is the value for σ. We propose using a density plot to determine the effect of different σ values on the density for length two patterns. The data set used to demonstrate this heuristic is eight dimensional and contains four subspace clusters. Figure 5.8 illustrates the density plot for one attribute of the synthetic dataset using $\sigma = 0.75$, 0.9, and 1.0, respectively. Four separate clusters can be seen, which mainly consist of patterns of similar values (on the diagonal). Decreasing the value

Param.	Usage	Setup	Exp. 1
δ	**Density Threshold** Separates noise from clusters.	see left	10
σ	**Expected Blur (Fuzziness)** Standard deviation in Gaussian Kernel. Value ~1 for categorical.	0.75-1.25	0.9
τ	**Significance Threshold** Cuts off insignificant weights. For very small value almost no error.	<<1	0.005
γ	**Compactness Parameter** Max. distance between patterns in cluster; ~1 for categorical.	1	1
ϕ	**Co-occurring Frequency** Correlates to data coverage of multicluster.	1% of the dataset	30

Figure 5.9: Parameter guidelines

of σ corresponds to splitting the rightmost cluster around coordinates (16,16) into two separate clusters (leftmost illustration). As this would result in two clusters which would be a lot less frequent and distinct from the remaining three clusters, splitting is considered inappropriate and a higher value for σ should be chosen. On the other hand, further increasing the value of σ corresponds to merging the two clusters at the center into a single cluster (rightmost illustration). This would create a larger and more frequent cluster than the remaining two. Hence, a lower value for σ is needed to separate these clusters.

These plots, as well as further experiments, suggest a choice for σ of about one. This allows separation of clusters as well as aggregation of similar values. A choice of one for σ also reflects the fact that a deviation in one value is generally deemed tolerable in many ordinal settings. An appropriate minimum density value can also be derived from the density plot. The ordinate value which separates the clusters from the "noise floor" can be visually determined and extrapolated to longer patterns. This leads to appropriate values for the density threshold ρ. (e.g. for this dataset $\rho = 10$). Note that the density plot can be created easily during the initialization of the index structure.

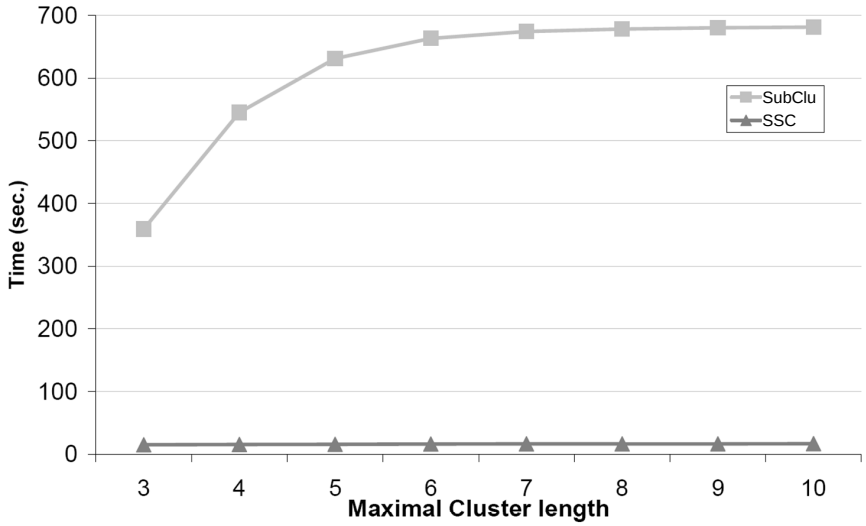

Figure 5.10: Comparison with SUBCLU

The remaining parameters can be determined in a straightforward manner. τ is set to very small values (0.01 or less) to cut off insignificant density values. Similar patterns should be clustered together which leads to a compactness threshold of one ($\gamma = 1$) in ordinal settings like the river dataset. Finally, experts are only interested in subspace clusters which cover a significant part of the dataset (e.g. one percent minimum frequency). Figure 5.9 summarizes our heuristics and parameter settings for the experiments.

Using this parameter setting the SSC algorithm indeed finds all subspace subsequence clusters hidden in the synthetic dataset. This first experiment demonstrates the effectiveness of the SSC algorithm and indicates that the developed heuristics help users find reasonable parameters.

5.6.2 Synthetic Data

As mentioned in Section 5.4.1, a naive approach for detecting clusters of arbitrary length would be to re-run a density-based subspace clustering algorithm for all possible lengths. We compare the performance and the results of the SSC algorithm with SUBCLU [KKK04], an extension of the density-based DBSCAN [EKSX96] algorithm to subspace clustering. Since SUBCLU was not developed to cluster subsequences we had to extend the original implementation by the density notion presented in Section 5.3.

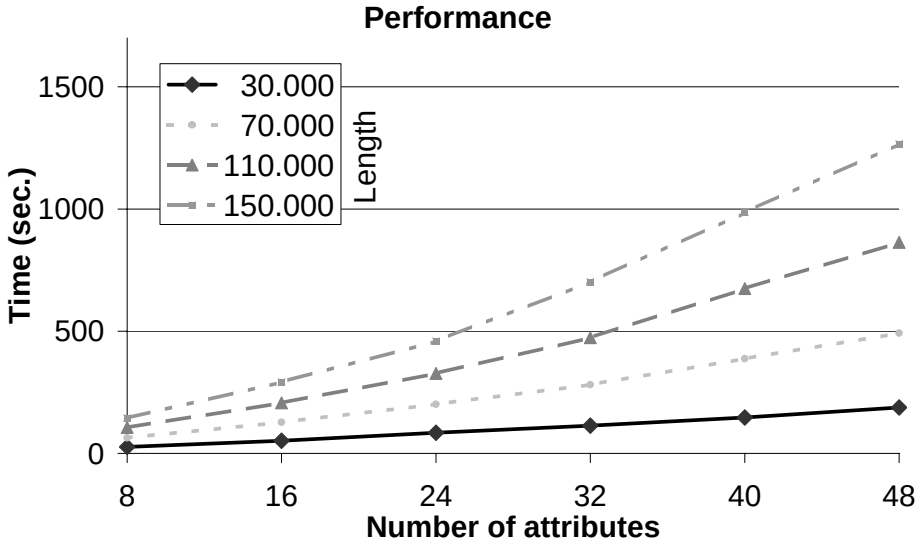

Figure 5.11: Scalability for number of attributes

To evaluate the scalability of our SSC algorithm in comparison with existing approaches, we varied the length of the data sequences and the number of sequences contained in the database. Additionally we investigated the influence of the number of different labels per sequence domain on the performance of the SSC algorithm. For this purpose we implemented a data generator which creates density-connected clusters in each sequence. The data generator gets the number of clusters, and the size and length of the database as input parameters. To parameterize the position and size of a cluster, a starting sequence and the number of pattern belonging to the cluster is specified for each cluster. To generate subspace clusters some of these clusters are correlated to parallel subspace subsequence clusters. All values not belonging to a sequence cluster are uniformly distributed based on the size of the corresponding domain to generate noise.

In our first experiment we generated a data set consisting of eight attributes with twenty different labels. We hid three to six clusters of length five to ten in each sequence. Each hidden subspace cluster has a minimum frequency of one percent and consists of three to six patterns. Figure 5.10 illustrates the runtime of both algorithms. Note that SSC is faster by an order of magnitude. The runtime of both algorithms depends on the number of subsequences belonging to a cluster. Since longer subsequences are less often dense the time to investigate longer subspace subsequence clusters is

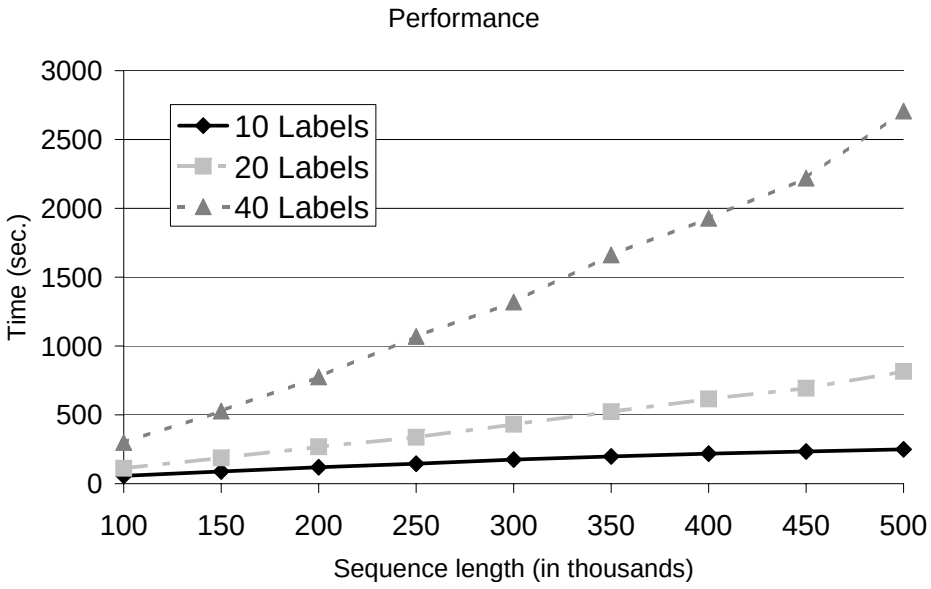

Figure 5.12: Scalability for sequence length

nearly constant. As far as the quality of the result is concerned, both algorithms discovered the main patterns of the six subspace clusters hidden in the database.

In our second experiment we used the same hidden clusters as in our first experiment but varied the number of sequences contained in the database. Figure 5.11 shows the result for four different sequence lengths (from 30,000 to 150,000). Even in 48 attributes the SSC algorithm is capable to find subspace clusters in reasonable time. Once again, we have only slightly more than linear behavior for increasing number of sequences.

To evaluate the scalability in terms of sequence lengths we again used the eight-dimensional data set from our first experiment and recorded the time for sequence lengths from 100,000 to 500,000. Additionally three different domains were used, depicted in Figure 5.12 as separate lines for 10, 20 and 40 ordinal values per sequence attribute. In order to keep the experiments comparable we adjusted σ using our density plot heuristics from $0.6, 1.2, 2.4$, respectively. The SSC algorithm shows linear behavior for 10 labels and slightly superlinear behavior for 20 and 40 labels. Even the largest setup with 40 different categories and $500,000$ sequence segments takes less than $2,800$ seconds. This means that our algorithm is capable of handling very large databases with large domains.

5.6.3 Real world data

This section evaluates the effectiveness and efficiency of our algorithm on different real world datasets. Our analysis mainly focuses on the flowing water renaturation project of the European Union mentioned before. Additionally, we investigate the effectiveness of the SSC algorithm on temporal data sets, using data from the "National Fire Danger Rating Retrieval System".

River data

The river data set consists of 120,000 river segments describing the structural properties of a river, such as the structure of the riverbed. Experts working on renaturing rivers are interested in finding co-occurring clusters to determine those subsequence patterns which occur repeatedly in the river database and which allow to derive broad potential improvement through measures designated. As a final result, experts should be capable of summarizing river segments into measure packages, ranking them according to the quality enhancements expected. This can then be used to create supra-regional schedules for political decision-making.

For the river dataset a value for σ between 0.7 and 0.85 has shaped up as a reasonable parameter. The SSC algorithm identifies more than a thousand clusters of length four by using a value for σ of 0.7 with a corresponding value for ρ. Many of those clusters do not show when mining clusters of length five. By using a higher value for σ more patterns are considered similar and cluster count drops between lengths five and six. For this dataset experiments have shown that independent of the σ value, subspace clusters in many attributes can be found only at length four to six. Beyond this length, parallel patterns are rare. This is an interesting result for the hydrologists studying this data. They find that they should develop sensible packages of measures in this range and estimate costs for packages of about 500 meters river improvement.

Figure 5.13 illustrates the runtime of the SSC algorithm for phase one (searching for clusters of arbitrary length) and for phase two (searching for multiclusters) separately as well as for both. As we can see, the time requirements for mining all clusters of arbitrary length are distributed rather evenly between the two phases. The total time for mining subspace clusters of arbitrary pattern length demonstrates the efficiency of our approach. Note that with increasing maximal length, few additional dense patterns are detected such that the increase in time consumption slows down. The steepest ascent

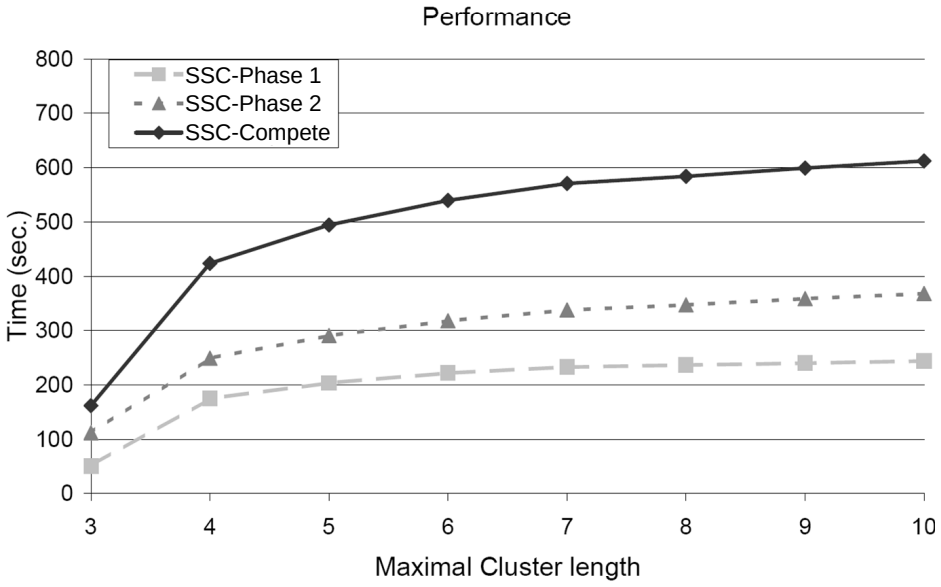

Figure 5.13: Performance

is for lengths of up to six, which corresponds to our result findings.

Similar to the synthetic dataset, we also applied SUBCLU on this real world dataset. However, even the first iteration of SUBCLU for subspace clusters of length ten did not finished after ten hours. One reason why SSC works extremely faster on the investigated dataset than SUBCLU are the time consuming neighborhood queries on the nineteen-dimensional data points. Even the use of index structures like the R-Tree does not speed up these neighborhood queries in these high dimensionalities. Another reason is the efficient combination of dimensions using an FP-tree as done by SSC.

For hydrologists to see how the detected subspace clusters are distributed and check in which areas the designed measures are applicable or where additional measure engineering is required, we visualize the subspace clusters in a geographical information system. An example for a cluster visualization is given in Figure 5.14. Those river segments which are part of the cluster are marked by dark lines. The corresponding cluster summary is visualized on the left side. It indicates the cluster length of five river sections (top to bottom) as well as the cluster range of ten out of nineteen attributes (left to right).

Additional tools for hydrologists give detailed information beyond this summary. Experts may browse clusters for an overview of the attributes

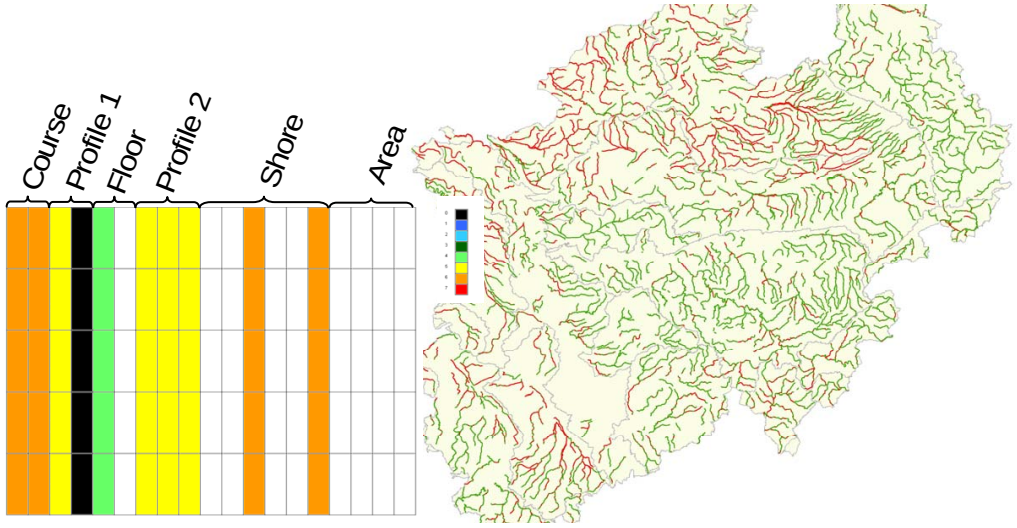

Figure 5.14: Cluster Visualization for river data

contained in clusters, their value distributions as well as their occurrence in the rivers. Moreover, individual attributes and river values may be checked for containment in clusters. By joining this information with the packages of measures designed, field experts can easily identify areas which are not yet met by measures.

Annotating the measures with cost and duration information, political decision making is supported. Hydrologists used the information derived from these clusters to build a decision support system [Bar05] that gives concise summaries as well as detailed inspection of the state of the rivers as it is now as well as a prognosis for future development depending on the packages of measures chosen.

Weather data

The second real world dataset is retrieved from the "National Fire Danger Rating Retrieval System" which provides senor data from various weather stations. We use hourly weather variables from 1998 to 2005 about temperature, relative humidity, wind speed and direction, weather status etc. Overall the dataset contains fifteen variables measured at $27,048$ time points [GFC05]. This weather dataset contains a mixture of real valued and categorical variables. The continuous attributes are transformed into ordered categories using the transformation technique presented in [LKLC03]. After

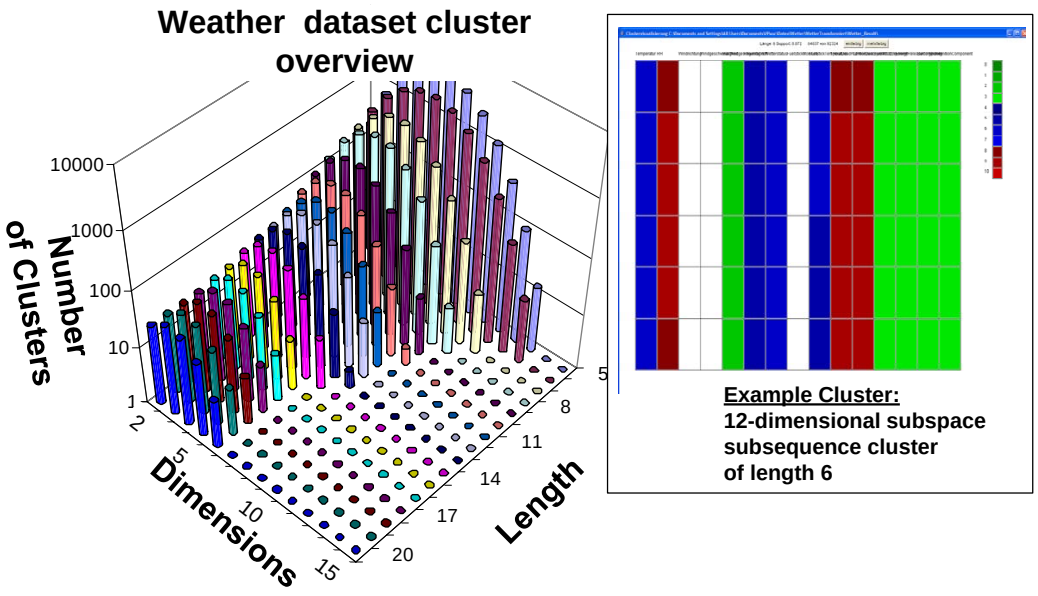

Figure 5.15: Clustering of weather dataset

normalization we used ten symbols for the quantization.

The weather dataset allows us to determine the quality of the SSC algorithm. Based on the weather variables the "National Fire Danger Rating System" calculates some indices like the "Ignition Component" which relates the probability that a fire that requires suppression action will result if a firebrand is introduced into a fine fuel complex. These indices are influenced by many variables measured in the rating system. They form natural co-occurring patterns in the data which we use to demonstrate that the SSC algorithm does find these subspace clusters. By inspecting the result we indeed find the correlation between the index variables and all measured sensor data. Figure 5.15 (right side) illustrates an example subspace cluster which nearly contains all attributes. The cluster overview (Figure 5.15 left side) shows many clusters containing nearly all attributes (13 and 14) with a length of up to seven. Thus the SSC algorithm indeed finds the intrinsic structure in a real valued dataset.

5.7 Conclusion and further work

Domain experts are supported in their need for pattern detection in sequence databases such as river quality records. The information derived using our approach is incorporated in a decision support system devised by hydrologists. Future work will concentrate on integrating GIS information to handle non-sequence spatial information as well and to extend the model to graph structures [KKSS04].

Part II

Efficient density-based methods for classification

Chapter 6

Classifying multi-dimensional data

Classifiers are part of many complex pattern recognition systems like speech recognition systems, image recognition systems, handwriting recognition system or text recognition system. Given a set of class labels categorizing the objects classifying an object means to predict the class label of unknown objects. Typically the first step of a recognition system is to extract a multi-dimensional feature from the original objects. Based on these features a training set of labeled objects is used to build a classifier. Hence, the challenge in classification is to learn a classifier from the training data which reflects the structure given by the classes.

6.1 Related Work

Classification is a field of extensive research. Several distinct branches of classification techniques have been developed.

Neural networks imitate the brain's parallel information processing by weighted connections between individual units (the "neurons") [CW02, MP43, SdSA05]. Individual units are activated based on their input values and an activation function. By adjusting weights during training, a discriminant function between individual classes is learned. Output values of the units generated on input data are then used for classification. Neural networks have shown to be good classifiers in applications where no understanding of the classification process is necessary, since no explication component is gen-

erated. The neural network or its weights generated during training provide no insight into the patterns learned.

Decision trees provide explanatory components by visualizing the decision taken during classification [Mur98, Qui86, Qui92]. The idea is to build a model on training data by successively partitioning the data along some splitting attribute which best separates the data according to the given class label. The resulting tree is then used to classify incoming data tuples by following the branches corresponding to this tuple until a leaf containing class information is found. An important question in this field is the choice of appropriate splitting attributes. Several proposals have been made, based on information gain, gini index or other specifically designed criteria [Qui86, Qui92]. In general, decision trees have been successfully applied to predict class labels in application domains where global patterns are present. However, when it comes to noise and local patterns, decision trees fail to reflect class structures. This is due to the fact that decision trees are built level by level, i.e. the choice of splitting attributes is based on a greedy-style evaluation strategy.

Bayes classifiers follow a statistical approach [Bay63, DHS01]. Probabilities of individual attribute values given a class, are used to classify new objects. To apply Bayes classifiers, the data distribution has to be estimated from the training data. A very simple approach is the naive Bayes classifier which assumes strong independence between attributes. Density distribution is determined for each class globally, thus local patterns are not identified. Details about Bayes classification is given in Chapter 6.3.

Support vector machines (SVMs) proposed by Vladimir Vapnik and Aleksei Chervonenkis [VC74] classify objects based on separating hyperplanes between the classes. As linear separation of the classes is not always possible, SVMs rely on two concepts: first they embed the input data into a higher dimensional space and second, they use soft margins to find an adequate separation hyperplane [Bur98]. Instead of embedding the data into a higher dimension, it suffices to compute the inner product in the higher dimensional space using a kernel function. As an optimization problem has to be solved to find the separating hyperplane SVM large data sets with many classes may lead to a high runtime complexity. Recent work has proven that the classification complexity depends linearly on the size of the data set [Ste03].

Another prominent approach to classification is the "lazy" classification by nearest neighbors [PF70]. The basic idea is not to build a model before-

hand, but instead query classified data (training data) similar to the data set to be classified as it comes in. Classifiers using this technique need to define an appropriate similarity model to identify these "nearest neighbors" and a voting criterion in case these neighbors belong to different classes. Popular solutions include majority voting, inverse distance weighted aggregation, etc. In Chapter 6.4 we discuss the nearest neighbor classifier in more detail.

6.2 Basic notations

For the rest of this chapter we use the following notation:

Definition 6.1 ***Class labels and training sets.***
The set of class labels is defined as:

$$\mathbf{C} = \{c_1, \ldots, c_n\}.$$

A training set based on a d-dimensional continuous feature space containing n labeled objects is defined by:

$$\mathbf{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \text{ with } \mathbf{x}_i = (x_{i_1}, \ldots, x_{i_d}) \in \mathcal{R}^d \text{ and } y_j \in \mathbf{C}$$

We further select the set of objects from the training set belonging to a specific class c_i by:

$$\mathbf{D}_{c_i} = \{(\mathbf{x}_j, y_j) \in \mathbf{D} | c_i = y_j\}.$$

Finally, a classifier can be defined as a mathematical function which maps an object to label.

Definition 6.2 ***Classifier.***
A classifier $\mathcal{G}$ maps a feature vector $\mathbf{x}$ to a class label c_i:

$$\mathcal{G} : \mathcal{R}^d \rightarrow \mathbf{C}$$

To evaluate the performance of a classifier a test set of labeled objects is often used. The test set is not used for training. A common way to obtain a training and test set is to partition a set of labeled objects into m partitions (folds) and to use one fold for testing and the other folds for training.

Definition 6.3 ***Classification accuracy.***
The classification accuracy $\mathbf{A}$ *for a given classifier* $\mathcal{G}$ *and a test set of* n' *labeled objects* $\mathbf{T} = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_{n'}, y'_{n'})\}$ *is measured as the ratio of correctly classified object to the total number of objects:*

$$\mathbf{A}_{\mathcal{G},\mathbf{T}} = \frac{|\{\mathcal{G}(x'_j) = y'_j\}|}{n'}$$

In this thesis, we focus on statistical and nearest neighbor classifiers. Statistical classifiers use the Bayesian decision theory for classifying objects. The fundamental theory of the Bayesian classifier provides a broad range of theoretical analysis ranging from asymptotical behavior to mean error rates for many applications. The well studied properties of Bayes classifiers make it a widely used approach for classification. In practical applications the nearest neighbor classifier is very popular as it is easy to implement, no assumption about the data is made and it typically shows a good performance. Nearest neighbor classifiers are also called lazy classifiers as they do not learn any density distribution or other model from the training data. We go into detail of statistical classifiers and nearest neighbor classifiers in the next section.

6.3 Bayes classification

Bayes classifiers constitute a statistical approach [Bay63, DHS01]. Conditional probabilities of individual attribute values per class are used to classify new objects. Bayesian classification has been successfully used in numerous applications in different domains. To apply Bayes classifiers, the data distribution is estimated from the training data. Different variants of classifiers exist which are based on the Bayesian classification approach. The naive Bayes classifier uses a simple model assuming strong independence of the dimensions. Other models used for Bayes classification do not make this strong independence assumption, e.g. dependencies between dimensions are taken into account by considering covariances, or more complex structures are represented by multi-modal densities.

The general idea of Bayesian classifier is to estimate the probabilistic distribution for the different classes. Based on a statistical model of the class labels, the Bayes classifier simply chooses the class with the highest *a posteriori probability* $P(c_i|\mathbf{x})$ given $\mathbf{x}$. To compute the class having the highest a posteriori probability $P(c_i|\mathbf{x})$, the Bayes rule

$$P(c_i|\mathbf{x}) = \frac{P(c_i) \cdot p(\mathbf{x}|c_i)}{p(\mathbf{x})}$$

is used. Following this rule the a posteriori probabilities of a query object can be estimated from the *a priori probability* $P(c_i)$ and the *class conditional density* $p(\mathbf{x}|c_i)$. Both the a priori probability and the class conditional density are estimated from the training data for each class individually. Since the Bayes classifier is only interested in the class having the highest probability it is typically not necessary to compute the denominator $p(\mathbf{x})$.

Definition 6.4 ***Bayes classification.***
The Bayes classifier estimates the a priori probability $P(c_i)$ *and the class-conditional density* $p(\mathbf{x}|c_i)$ *based on the training data* $\mathbf{D}$ *and assigns an object* $\mathbf{x}$ *to the class with the highest a posteriori probability:*

$$\mathcal{G}_{Bayes}(\mathbf{x}) = \underset{c_i \in \mathbf{C}}{argmax}\{P(c_i|\mathbf{x})\} = \underset{c_i \in \mathbf{C}}{argmax}\{P(c_i) \cdot p(\mathbf{x}|c_i)\}$$

The a priori probability $P(c_i)$ can be easily estimated from the training data as the relative frequency of each class $P(c_i) = \frac{|\mathbf{D}_{c_i}|}{|\mathbf{D}|}$. Since $\mathbf{x}$ is typically multi-dimensional, the task of estimating the class-conditional density $p(\mathbf{x}|c_i)$ is much more complex. To estimate the class conditional probability density functions only objects belonging to the respective class are considered.

Estimating probability densities from observations has been widely studied in statistics. A simple method for estimating the class-conditional densities is to use the naive Bayes approach which makes an independency assumption of the dimensions. Using a normal (Gaussian) distribution to model each dimension is a common approach. A multivariate normal distribution additionally considers dependencies between dimensions by using a covariance matrix.

Any unimodal model assumption (e.g. a single Gaussian distribution) may not reflect the true data distribution in real world data sets. Mixture densities relax the assumption that the data follows an unimodal model. Instead of using one model mixture densities a combination of probability density functions to represent the data. Kernel densities do not make any assumption about the underlying data distribution (thus often termed "model-free" or "non-parameterized" density-estimation). We discuss the three different ways to implement the Bayes classifier in the next sections.

6.3.1 Naive Bayes

The basic idea of the naive Bayes classifier is to assume independent features [DHS01]. More precisely for a multi-dimensional feature vector $\mathbf{x} = (x_1, \ldots, x_n)$ the class-conditional density $p(\mathbf{x}|c_i)$ is calculated by:

$$p(\mathbf{x}|c_i) = \prod_{i=1}^{d} p(x_i|c_i)$$

Since this approach is very simple it is also sometime called "Idiot's Bayes" [HTF02]. Different distributions per dimension can be applied. In this thesis we focus on normal distributions which is reasonable for many applications. The naive Bayes approach for a Gaussian distribution is given by:

Definition 6.5 ***The Gaussian naive Bayes classifier.***
The naive Bayes classifier based on Gaussian distributions assumes an independent Gaussian distribution per dimension to determine the class conditional density for an object $\mathbf{x}$*:*

$$p_{naive}^{Gauss}(\mathbf{x}|c) = \prod_{i=i}^{d} \frac{1}{\sigma_i\sqrt{2\Pi}} e^{-\frac{(\mathbf{x}_i-\mu_i)}{2\sigma_i}} = \frac{1}{\left(\prod_{i=i}^{d}\sigma_i\right)\sqrt{(2\Pi)^d}} e^{-\sum_{i=1}^{d}\frac{(\mathbf{x}_i-\mu_i)}{2\sigma_i}}$$

$$\textit{with } P(c) = \tfrac{|\mathbf{D}_{c_i}|}{|\mathbf{D}|},\ \mu_i = \tfrac{1}{|\mathbf{D}_{c_i}|}\sum_{\mathbf{x}\in\mathbf{D}_{c_i}} x_i \textit{ and } \sigma_i = \tfrac{1}{|\mathbf{D}_{c_i}|}\sum_{\mathbf{x}\in\mathbf{D}_{c_i}} (x_i-\mu_i)^2.$$

Due to the independency assumption the naive Bayes classifier determines the free parameters μ_i, σ_i for each dimension individually. Hence, implementing the training phase of a naive Bayes classifiers can be done efficiently.

Using d independent Gaussian distributions corresponds to one multivariate (multi-dimensional) Gaussian distribution without covariances (see second equation in Definition 6.5). This fact is also illustrated in Figure 6.1. For two different classes (blue, red) the class-conditional distribution is illustrated using one multi-dimensional Gaussian per class. The objects contained in the training set are depicted by blue and red points, respectively. As we can see the multi-dimensional Gaussian corresponds to the multiplication of two one-dimensional Gaussian distributions (illustrated by the red and blue line).

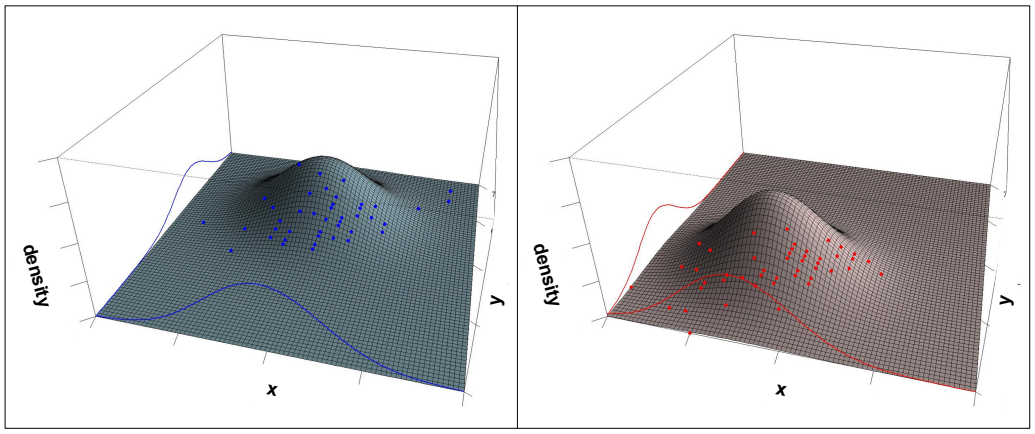

Figure 6.1: Class conditional density for two classes (blue and red) using the naive Bayes approach (independency assumption illustrated by the one dimensional normal distributions)

Considering the example depicted in 6.1 the naive Bayes classifier assigns an object to that class which has the higher density for the depicted unimodal Gaussian distributions. To determine the density for an object the naive Bayes approach only evaluates d functions per class. This makes the naive Bayes classifier a very efficient classifier in terms of classification runtime.

6.3.2 Mixture densities

Using an unimodal distribution (a distribution having only one maximum) is often not appropriate to capture the intrinsic structure of a data set. A very intuitive example for a distribution containing two modes is the body height of men and women. A data set for the body height of a population typically shows one mode for women and one mode for men. Consequently such data sets are not well represented using an unimodal density.

The expectation maximization algorithm (EM algorithm) first introduced by Dempster et al. [DLR$^+$77] is a popular tool to learn a mixture of densities from a given training set. Based on a maximum likelihood estimator the free parameters of a mixture density are determined [HTF02]. The result of the EM algorithm can be interpreted as a clustering as each density represents one cluster (see also Chapter 1.1). A mixture of Gaussian densities is defined as follows:

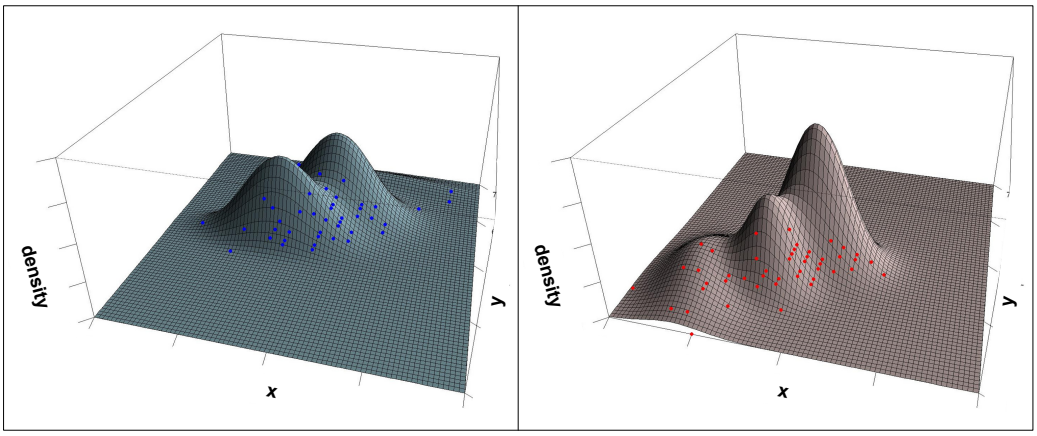

Figure 6.2: Class conditional density for two classes (blue and red) using Gaussian mixture densities with three components per class

Definition 6.6 ***Gaussian mixture densities.***

A multivariate Gaussian mixture model combines k Gaussian probability density functions $g(\mathbf{x}, \mu, Cov)$ *of the form*

$$g(\mathbf{x}, \mu, Cov) = \frac{1}{(2\pi)^{d/2} \cdot det(Cov)^{1/2}} e^{\left(-\frac{1}{2}(\mathbf{x}-\mu)^T Cov^{-1}(\mathbf{x}-\mu)\right)}$$

where Cov is a covariance matrix, $det(Cov)$ *the determinate of Cov and* Cov^{-1} *the inverse. When combined, the functions are weighted with an individual weight* w_i *with* $\sum_{i=1}^{k} w_i = 1$*. A Gaussian or normal mixture model is than defined as:*

$$p_{\mathcal{M}ixtue}^{Gauss}(\mathbf{x}|c) = \sum_{i=1}^{k} w_i \cdot g(\mathbf{x}, \mu_i, Cov_i)$$

In Figure 6.2 the EM algorithm was used to train a mixture of densities using 3 components for each class. We used the same data set as in the last example (see Figure 6.1). The blue class represented in the left part consists of two Gaussian distributions having a high weight ($w_1 \approx w_2 \approx 0.5$) and one Gaussian having a low weight ($w_3 < 0.1$). Hence, the blue data set mainly contains two clusters. For the red class (right part of the figure) three different clusters can be clearly identified.

Mixture models show a good performance both in terms of efficiency and effectiveness if a sufficient number of components is used to represent the

data set [ALV03]. However, in practical applications, where the number of components is not known in advance, using a fixed number of components is often not adequate. Recent work has shown that the number of components of a mixture model grow with the size of the database [GM03]. Hence, for large data sets mixture models have a high runtime complexity for both training and classifying objects.

6.3.3 Kernel density estimation

Density estimation using kernels is called nonparametric as no rigid assumptions about the density model is made nor are independent dimensions assumed. The basic idea of kernel estimators is to let the data speak for themselves [Sil86]. Kernel densities are a special case of mixture densities where each data point induces its own component. Kernel density estimation has been first introduced by Emanuel Parzen [Par62] for the univariate case. Several later works extended the univariate kernel to the multivariate case [Sil86, Epa69].

In this chapter we focus on radial symmetric kernel functions. Generally, a kernel function $\mathbf{K}$ satisfies the following condition $\int_{[0...1]^d} \mathbf{K}(\mathbf{x})d\mathbf{x} = 1$. Definition 6.7 gives the definition for the class conditional probability density function $p(\mathbf{x}|c)$ based on a kernel $\mathbf{K}$ with smoothing parameter (bandwidth) h.

Definition 6.7 ***Kernel densities.***

For a given kernel $\mathbf{K}$ *with smoothing parameter* h_i*, the class conditional density probability is given by:*

$$p_{kernel}(\mathbf{x}|c) = \frac{1}{|\mathbf{D}_c| \cdot h^d} \sum_{\mathbf{x}_i \in \mathbf{D}_c} \mathbf{K}\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|}{h}\right)$$

Kernel estimators can be seen as a sum of influence functions centered at each data object. They can even be applied if the nature of the true density model is not known. Thus, kernel estimators make no rigid assumption about the density distribution. Figure 6.3 gives an example for the two classes already depicted in the last two subsections. As we can see the density model based on kernel densities automatically adapts to the training data.

For Bayes classification kernel densities has shown its usefulness in many applications [JL95]. Especially for huge data sets the estimation error using

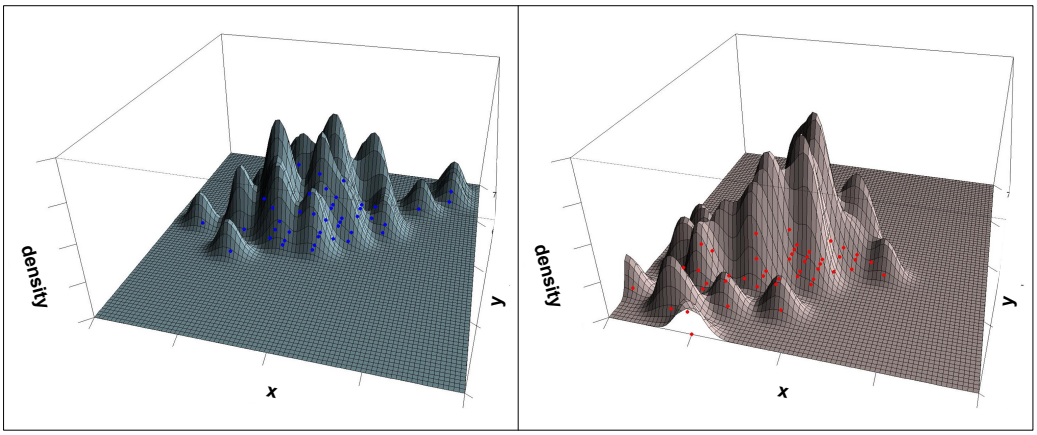

Figure 6.3: Class conditional density for two classes (blue and red) using kernel densities

kernel densities is known to be very low and even asymptotically optimal. A thorough analysis of the naive Bayes compared with kernel estimate Bayes showed that the kernel approach outperforms the normal naive Bayes on most datasets and shows far better classification accuracy than other discretization methods [Bou04]. One disadvantage of kernel densities is the high runtime complexity. Compared to the naive Bayes and mixture model approach, kernel densities have the highest computational cost for classifying an object.

6.4 Nearest neighbor classifiers

Another nonparametric classification method is the nearest neighbor approach. Instead of building a density model for the training data prior to the actual classification process nearest neighbor approaches directly classify objects based on the training data only. Thus, prior to classification, no explicit model has to be constructed, but access to the entire original data is necessary. Since no training phase is needed nearest neighbor classification is also called "lazy" classification. Even though no explicit training phase is needed the efficiency of nearest neighbor classifiers can be improved by storing the data in a multi-dimensional index structure like the R*-tree [AKKS99].

Nearest neighbor approaches rely on a distance function d (e.g. Euclidean distance) to determine the nearest neighbors for an object $\mathbf{x}$. The number of objects to be considered is specified by a parameter k (e.g. $k = 1$). A

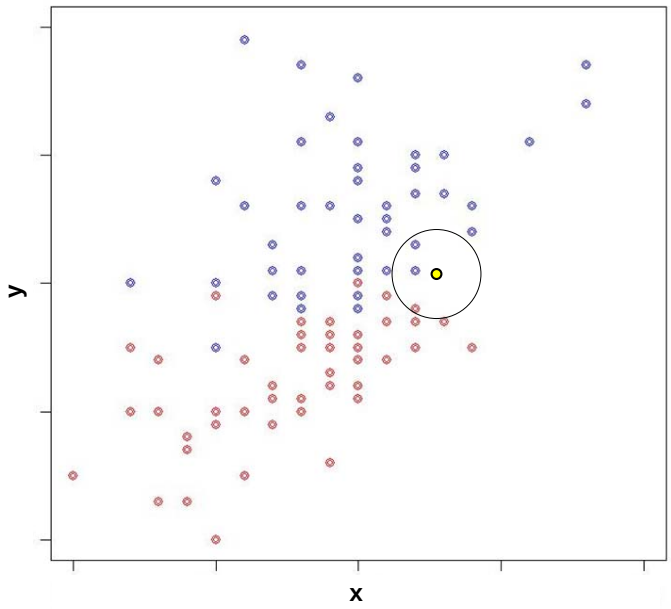

Figure 6.4: Nearest neighbor classification based for two classes (blue and red) using Euclidean distance

voting function (e.g. majority voting) is finally used to determine the class label from the nearest neighbors.

Definition 6.8 ***Nearest neighbor classifier.***

A nearest neighbor classifier for a given distance function $d(\mathbf{x}, \mathbf{o})$*, a parameter* k *and a voting function vote is defined as:*

$$\mathcal{G}_{NN} \quad = \quad vote\{NN_k(\mathbf{x})\} \text{ with}$$

$NN_k(\mathbf{x}) \subseteq \mathbf{D}$, $|NN_k(\mathbf{x})| \geq k$, NN_k *minimal and*
$\forall \mathbf{o} \in NN_k(\mathbf{x}), \forall \mathbf{o}' \in (\mathbf{D} \setminus NN_k(\mathbf{x})) : \; d(\mathbf{x}, \mathbf{o}) \leq d(\mathbf{x}, \mathbf{o}')$

Figure 6.4 illustrates a nearest neighbor approach for the example data set containing two classes (blue and red). For the yellow marked object the 3 nearest neighbors are evaluated using the Euclidean distance. As depicted the nearest neighbor set NN_3 contains two blue and one red object. Using majority voting would classify the yellow object as blue.

Chapter 7

Anytime stream classification using the Bayes tree

Anytime classification has gained importance with ubiquitous streams in many mobile and monitoring applications. Under greatly varying time constraints of apriori unknown stream inter-arrival rates, anytime algorithms provide the best classification up to a point of interruption dictated through the arrival of the next stream element. Traditional Bayes classification on kernel density estimators is known to provide good results, yet cannot be interrupted in a meaningful manner.

To enable anytime interruptions, we propose a novel hierarchy of density estimators. The density estimators are stored in our new *Bayes tree* for fast access and adaptable refinement until interruption. Storing the coarsest density model in the root node, our Bayes tree allows for direct classification. We exploit the levels of granularity in our novel hierarchy by successively refining an initial Bayes classification on coarse density estimates as long as time permits. If not interrupted, ultimately the finest level is reached on which the kernel densities are used for classification. Additionally, our approach prioritizes refinement of those parts of the model that provide most information to each probability density query. Thorough experiments on synthetic and real world databases demonstrate that our anytime probability density queries on the Bayes tree outperforms existing anytime classifiers.

7.1 Introduction

Stream data is ubiquitous in applications ranging from sensor data in monitoring systems to speech recognition. Stream mining aims at knowledge extraction from stream data, i.e. ordered sequences of data objects arriving continuously as time proceeds. The enormous amounts of data arriving in streaming applications can neither be stored prior to classification nor can classification of one item take longer than the time until the next item arrives. Anytime classifiers are capable of dealing with these varying time constraints and high data volumes [Zil96, YWKT07, UXKL06]. The streaming rate may vary, thus the time for computation of class labels may vary greatly. As it is desirable to achieve the best possible results w.r.t. the given time, classifiers should flexibly exploit all available time to improve their results.

Further more, in monitoring systems, immediate reaction to specific events for emergency detection is necessary. Likewise, interactive voice response in telephony implies direct recognition of incoming speech signals. These tasks thus call for classification, i.e. assigning of incoming data objects to a class label under time constraints.

In a recent project we worked on a health application to remotely monitor medical patients via body sensor networks [KDS+08]. The amount and frequency of data sent by the patients depends on their condition as preclassified locally on the body sensor controller. The receiving server has to classify all incoming patient data as accurate as possible. With many patients potentially sending aggregated or detailed data, the amount of data and their arrival rate may vary widely. Hence, classification for each individual observation must be done quickly and improve as long as time permits.

Besides stream mining, other classification tasks require interactive response times. *Anytime classification* allows for interruption of the classifier at any given time point to provide such interactive responses or online classification of stream data.

Anytime classifiers have to be capable of dealing with these time constraints and data volumes [Zil96, YWKT07, UXKL06]. Anytime classification applications have several characteristics in common. First, typically large volumes of data need to be processed. Second, there is a varying flow of more data. Third, immediate reaction is required. And finally, in many applications the time allowance is not known in advance, but depends on varying stream volume or user interruption.

In contrast to anytime algorithms, so called *budget* or *contract* algorithms [Zil96] use a fixed (*budgeted*) time limit for their calculation. If the available time is less than the contracted budget, they cannot give any result. Another severe limitation is their inability to improve the result if more time is available. Their only chance is to restart computation with a more detailed model. This exhibits the major drawback, since they cannot use the results they had so far, but have to start computation from scratch.

A good anytime classifier makes best use of the time allowance for classification. It can be interrupted at any time and still provide a good classification estimate until then. This can be summarized in three quality criteria: *efficiency*, *effectiveness*, and *interruptibility* of the classifier.

In terms of classification effectiveness, Bayes classifiers using kernel estimates have shown to perform well in traditional classification tasks. They provide a detailed model of the data on sound statistical foundations. In terms of interruptibility or efficiency, however, this detailed model is disadvantageous, since classification is based on a computation that includes all kernel estimates.

For anytime classification, we propose a *hierarchy* of mixture densities that includes kernel densities as the most detailed level and successively coarser mixture densities on upper levels. This organization of models allows interruption at any level of the hierarchy, using as much detail as was read until the point of interruption.

Kernel estimates are based on the actual data objects, and, therefore, they are best organized by means of any multidimensional indexing structure like the R-tree or R*-tree [Gut84, BKSS90]. They have shown to provide substantial efficiency gains in a number of applications on both point and spatial data. For probability density queries required for Bayes kernel classification, all kernels have to be examined. As minimum bounding rectangles in multidimensional indexes provide no information for classification, we propose storing mixture density parameter information in directory nodes to allow for anytime classification.

Our novel Bayes tree indexing structure provides model information about the kernels in the subtree of any node at any hierarchy level of the tree. It builds nodes in a bottom-up fashion that reflect density models in a compact way at different granularities. For our new *probability density query*, descending the tree is based on strategies that favor classification accuracy. These queries require aggregation of mixture densities at different granularities as

opposed to localizing points or point neighborhoods.

Our tree provides a very coarse model in the root node, and successively finer models on the lower levels of the tree. As different granularities are available and compact models are located at the top levels, the probability density query can be interrupted early on. As long as time permits, the model is further refined. Our Bayes tree is an efficient indexing strategy for the established effective Bayes classifier.

Our approach includes

- statistically sound mixture density classification
- hierarchy of mixture density models
- index-based anytime Bayes classification
- anytime probability density queries

This chapter is structured as follows: we review related work on anytime algorithms, classification and indexing in Section 7.2. Anytime classification prerequisites and quality criteria are discussed in Section 7.3. Our novel Bayes tree indexing structure is defined in Section 7.4.2 along with different descent strategies and model aggregation. Our thorough experiments in Section 7.5 demonstrate on both synthetic and real world data that Bayes tree classification is more efficient and effective than existing techniques. We conclude this chapter in Section 7.6.

7.2 Related work

Anytime algorithms are used in robotics and other artificial intelligence systems where interruption is common [KS02]. In anytime learning, the focus is on restrictions of the training phase [GR92, LMG03]. Other approaches study monitoring of the performance of anytime algorithms [HZ96], or anytime computation of top-k queries [ADGK07].

Anytime classification is classification up to a point of interruption [MKSW00, EM05, UXKL06]. In addition to high classification accuracy as in traditional classifiers, anytime classifiers have to be highly efficient to make best use of the limited time available, and, most notably, they have to be interruptible at any given time point. This time point is usually not known

in advance and may vary greatly [Zil96]. Anytime classification has been discussed for boosting techniques [MKSW00], decision trees [EM05], and nearest neighbors [UXKL06]. For Bayesian networks, a sampling approach for candidate selection has been suggested [HD02]. For naive Bayes, an anytime algorithm that takes some dependencies into account using a weaker attribute independence assumption is discussed in [YWKT07]. For Bayes classifiers based on kernel densities that do not make the strong independence assumption of naive Bayes no anytime algorithm exists to the best of our knowledge.

Multidimensional indexing structures like the R-tree, R*-tree, X-tree or Gauss-tree have been shown to provide substantial efficiency gains in similarity search [Gut84, BKSS90, BKK96, BPS06]. The basic idea is to organize the data such that only relevant parts of the database have to be accessed. This is based on the assumption that in traditional similarity search or retrieval scenarios, query processing requires only a small portion of the data. This assumption does not hold in Bayes kernel classification. As the entire model may potentially contribute to the class label decision, probability density queries are fundamentally different from similarity queries in that the entire database has to be accessed in order to answer any query without loss of accuracy (see Section 7.4 for details). Thus, approaches like the Gauss-tree that store univariate probabilistic features for similarity search or the R*-tree that store point or spatial data cannot provide the necessary density estimation for Bayes classification. Consequently, simply storing kernel estimates in multidimensional indexes, does not suffice for anytime classification. Thus, a meaningful hierarchy of model information is required that allows classification based on partial information from the index. Consequently, indexes have to provide information on the hierarchy of the models as discussed in Section 7.4.

7.3 Anytime stream classification

In stream mining or other online mining tasks the incoming data flow exceeds both the time and space limits of traditional algorithms. As large volumes of data have to be processed while novel data is constantly arriving, *anytime algorithms* have to provide good results in an efficient manner. Moreover, these algorithms have to keep pace with constantly incoming data. As the

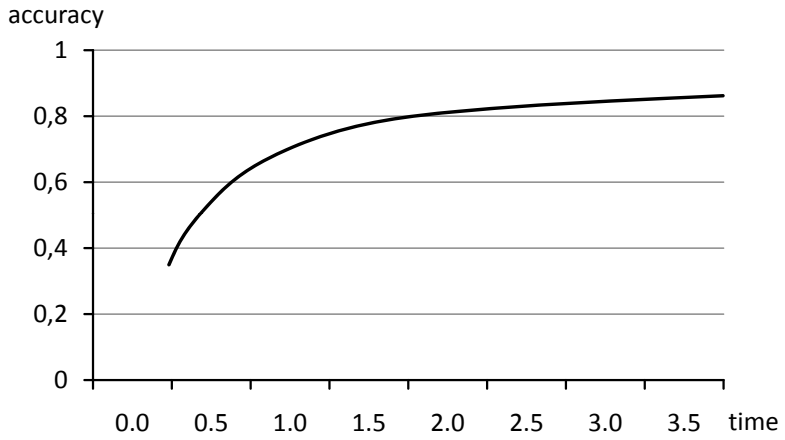

Figure 7.1: Prototypical anytime classification accuracy: very small time allowances typically lead to medium classification accuracy, greater time allowances provide a rapid accuracy increase.

arrival rate may vary, *interruptibility* at any time point is a key requirement. This is opposed to *contract algorithms* where time allowance is known in advance [Zil96]. Besides stream mining, any application with interactive response times requires anytime algorithms.

For classification, *anytime classifiers* are interruptible classifiers that provide the best *quality* for given time or page access constraints. For example, in stream monitoring, events have to be classified with respect to action required.

Accuracy in anytime classification denotes per time step the absolute classification accuracy as in traditional classifiers as well as the relative classification accuracy with respect to traditional unbound time classification. In stream classification the available computation time varies, therefore the accuracy of an anytime classifier is given by the average classification accuracy with respect to the inter-arrival rate (speed) of the data stream. We evaluate this stream specific anytime classification accuracy in our experiments in Section 7.5.

Obviously, the usefulness of an anytime classifier depends on its quality. Quality can be determined with respect to *efficiency*, *effectiveness*, and the *accuracy increase*. Efficiency refers to the speed at which classification decisions are made. Effectiveness denotes the absolute classification accuracy as in traditional classifiers as well as the relative classification accuracy with re-

spect to traditional infinite time classification. Increasing quality means that anytime classifiers should show rapid improvement of accuracy. Ideally, even for little time, classification accuracy is close to the best answer. With more time, classification accuracy should soon reach the classification accuracy of the infinite time classifier [Zil96, UXKL06]. This is illustrated in Figure 7.1. Early on, the ideal anytime classifier should provide high accuracy with rapid increase. These requirements can be summarized as:

Anytime classifier quality criteria.

- **efficiency**, i.e. low runtime, is more important than in most traditional classification scenarios as the overall time allowance is limited
- **effectiveness**, i.e. high classification accuracy, as in traditional classification refers to the quality of the resulting class label assignment
- **accuracy increase**, i.e. the greater the time allowance, the better classification accuracy should be

In this chapter we propose a new anytime classifier based on both Gaussian kernels and Gaussian mixture models (see also Chapter 6.3). We develop a consistent model hierarchy which allows for easy mixing of models and kernels. We store this model hierarchy in our new index structure the Bayes tree. Details on updating of the model in our anytime classification scheme are discussed in the next section.

7.4 The Bayes tree

We are now ready to propose our novel anytime approach based on the Bayes decision theory using kernel density estimation. Our overall goal is an efficient algorithm that can be interrupted at any given time point and produces a meaningful classification result that improves with additional time allowances. The main problem of anytime Bayes classifier is to obtain a good approximation of the class conditional density $p(\mathbf{x}|c_i)$ in reasonable time. A Naive Bayes classifier or a mixture density approach with few components is a very efficient way to determine the class conditional density $p(\mathbf{x}|c_i)$ (see also Chapter 6.3). However, an efficient refinement of the density model is not possible.

In the next section we give an overview of our technique for estimating the class conditional density before giving the technical details in the following sections. Finally we describe how the classification decision is made using our novel index structure.

7.4.1 Outline

As discussed in Section 7.3, any anytime classifier has to meet three quality criteria: it should be efficiently computable to ensure that the time allowance is made good use of, it should be effective to ensure good classification accuracy, and finally, this accuracy should increase as time proceeds.

Effectiveness. Bayes classification using kernel densities is an effective choice. However, it cannot be interrupted at will, as meaningful classification is only possible once all density estimators have been read (see Chapter 6.3.3). Consequently, it cannot be used in anytime classification scenarios as is.

Efficiency. Indexing provides means for efficiency in traditional query processing. Grouping similar data on hard disk and providing directory information on disk page entries, only the relevant parts of the data have to be accessed during query processing. Traditional queries are usually specified via a query object and a similarity tolerance, given by a threshold range ε or by a number of k nearest neighbors. Using this ε range or information on the neighbors detected so far, irrelevant parts of the data may be pruned during query processing based on directory information. Thus the amount of data that has to be accessed is reduced, which in turn may greatly reduce runtimes. This is illustrated in Figure 7.2. The query and the ε range allow for straightforward pruning of database objects whose directory entries are at more than ε distance.

However, this traditional pruning is infeasible when dealing with kernel densities. The data objects could be stored at leaf level as in range query applications. As classification requires reading all kernel estimators of the entire model, accuracy would be lost if kernel densities were ignored. Consequently, there is no irrelevant data, and hence no pruning. In Figure 7.2 this scenario is illustrated. As the entire model is required for classification, no pruning is possible. As illustrated, the upper levels of an index that supports anytime classification would have to provide information that can be used for assigning class labels even before the leaf level containing the kernels is reached.

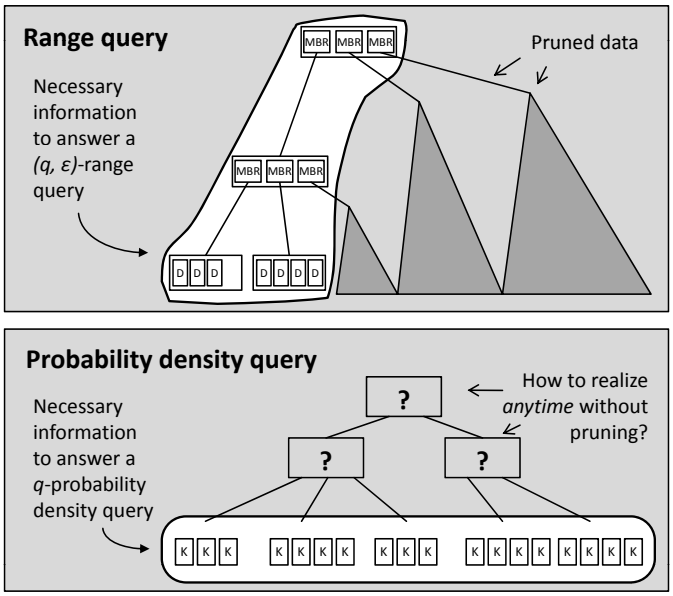

Figure 7.2: Traditional pruning is not possible for probability density queries, since all kernels have to be accessed to answer a q-probability density query.

Thus, accessing the entire kernel density model is clearly not efficient and not interruptible. Moreover, kernel densities provide only a single model of the data, i.e. no incremental improvement of classification is possible as required for anytime classification.

Our goal is therefore a hierarchy of classifiers built on top of kernel densities. Each hierarchy level should preserve as much information as possible on the underlying observations, while allowing interruption and query-based refinement as long as time permits.

Mixture densities naturally provide a way to summarize kernel density information. A set of kernel densities can be represented as mixture densities on a coarser level, i.e. a set of Gaussians. These Gaussians in turn, may be represented at an even coarser level using fewer Gaussians, and so on. The top level describes the entire data as a single Gaussian. These levels are illustrated in Figure 7.3.

Moreover, the information on different granularity levels should be indexable for greater efficiency, and should provide the means for refinement of the query at runtime as long as time permits and in the most relevant parts of the model depending on the query.

Thus, our Bayes tree approach comprises:

- **Bayes classification using kernel densities** at the finest level if time permits reading all kernel density estimators
- a **hierarchy** of mixture densities that allows for classification using the currently read model
- **indexing** of mixture densities for efficient representation and page-based access
- **refinement** of the most relevant parts of the model depending on the individual query

7.4.2 Structure of the Bayes tree

In this section we describe how to obtain a good approximation of the class conditional probability density-based on kernel densities for one given class c_i. Recall that kernel densities are based on the objects of the training data belonging to a specific class. We store the objects belonging to one class in one Bayes tree. The Bayes decision rule on multiple trees is discussed in Section 7.4.4.

Figure 7.4 shows the general idea: a hierarchy of mixture densities is stored in a multidimensional index. Each level of the tree stores a complete model of the entire data at different levels of granularity. To this end, a

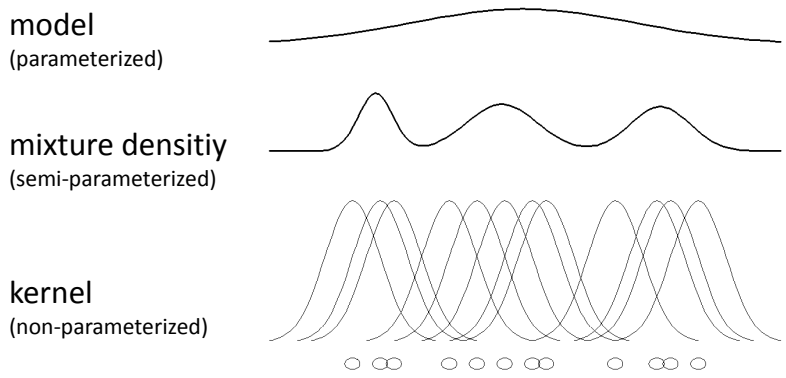

Figure 7.3: Different density estimation approaches: model-based assuming a single model (parametric), mixture densities using a combination of models (semi-parametric), and kernels adapting to the data (non-parametric)

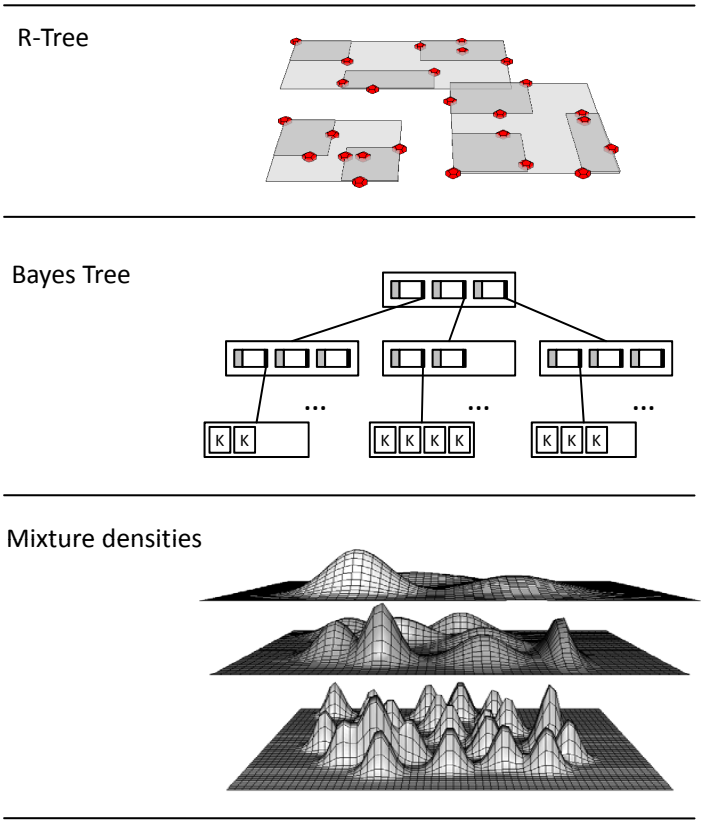

Figure 7.4: R-tree, Bayes tree and mixture densities on three Bayes tree levels: the R-tree encloses the objects by a hierarchy of minimum bounding rectangles (left), the Bayes tree builds a hierarchy over kernel density estimates at the leaf level (center) that correspond to the depicted mixture densities per level (right).

balanced structure as in R-trees is used to store the kernels at leaf level. The hierarchy on top is built in a bottom-up fashion, providing a hierarchy of node entries, each of which is a Gaussian that represents the entire subtree below it. The index is dynamic and easily adapts to novel data as not the actual parameters of the Gaussian, its mean and its variance, are stored, but easily maintainable information for their computation.

Multidimensional indexing structures from the R-tree family summarize data recursively in minimum bounding rectangles (MBRs). Leafs store the actual data objects and the subsequently higher levels provide directory in-

formation via the lower and upper bounds of these MBRs. This approach has been successfully used in a number of applications involving point or spatial data objects.

For kernel densities storing the actual data objects is sufficient. To applicate the kernel density model on the actual objects requires setting the smoothing parameter h (also called bandwidth - see also Chapter 6). We use a common method proposed in [JL95], setting $h_i = 1/\sqrt{|\mathbf{D}_{c_i}|}$. This approach has the advantage of allowing simple updates of our models in our Bayes tree at low computational cost. Choosing the type of kernel itself depends on the application requirements. In principle, our approach works for arbitrary kernels. Throughout this work, we instantiate our approach by using the Gaussian kernel $\mathbf{K}_{Gauss}(w) = \frac{1}{(2\cdot\pi)^{d/2}} e^{-\frac{w^2}{2h_i}}$ which is a very common choice for mixture densities (see Chapter 6.3.2).

However, storing MBRs information on the upper levels in not sufficient to determine the probability density distribution of subtrees. Consequently, we propose storing more information than MBRs to provide the information necessary to compute parameters of the mixture densities at any given level of the hierarchy.

Naively, we could store the parameters of all Gaussians at the respective level of the tree. For the first issue, recall that the parameters of mixture densities are the mean and variance of Gaussians (w_i, μ_i and Cov_i, respectively). That is we could store these parameters of a subtree at the corresponding node (see also Section 6.3.2). Please note if only variances $(\sigma_1, \ldots, \sigma_d)$, but no covariances are considered, the covariance matrix Cov degenerates to a diagonal matrix and $det(Cov) = \prod_{i=1}^{d} \sigma_i^2$.

Storing this information directly would require accessing the actual objects on leaf level to generate any model on a higher level. For example, assume we have two mixture densities. Building a coarser mixture density that represents both is not possible without additional information. The mean and variance are not distributive, i.e. from two means alone we cannot infer the overall mean (see also Figure 7.5). Instead, the number of instances is required to weight the two means accordingly. Mean and variance are thus algebraic measures that require the number of instances, their linear sum and quadratic sum.

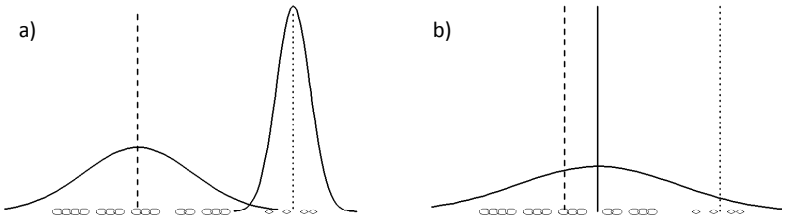

Figure 7.5: Computing mean and variance: computation of the overall mean in b) takes the number of objects (depicted as dots at the bottom) into account to weight the two means in a).

Definition 7.1 ***Bayes tree node entry.***
A subtree $\mathbf{T_s}$ *of a d-dimensional Bayes tree is associated with the set of objects stored in the leafs of the subtree:* $\mathbf{T_s} = \{\mathbf{t}_{(s,1)}, \dots \mathbf{t}_{(s,n_s)}\}$. *An entry* e_s *then stores the following information about the subtree* $\mathbf{T_s}$*:*

- *The* ***minimum bounding rectangle*** *enclosing the objects stored in the subtree* $\mathbf{T_s}$ *as* $MBR_s = ((l_1, u_1), \dots, (l_d, u_d))$
- *A* ***pointer*** ptr_s *to the subtree* $\mathbf{T_s}$
- *The* ***number*** n_s *of objects in* $\mathbf{T_s}$
- *The* ***linear sum*** $\sum_{j=1}^{n_s} \left(\mathbf{t}_{(s,j)}\right)$ *of all objects in* $\mathbf{T_s}$
- *The* ***quadratic sum*** $\sum_{j=1}^{n_s} \left(\mathbf{t}_{(s,j)}{}^2\right)$ *of all objects in* $\mathbf{T_s}$

Please note that all objects stored in the Bayes tree are d-dimensional kernels. A normalization to the unit hypercube $[0 \dots 1]^d$ *can be achieved by using the minimum bounding rectangle enclosing the root.*

Figure 7.6 illustrates the structure of a Bayes tree entry. In the following we also use the word *entry* for a node entry according to Definition 7.1. From the information stored according to Definition 7.1, mean and variance are computed as follows [HTF07]:

Lemma 7.1 ***Computing mean and variance.***

The mean point μ_s and the variance vector σ_s^2 for a subtree $\mathbf{T_s}$ can be computed by the stored values of the respective entry e_s with:

$$\begin{aligned}\mu_s &= \frac{1}{n_s}\sum_{j=1}^{n_s}\left(\mathbf{t}_{(s,j)}\right)\\ \sigma_s^2 &= \frac{1}{n_s}\sum_{j=1}^{n_s}\left(\mathbf{t}_{(s,j)}{}^2\right) - \left(\frac{1}{n_s}\sum_{j=1}^{n_s}\left(\mathbf{t}_{(s,j)}\right)\right)^2\end{aligned}$$

We therefore extend multidimensional indexing from the R-tree family to store model specific information. Our Bayes tree extends the R*-tree in the following manner:

Definition 7.2 ***Bayes tree.***

A Bayes tree with fanout parameters m, M and leaf node capacity parameters l, L is a balanced multidimensional indexing structure with the following properties:

- *Each inner node $node_s$ contains between m and M entries (see Definition 7.1). The root has at least 1 entry.*
- *Each inner node with ν_s entries has exactly ν_s child nodes (see Figure 7.6: $node_s$ has two entries $e_{s\circ 1}$ and $e_{s\circ 2}$ with their child nodes $node_{s\circ 1}$ and $node_{s\circ 2}$).*
- *Leaf nodes store between l and L observations (d-dimensional objects).*
- *A path from the root to any leaf nodes has always the same length (balanced).*

This structure has some very nice and intuitive benefits: since the number of entries, their linear sum as well as their quadratic sum are all distributive measures, the information stored in any node of our Bayes tree may be used to build the same, yet more coarser level, information in nodes on higher levels. More precisely, we may simply use the built procedure of any standard R*-tree to create our mixture densities.

Recall that R*-trees, or any other tree from the B-tree or R-tree family, grows in a bottom-up fashion. Whenever there are more entries assigned to

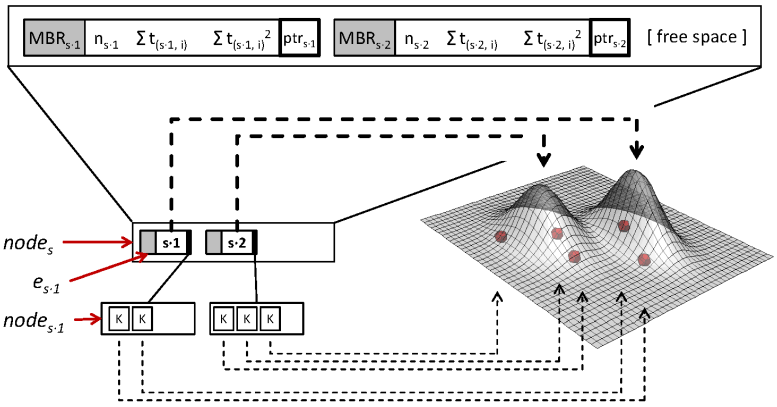

Figure 7.6: Nodes and Entries in the Bayes tree (here: above the kernel estimate level). Entries store minimum bounding rectangles (MBR) and additionally information to compute mean and variance, i.e. number of objects (n_s), linear sum ($\sum x_i$), quadratic sum ($\sum x_i^2$), and child pointers (p_i). Kernel positions are depicted as dots, higher level mixture density representation as density curves (bottom right).

any node than allowed by the fanout parameters M, which in turn reflects the page size on hard disk, an overflow occurs. This overflow leads to a node split, i.e. the entries of this overfull node are separated onto two new nodes. Their ancestor node then stores aggregate information on these two nodes. In the R*-tree case, this was simply the MBR of the two nodes, for the Bayes tree additionally, the overall number, linear and quadratic sum have to be computed and stored in the ancestor node. This way, in a very straightforward manner, coarser mixture density models are created.

Consequently, building Bayes-trees is a simple procedure that starts from the original kernel density estimates that are stored at leaf level until a split is necessary. This first split leads to creation of a second level in the tree, with a coarser mixture density model derived through R*-tree-style split. Successive splitting of nodes as more kernel densities are inserted, leads to additional growth of the tree, eventually yielding a hierarchy of mixture density models as coarser representations of the kernel densities (see Figure 7.7).

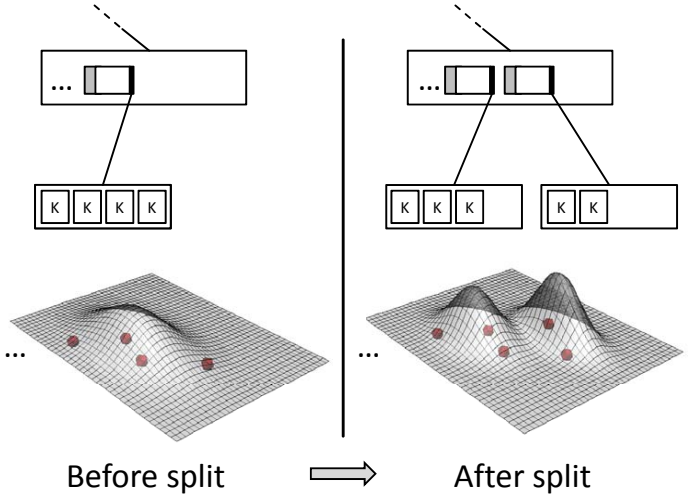

Figure 7.7: Split in the Bayes tree: to the left a completely filled leaf node with four kernel estimates is depicted. Additional kernel densities for new observations may be dynamically inserted, leading to a split of the leaf with automatic generation of new mixture density representations at higher levels.

7.4.3 Query processing

In the following we first elaborate strategies to answer probability density queries on a Bayes tree and then describe our approaches for classification refinement in Section 7.4.4.

Probability density queries

Recall that an entry e_s represents all objects in its corresponding subtree by storing the necessary information to calculate its mean and variance. Hence, a set $E = \{e_1 \dots e_k\}$ of entries defines a Gaussian mixture model p^{Gauss}_{Mixtue} according to Definition 6.6.

Definition 7.3 ***Probability density query pdq.***
Let $E = \{e_1 \dots e_k\}$ be a set of entries and $\mathbf{n} = \sum_{i=1}^{k} n_{e_i}$ the total number of object represented by E. A probability density query pdq returns the density for an object $\mathbf{x}$ using the Gaussian mixture model given by E:

$$pdq(\mathbf{x}|E) = \sum_{i=1}^{k} \frac{n_s}{\mathbf{n}} \cdot g(\mathbf{x}, \mu_{e_i}, \sigma_{e_i})$$

where μ_{e_i} and σ_{e_i} are calculated according to Lemma 7.1. For a leaf entry a kernel estimator as discussed in Section 6.3 is used and obviously μ_{e_i} is the object itself.

Before we can go into detail on our query processing strategies, we need a clear definition for the enumeration of nodes and entries.

Definition 7.4 ***Enumeration of nodes and entries.***
To address each entry stored in the Bayes tree we use a label s with the following properties:

- *The (virtual) entry pointing to the root node is labeled $e_\emptyset$ with $\mathbf{T}_\emptyset$ the complete set of objects stored in the Bayes tree.*
- *The child node of an entry e_s has the same label as its parent entry, i.e. the child of e_s is $node_s$. T_s denotes the set of objects stored in the respective subtree of e_s.*
- *The ν_s objects stored in $node_s$ are labeled $\{e_{s\circ 1} \dots e_{s\circ \nu_s}\}$, i.e. the label s of the predecessor concatenated with the entry's number in the node (recall Figure 7.6).*

A set of nodes which can be reached from the root is defined by a prefix-closed subset. The assumption of prefx-closed merely formalizes that a node can only be read if the node containing its parent entry has also been read.

Definition 7.5 ***Prefix-closed subset and frontier.***
A set of nodes $\mathcal{N}$ is prefix-closed iff:

- *$node_\emptyset \in \mathcal{N}$ (the root is always contained in a prefix-closed subset)*
- *$node_{s\circ i} \in \mathcal{N} \Rightarrow node_s \in \mathcal{N}$ (prefix-closed)*

Let $\mathcal{E}(\mathcal{N})$ be the set of entries stored on the nodes contained in a prefix-closed node set $\mathcal{N}$: $\mathcal{E}(\mathcal{N}) = \bigcup_{node \in \mathcal{N}} (e \in node)$. The frontier $\mathcal{F}(\mathcal{N}) \subseteq \mathcal{E}(\mathcal{N})$ contains all entries $e \in \mathcal{E}(\mathcal{N})$ that satisfy

- *$e_{s\circ i} \in \mathcal{E}(\mathcal{N}) \Rightarrow e_s \notin \mathcal{F}(\mathcal{N})$: all of its predecessors are not in $\mathcal{F}(\mathcal{N})$ (**predecessor-free**)*

- $e_s \in \mathcal{F}(\mathcal{N}) \Rightarrow \nexists i \in \mathbb{N}$ *with* $e_{s \circ i} \in \mathcal{F}(\mathcal{N})$*: all successors of a frontier entry are not in* $\mathcal{F}(\mathcal{N})$ *(**successor-free**)*

Figure 7.8 a) illustrates both a prefix-closed subset of nodes and the corresponding frontier. The subset, separated by the curved red line, contains the root, $node_2$ and $node_{23}$, hence it is prefix-closed. Its corresponding frontier (highlighted in white) contains the entries e_1, e_{21}, e_{23}, e_{221}, e_{222} and e_{223} where the last three represent kernels. Note that the frontier is both predecessor-free and successor-free.

Using the above definitions we now define the processing of a probability density query on the Bayes tree.

Definition 7.6 ***Anytime pdq processing.***
Let $T_\emptyset$ be the set of objects stored in a Bayes tree containing t_{max} nodes. Anytime pdq processing for an object $\mathbf{x}$ processes a prefix-closed subset $\mathcal{N}_t$ in each time step t with:

- $\mathcal{N}_0 = \{node_\emptyset\}$*: the processing starts with the root node*
- $|\mathcal{N}_t| + 1 = |\mathcal{N}_{t+1}|$*: in each time step one more node is read*

The probability density for the query object $\mathbf{x}$ at time step t is then calculated using the mixture model corresponding to the frontier $\mathcal{F}(\mathcal{N}_t)$ of the current prefix-closed subset $\mathcal{N}_t$ as in Definition 7.3, that is $pdq(\mathbf{x}|\mathcal{F}(\mathcal{N}_t))$.

Note that $\sum_{e_s \in \mathcal{F}(\mathcal{N}_t)} n_{e_s} = |T_\emptyset|$.

The costs for calculating the new probability density for x after reading one additional node is very low due to the algebraic nature of mean and variance. From time step t to $t+1$ $\mathcal{N}_t$ becomes $\mathcal{N}_{t+1}$ by adding the child node $node_s$ from one frontier entry $e_s \in \mathcal{F}(\mathcal{N}_t)$. If $node_s$ has ν_s entries, then the frontier $\mathcal{F}(\mathcal{N}_t)$ changes to $\mathcal{F}(\mathcal{N}_{t+1})$ by

$$\mathcal{F}(\mathcal{N}_{t+1}) = (\mathcal{F}(\mathcal{N}_t) \setminus \{e_s\}) \cup \{e_{s\circ 1}, \ldots, e_{s \circ \nu_s}\}$$

Hence, the probability density for x in time step $t+1$ can simply be calculated by

$$\begin{aligned} pdq(\mathbf{x}|\mathcal{F}(\mathcal{N}_{t+1})) &= pdq(\mathbf{x}|\mathcal{F}(\mathcal{N}_t)) \\ &\quad - \frac{n_s}{|T_\emptyset|} \cdot g(\mathbf{x}, \mu_{e_s}, \sigma_{e_s}) \\ &\quad + \sum_{i=1}^{\nu_s} \frac{n_{s \circ i}}{|T_\emptyset|} \cdot g(\mathbf{x}, \mu_{e_{s \circ i}}, \sigma_{e_{s \circ i}}) \end{aligned}$$

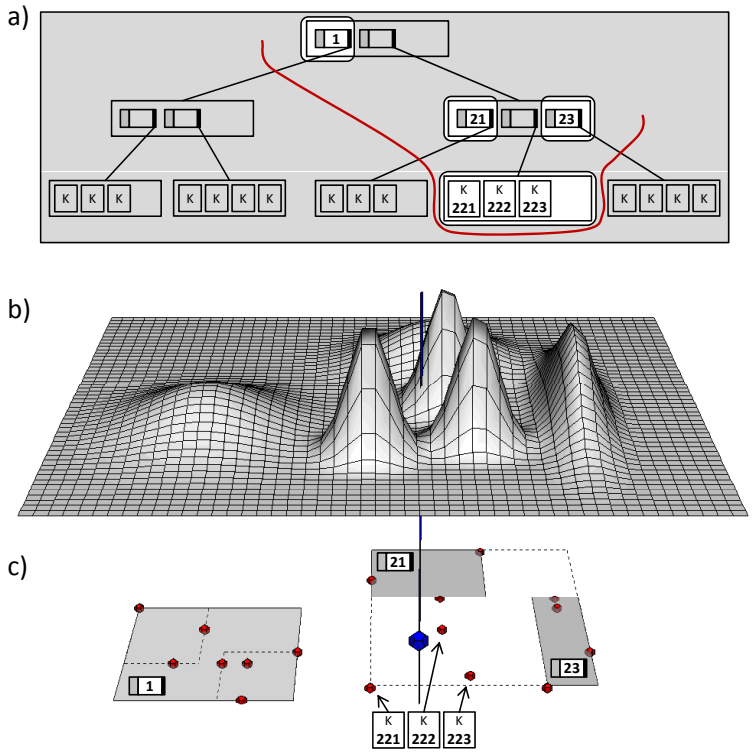

Figure 7.8: Tree frontier: a) depicts an exemplary frontier in a Bayes tree that corresponds to a model with mixed levels of granularity: nodes 2 (root level) and 22 (inner level) are refined, yielding 1 (root level), 21 (inner level), 221, 222, 223 (all leaf level), 23 (inner level). The resulting mixture density model is depicted in b), the actual kernel positions and the hierarchy are depicted in c).

Note that after reading all t_{max} nodes $pdq(\mathbf{x}|\mathcal{F}(\mathcal{N}_{t_{max}}))$ is a full kernel density estimation taking all kernels at leaf level into account.

Answering a probability density query requires a complete model as found at each level of the tree. Besides these full models, local refinement of the model to adapt flexibly to the query provides models composed of coarser and finer representations. More precisely, in any model, each node represents its subtree. This node representation may be replaced by the finer representation at any level below it. This idea leads to query-based refinement in our anytime algorithm. Each mixed granularity model corresponds to a frontier in the tree, i.e. a set of entries in the tree such that each kernel estimate is represented exactly once. Taking a complete subtree of all nodes starting

from the root, the frontier describes a non-redundant model that combines mixture densities at different levels of granularity.

Figure 7.8 b) shows the resulting mixture density for the example frontier from part a). The leftmost Gaussian stems from the entry e_1 which is located at root level. The rightmost Gaussian and the one in the back correspond to entries e_{23} and e_{21} respectively, the remaining represent kernel densities at leaf level. Part c) of the image depicts the underlying R*-tree MBRs and the kernels as dots. The bigger blue dot and the vertical line represent the query object from which the above frontier originated.

A frontier is thus a model representation that consists of node entries such that each kernel estimate contributes (prefix-closed subtree) and such that no kernel estimate is represented redundantly (predecessor-free). Each possible frontier represents a possible model for our anytime algorithm. The number of possible models in any Bayes tree depends on the actual usage of the tree, i.e. the degree to which it is filled and the height of the tree. Figure 7.9 illustrates the four possible models for a Bayes tree of height two and fanout two. For realistic trees, with increasing height and fanout, the number of possible models is enormous.

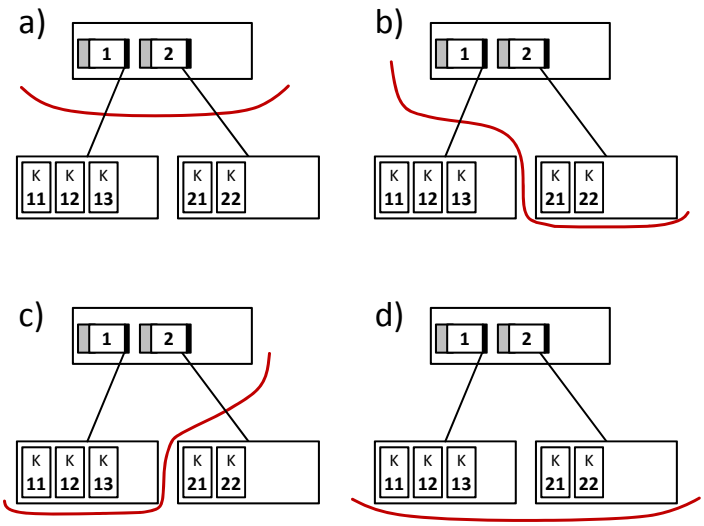

Figure 7.9: A tree of height 2 and fanout 2 yields four possible models: a) root level model, b) and c) mixed root and leaf level model, and d) leaf level model.

Choosing among all these models is crucial for anytime algorithms, where the time available (typically unknown a priori) should be spend such that the most promising model information is used first. For tree traversal we propose three different descent approaches to answer probability density queries on a Bayes tree. First, we could descend in a breadth-first (*bft*) fashion, refining each model level completely before descending down to the next level. Alternatively, we could descent the tree in a depth-first (*dft*) manner, refining a single subtree entirely down to the actual kernel estimates before refining the next subtree below the root. The choice of the subtree to refine is made according to a priority measure. The third approach, which we call *best first*, orders nodes globally with respect to a priority measure and refines nodes in this ordering.

As for the priority measures we use a *geometric* approach and a *probabilistic* approach. The geometric approach gives highest priority to the node with the smallest distance based on its minimum bounding rectangle as in traditional R-trees, that is $dist(\mathbf{x}, \text{MBR}) = \sqrt[p]{\sum_{i=1}^{d} dist(x_i, (l_i, u_i))^p}$ with

$$dist(x_i, (l_i, u_i)) = \begin{cases} x_i - u_i & \text{, if } x_i > u_i \\ l_i - x_i & \text{, if } x_i < l_i \\ 0 & \text{, otherwise} \end{cases}$$

where d is the dimensionality, $x = (x_1, \ldots, x_d)$ is the query object and $MBR = ((l_1, u_1), \ldots, (l_d, u_d))$ as defined in Definition 7.1. p stems from the L_p norm, where we use the Euclidean distance ($p = 2$) for our computations.

The probabilistic priority measure exploits the statistical information stored in each node and awards priority with respect to the actual density value for the current query object. More specifically, at time step t, having read the prefix-closed set of nodes $\mathcal{N}_t$, the probabilistic approach descends at the next time step $t + 1$ into the subtree $\mathbf{T}_{\hat{s}}$ belonging to the entry $e_{\hat{s}}$ with

$$e_{\hat{s}} = \underset{e_s \in \mathcal{F}(\mathcal{N}_t)}{argmax} \{g(\mathbf{x}, \mu_{e_s}, \sigma_{e_s})\}$$

where x is the current query object, μ_{e_s} and σ_{e_s} are the mean and variance of $\mathbf{T}_{\hat{s}}$ as derived from the information stored in e_s and g is a Gaussian probability density function as in Definition 6.6.

In our experiments we demonstrate the outcome of all combinations between the three descent strategies and the two priority measures, where the priority measure determines for the breadth-first approach the order of nodes per level.

7.4.4 Classification refinement

So far we described in the last section different strategies how to descend in one Bayes tree, i.e. how to improve the class conditional density for one class using increasingly finer mixture models. Now we still have to define refinement strategies for classification with respect to several classes, i.e. how to divide or spend the time given on descending several Bayes trees, one per class (see Section 7.4.2). For classification purpose optimally each tree should be refined. One question is having read m nodes so far, i.e. being at time step t_m, which of the n trees representing the different classed should be refined in the following time step t_{m+1}?

Initializing our classifier we evaluate the coarsest model for each class $c_i \in \mathbf{C}$, i.e. the model consisting of only one Gaussian probability density function representing all objects of c_i. For initialization we refer to these n Gaussians as the *virtual root*. Please recall that the model represented by the virtual root corresponds to the naive Bayes classifier 6.3.1.

For classification refinement we propose two different approaches. The first approach uses a simple strategy which refines the next node in turn to all trees in their natural order and start over again each time every tree was accounted for. We refer to this refinement strategy as *order*. The second refinement approach gives permission to read the next node to the Bayes tree whose class has the maximal probability for the current query at that time. More precisely, at time step t, the next node is read in the Bayes Tree of class $c_{\hat{i}}$ with

$$c_{\hat{i}} = \underset{c_i \in \mathcal{C}}{argmax}\{pdq(\mathbf{x}|\mathcal{F}(\mathcal{N}_{t,c_i}))\}$$

where $\mathbf{x}$ is the present query object, $\mathbf{C}$ the set of class labels and $pdq(\mathbf{x}|\mathcal{F}(\mathcal{N}_{t,c_i}))$ is the density value for $\mathbf{x}$ in the current mixture model for class c_i at time step t (recall Definition 6.6). We therefore refer to the second refinement approach as *maximal*.

Summary. The Bayes tree is a general indexing structure for answering probability density queries. By storing the number of objects, their linear and quadratic sums, respectively, it easily adapts to new observations by inserting the corresponding kernel estimates and updating the algebraic measures. Using the R*-tree split, the tree grows in a natural fashion and generates additional levels of granularity as the tree grows. Arbitrary prefix-closed cuts can be processed resulting in a predecessor- and successor-free frontier and its

corresponding mixture density, which allows the desired anytime behavior. Different descent strategies, priority measures and refinement approaches can be employed. In the next section we demonstrate the performance of probability density queries on the Bayes tree, analyzing the above mentioned strategies and their combinations in detail.

7.5 Experiments

We ran extensive experiments on real world and synthetic data to evaluate our anytime classifier. After each time step t we interrupted the Bayes tree classification and measured the classification accuracy. For secondary storage index structures a time step corresponds to a page access. Experiments were run using 2KB page sizes on Pentium 4 machines with 2.4 Ghz and 1 GB main memory.

7.5.1 Experimental setup

The performance of our anytime classifier is evaluated using both synthetic and real world data sets. We generated different synthetic data sets containing two classes. For each class, the data objects are randomly generated from a random Gaussian mixture model as in [HTF02]. Generation is in three steps. First we use one Gaussian distribution for class one with mean $\mu_1 = (0 \dots 0)$ and one Gaussian for class two with mean $\mu_2 = (v \dots v)$; $\|\mu_2 - \mu_1\| = \sqrt{d \cdot v^2} = 4.0$. Variance for each dimension was set to 4.0. In the second step we randomly generated $n_{mix} = 55$ means from each Gaussian distribution. These 55 mean values are used as mean values for the components of the mixture model and the variance is uniformly distributed between 0.1 to 1.0. Finally, we generated 100 objects for each Gaussian in the third step, thus in total the data set contains $5,500$ objects per class. We randomly picked $1,000$ objects for testing and $10,000$ for training. We varied the dimensionality between $d = 4$, $d = 8$ and $d = 12$ dimensions.

Further on we use continuous valued attributes from three different real world data sets (see Table 7.1). The USPS data set is first transformed and reduced to 10 dimensions using linear discriminant analysis [DHS01].

In all experiments we use a 4 fold cross validation and study the achieved classification accuracy for different time allowance. We compare the Bayes

Name	size	classes	features	reference
Vowel	990	11	10	[HB99]
USPS (LDA)	8,772	10	10	[HTF07]
Gender	189,961	2	9	[AS04]

Table 7.1: Data sets used in our experiments

tree with a recent anytime nearest neighbor classifier (Anytime NN) [UXKL06]. The Anytime NN approach sorts the data according to a discriminant function and always determines the nearest neighbor w.r.t. the data read to the point of interruption (see also Chapter 6.4 for details on the nearest neigbor classifier).

Both anytime classifiers need a short setup for evaluating the virtual root or for calculating the distance to one representative per class. After this initialization phase more and more details are used for classification. For both approaches we start with time point zero after this initialization step.

We additionally show the classification accuracy of the naive Bayes classifier as a base line comparison (see CHapter 6.3.1). The Naive Bayes classifier assumes independence of dimensions. As Naive Bayes only evaluates one Gaussian probability per class, it needs the same time for classifying an object as the setup phase of the anytime classifiers. Since it is not an anytime classifier, its classification accuracy is depicted as a flat line over time. Further on we report the classification accuracy of the Bayes classifier based on kernel densities as a second baseline of the maximal accuracy reachable by the Bayes tree.

7.5.2 Evaluating the strategy

First, we study the effects of the different query strategies on the classification accuracy using the 4-dimensional data set. Figure 7.10 (left part) presents the results using probabilistic descent strategies (Section 7.4.3). The *best first* strategy clearly outperforms the other two approaches (*depth* and *breadth-first*). The depth-first approach refines the same densities as the priority first approach for the first four time steps. After this the locality assumption of this approach fails to identify the most relevant refinements.

The breadth-first strategy achieves a similar classification accuracy as the

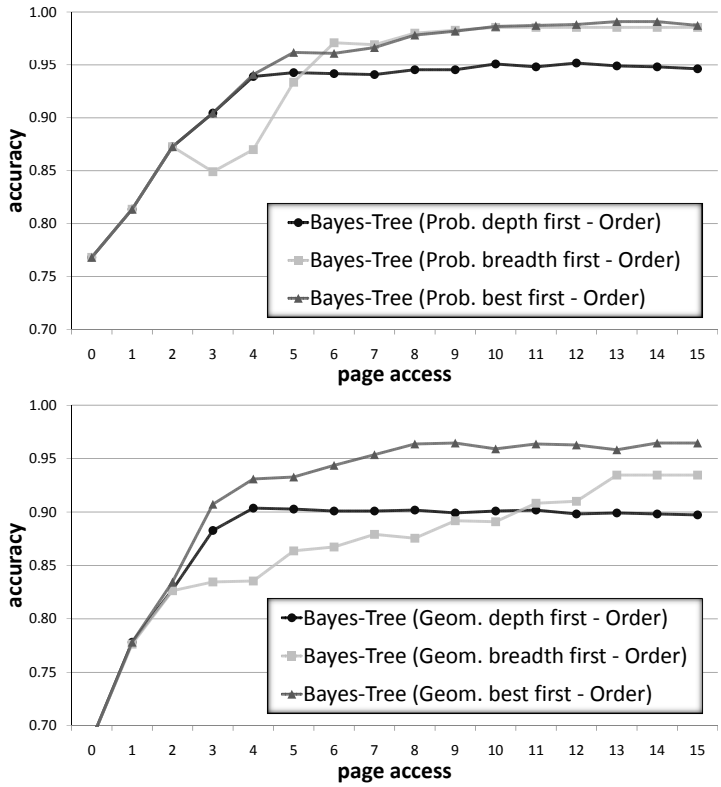

Figure 7.10: Probabilistic descent (top) and geometric descent (bottom) on synthetic data

other methods for the first two steps. After this its processing of complete levels before descending only slightly increases accuracy. At time point six the first level of the Bayes tree is completely read and the method continues with the next level. At time point four the component with highest probability at level 2 is refined which drastically increases breadth-first accuracy. The first levels of a tree do not contain many nodes and hence the breadth-first approach quickly achieves a better classification accuracy than the depth-first approach. The best first strategy globally selects the subtree with highest probability density for descent, resulting in better classification accuracy after a few steps only (time step four).

The same experiment with *geometric* descent (Section 7.4.3) again shows that the best first approach performs better than the other two methods (see Figure 7.10 - right part). The depth-first approach shows a lower overall performance compared to the probability depth-first approach (nearly 0.07).

In this experiment, the depth-first approach does not even achieve a better results if more time is available. This result indicates that the geometric depth-first approach does not choose a good subtree for refinement in the first steps. A similar result can be seen for the breadth-first approach. Compared with the probability breadth-first the geometric breadth-first approach shows slow improvement after a complete level of the tree has been read (time step three to four and seven to eight). Overall the probability descent strategies show a better performance than the geometric approaches. Similar results were obtained for other data sets. Thus, from now on, we focus on the probability best first strategy only.

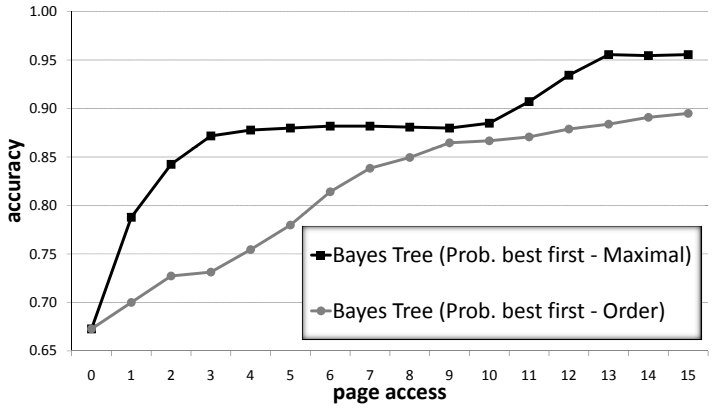

Figure 7.11: Refinement strategies *Order* vs. *Maximal*

Finally, we evaluate the effect of our classification refinement methods: *order* of the classes or *maximal* class conditional density. Recall that the order method refines each individual Bayes tree (each class) once before continuing with the first one again. In contrast the *maximal* method always refines the class with the highest posteriori probability and hence may refine one class multiple times. The vowel data set (see Table 7.1) results are illustrated in Figure 7.11. The maximal order clearly performs better than the arbitrary class order as classification accuracy increases dramatically at the beginning. For this data set the maximal order typically also refines each class in the first 11 steps. After this point, the classification accuracy for the maximal order again increases steeply and converges after 13 time steps. The class order does not converge until time step 22, i.e. after refining all

classes again. Thus, for the remaining experiments we use maximal order with probability best first traversal.

7.5.3 Evaluating the scalability and accuracy

In this section we evaluate the performance and scalability of the Bayes tree. Scalability in terms of dimensionality is studied on 4, 8 and 12-dimensional data sets in comparison with anytime nearest neighbor classification [UXKL06] (see Figure 7.12). As we can see, the Bayes tree clearly outperforms the Anytime NN approach on all three data sets. With increasing dimensionality the two classes are more easily separated and hence the overall classification accuracy increases. The Bayes tree scales very well and improvement of the classification accuracy is nearly optimal. At the beginning, the classification accuracy improves dramatically and converges after a few time steps only (reaching the classification accuracy of the kernel densities). Since the classification model of the Bayes tree at time point zero already uses a well defined model the Bayes tree starts with a high classification accuracy. The Anytime NN approach always starts with a lower accuracy and even looses quality in some cases if not interrupted.

In Figure 7.13 we present the results for all three real world data for different time allowance. As in the last experiment we compare our approach with the Anytime NN approach and additionally report the results of the Naive Bayes approach and the Bayes classifier based on Kernel densities.

On the real world data sets all methods perform very similarly. As depicted, the Bayes tree always starts with a high classification accuracy. With more time allowance the classification accuracy of the Bayes tree increases.

For the Vowel data set the accuracy of the Bayes tree reaches a high accuracy after 3 time steps only. As discussed in the last section the Bayes tree converges after 13 steps and nearly reaches the accuracy of Kernel densities at this point. The Anytime NN approach starts with a very low accuracy of 0.37. The good result of the Bayes tree is due to the fact that the coarse density model in the virtual root of the Bayes tree contains more information about the class distribution than individual objects. After 14 steps the Anytime NN also converges for the time being but does not reach the accuracy of the Bayes tree. Overall, the anytime kernel density model of the Bayes tree clearly outperforms the Anytime NN approach.

Next we evaluate the performance of the Bayes tree on LDA (linear dis-

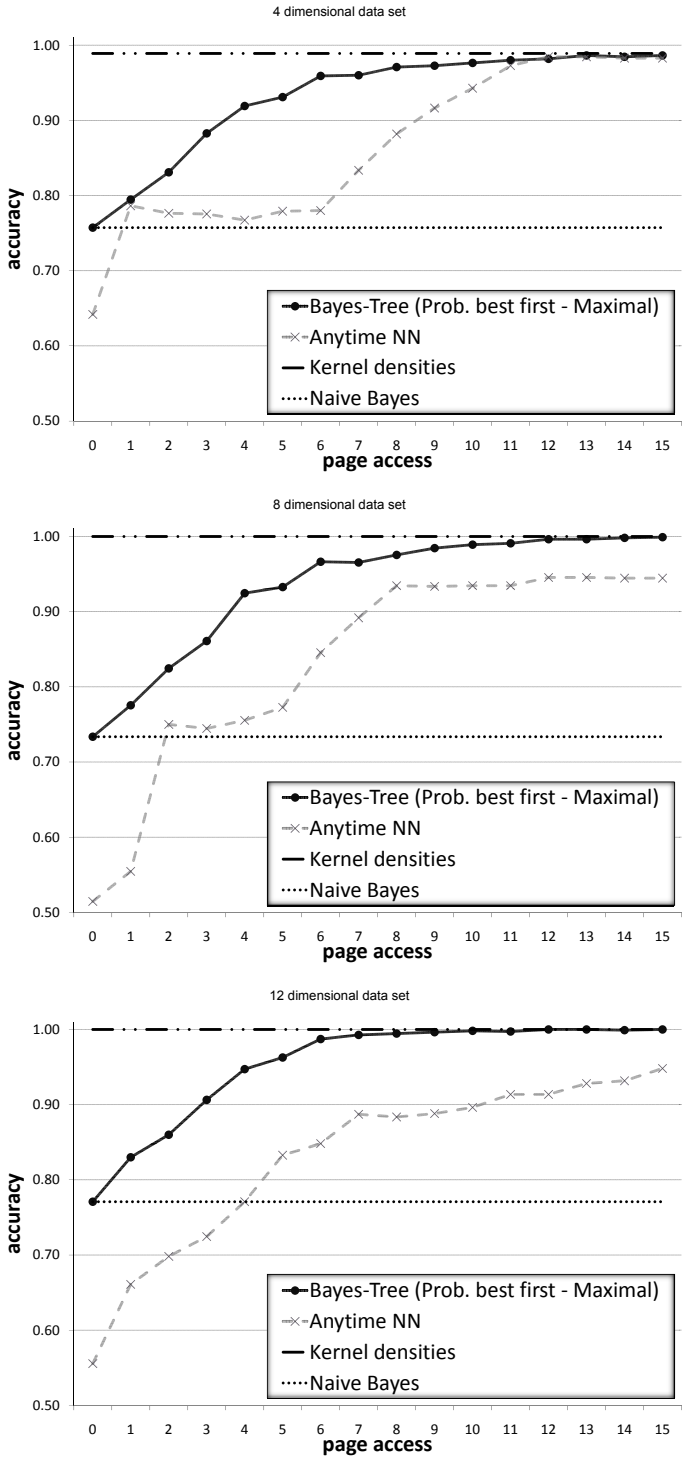

Figure 7.12: Comparing Bayes tree to Anytime NN on synthetic data of dimensionality 4,8 and 12 (top to bottom)

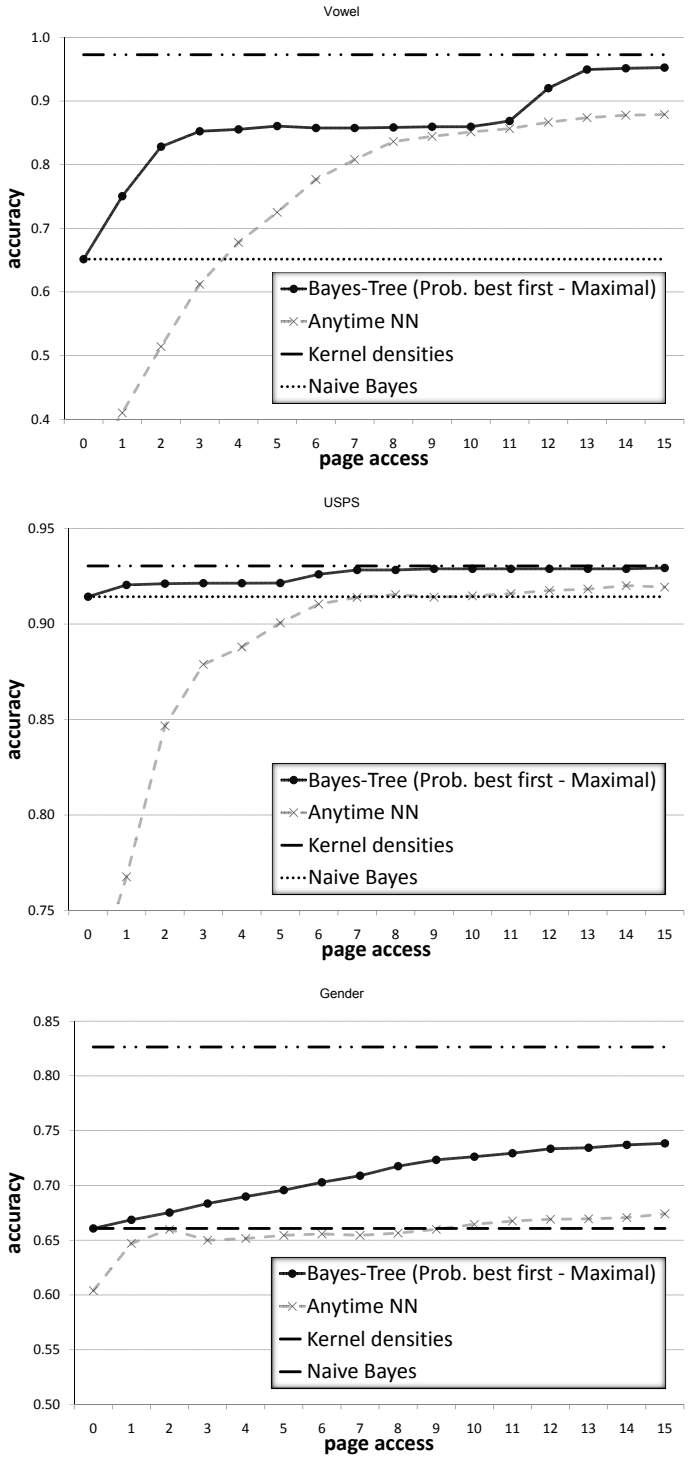

Figure 7.13: Comparing Bayes tree to Anytime NN on Vowel, USPS and Gender data set (from top to bottom)

criminant analysis) pre-processed data from machine learning [HTF07] (middle part of Figure 7.13). In this case the mixture model of the root is very close to convergence as classes are well separated through LDA analysis. Hence the Bayes tree starts with a classification accuracy of about 0.92. After a few time steps only the Bayes tree has reached the classification accuracy of the Kernel densities. For this data set the Anytime NN only reaches the performance of the Naive Bayes after 15 time steps.

For the last data set in Figure 7.13 the gender data set, the accuracy of the Anytime NN even shows non-monotonic behavior. The accuracy of the Bayes tree in contrast monotonically increases and clearly performs better than Anytime NN. The gender data set is the largest data set investigated by our experiments. As we can see if more objects are available the Bayes tree also needs more time for reaching the accuracy of the Kernel densities (after 15 time steps an improvement of 0.1 is still possible). Nevertheless the first steps show a great classification improvement.

7.5.4 Evaluating Poisson streams

To evaluate anytime classification under variable stream scenarios, we recapitulate a stochastic model that is widely used to model random arrivals. A Poisson process describes streams where the inter-arrivals are independently exponentially distributed. Poisson processes are parameterized by an arrival rate parameter λ:

Definition 7.7 ***Poisson stream.***

The probability density function for the inter-arrival time of a Poisson process is exponentially distributed with parameter λ:

$$p(t) = \lambda \cdot e^{-\lambda t}.$$

The expected inter-arrival time of an exponentially distributed random variable with parameter λ is $E[t] = \frac{1}{\lambda}$.

We use a Poisson distribution to model random stream arrivals for each fold of the cross validation. We randomly generate exponentially distributed inter-arrival times for different λ. If a new object arrives (the time between two objects has passed) we stop our anytime classifier and measure the classification accuracy. We repeat this experiment using different inter-arrival

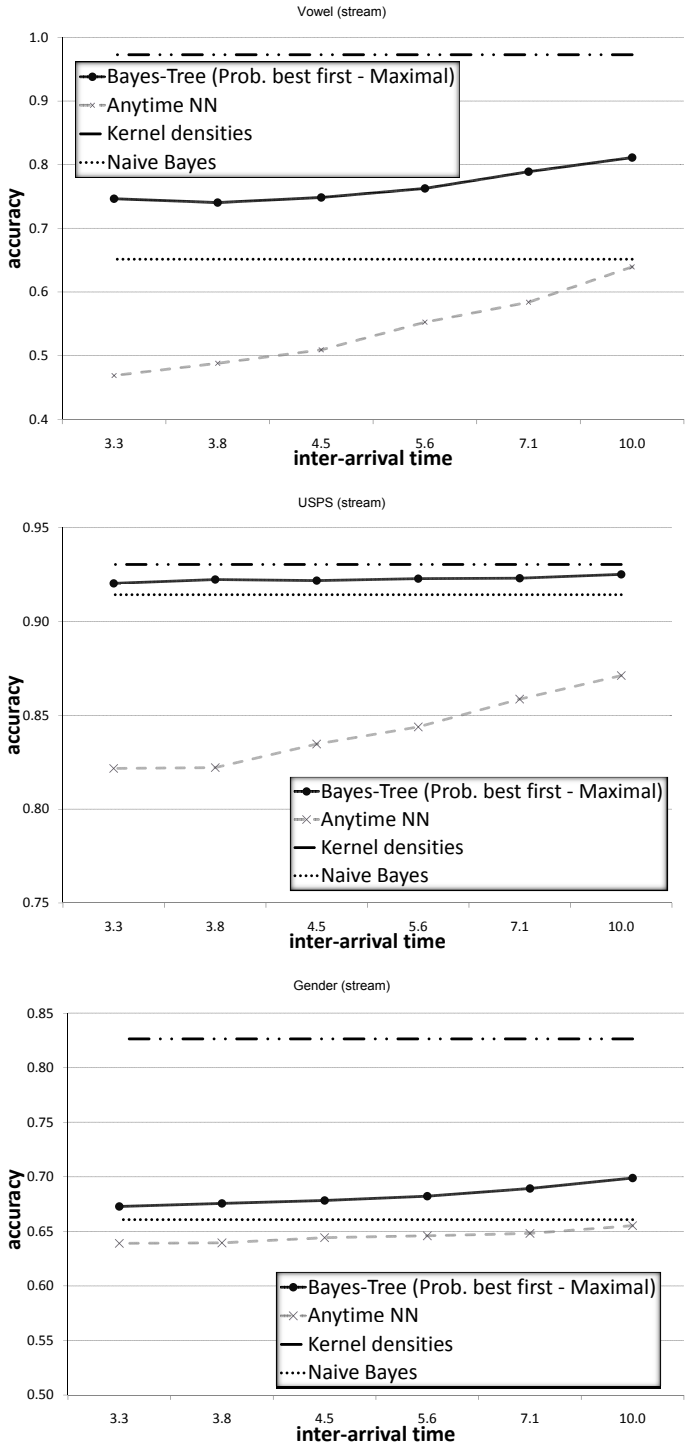

Figure 7.14: Comparing Bayes tree to Anytime NN using Poisson streams for Vowel, USPS and Gender data set (from top to bottom)

times $\frac{1}{\lambda}$. As the Bayes tree and the Anytime NN [UXKL06] both need an initialization phase prior to anytime classification, we assume that any object arrives after the initialization phase at the earliest.

The Naive Bayes classifier could also be applied to this stream scenario as Naive Bayes only needs a very small time budget for classifying an object (equal to the initialization phase of the anytime classifiers). As Naive Bayes cannot make use of additional time available the classification accuracy does not change for different inter-time arrivals. The performance of Kernel densities are again only reported as additional information as evaluating all Kernel densities for each object is not practical in stream scenario.

In Figure 7.14 we illustrate the results for different Poisson streams. We varied λ from 0.1 to 0.3 and hence achieved different inter-arrival times for the stream with an average of 10 to 3.3. As we can see if inter-arrival times increases the classification accuracy also increases for both anytime classifiers (the Bayes tree and Anytime NN).

For nearly all settings of λ the Anytime NN shows a worse performance than the Naive Bayes approach. The Bayes tree in contrast is capable of using the additional time to achieve a good average classification accuracy. The major reasons for the good results of the Bayes tree are the high initial accuracy and the rapid improvements of the accuracy during the first steps (compare Figure 7.13). These results indicate that the Bayes tree is a well designed anytime classifier for stream applications.

7.6 Conclusion

Anytime classification requires classifiers that can be interrupted at will and still deliver good classification results. In this work, we propose a novel index-based technique that allows the established Bayes classifier on effective kernel density estimation to be used in anytime classification. The Bayes tree automatically generates a data adaptable hierarchy of mixture densities that represent kernel density estimates at successively coarser levels. Our efficient probability density queries provide the necessary information for very effective classification at any point of interruption. Our extensive experiments on synthetic and real world data sets demonstrate that our anytime classification approach delivers high classification accuracy at any time.

Chapter 8

Classification using subspace clusters

Classification has been widely studied and successfully employed in various application domains. In multidimensional noisy settings, however, classification accuracy may be unsatisfactory. Locally irrelevant attributes often occlude class-relevant information. A global reduction to relevant attributes is often infeasible, as relevance of attributes is not necessarily a globally uniform property. In a current project with an airport scheduling software company, locally varying attributes in the data indicate whether flights will be on time, delayed or ahead of schedule. To detect locally relevant information, we propose combining classification with subspace clustering (*SubClass*). Subspace clustering aims at detecting clusters in arbitrary subspaces of the attributes. It has proved to work well in multidimensional and noisy domains. However, it does not utilize class label information and thus does not necessarily provide appropriate groupings for classification. We propose incorporating class label information into subspace search. As a result we obtain locally relevant attribute combinations for classification. We present the SubClass classifier that successfully exploits classifying subspace cluster information. Experiments on both synthetic and real world datasets demonstrate that classification accuracy is clearly improved for noisy multidimensional settings.

8.1 Introduction

Data produced in application domains like life sciences, meteorology, telecommunication, and multimedia entertainment is rapidly growing, increasing the demand for data mining techniques which help users generate knowledge from data. Many applications require incoming data to be classified according to models derived from labeled historic data. In a current project, we investigate flight delays for airport scheduling purposes. The significance of flight delays can e.g. be studied in reports of the Bureau of Transportation Statistics in the U.S. [Bur05] and the Central Office for Delay Analysis of Eurocontrol [Eur05]. Extensive flight data is recorded by flight information systems at all major airports. Using such databases, we classify flights as on time, delayed or ahead of schedule. This classification is essential in refining robust scheduling methods for airport resources and ground staff (like the one presented in [Bol00]).

For classification, numerous techniques exist. For our noisy database that contains nominal attributes, numerical classifiers are not applicable. Neural networks or support vector machines do not allow users to easily understand the decision model for flight classification [SdSA05, Pla98] (see also Chapter 6.1). Bayes classifiers, decision trees, and nearest neighbor classifiers provide explanatory information, yet assume globally uniform relevance of attributes [SdSA05, Qui92, AKA91]. It has been shown that each type of classifier has its merit; there is no inherent superiority of any classifier [DHS01].

However, classification is difficult in the presence of noise. Moreover, patterns may not show across all data attributes for all classes to be learned. In multidimensional data only a subgroup of attributes may be relevant for classification. This relevance is not globally uniform, but differs from class to class and from instance to instance.

We have validated the assumption of local relevance of attributes for the flight classification project by training several types of classifiers. When using only attributes which are determined as relevant by standard statistical tests, classification accuracy actually drops. This suggests that globally irrelevant attributes are nonetheless locally relevant for individual patterns. We therefore target at grouping flights with similar characteristics and identifying structure on the attribute level. In the flight domain, several aspects support the locality of flight delay structures. As an example, passenger figures may only influence departure delays when the aircraft is parked at

a remote stand, i.e. when bus transportation is required. At some times of the day, these effects may be superposed by other factors like runway congestion. Weather conditions and other influences not recorded in the data cause significant noise.

Recent classification approaches like [DPG02], use local weighting in nearest neighbor classification to overcome this drawback. In this work, we take a different approach to identify locally relevant attributes by subspace clustering. Note that our approach is different from semi-supervised learning where unlabeled data is used for training [Zhu05]. Our approach assumes class labels that are directly incorporated into subspace clustering. Clustering is helpful for understanding the overall structure of a data set. Its aim is automatic grouping of the data in absence of any known class labels in historic data [HK01]. Since class labels are not known in advance ("unsupervised learning"), they are not used to classify according to given groupings ("supervised learning"). Hence clustering is not appropriate for classification purposes by its very nature [HK01]. However, the structures detected by clustering may be helpful for detecting local relevance of attributes. For noisy and high dimensional data, clustering is often infeasible as clusters are hidden by irrelevant attributes. Different attribute combinations might show different clustering structures, thus the aim of subspace clustering is to detect clusters in arbitrary projections ("subspaces") of the attributes. As the number of subspaces is exponential in the number of attributes, most approaches try to prune the subspace search space [AGGR98, CWZZ99, AKGS06]. Subspace clustering has been shown to successfully detect locally relevant attribute combinations [KKK04, AKMS07b].

We propose combining both worlds, supervised learning and unsupervised learning by incorporating class label information into subspace search and clustering. Classification based on these classifying subspace clusters exploits both class and local correlation information. The flight classification problem is used to evaluate our model. Its applicability, however, goes beyond this scenario. In fact, there are many more application areas where classification has to handle noisy multidimensional data with locally relevant attributes.

8.2 Subspace classification

Subspace clustering is a recent research area which tries to detect local structures in the presence of noise or high dimensional data where meaningful clusters can no longer be detected in all attributes [AGGR98, CWZZ99, KKK04]. As searching all possible subspaces is usually intractable, subspace clustering algorithms try to focus on promising subspace regions. The challenge is a suitable notion of interestingness for subspaces to find all relevant clusters. Subspace clustering is a technique well-suited to identify relevant regions of historic data, however, it is not suited for classification "as is". Our classification approach is capable of exploiting local patterns in the data for classification. This requires detecting subspaces and subspace clusters that are also based on class structure. Our *SubClass* model thus comprises three steps:

- step 1: **interesting subspaces** for classifying clusters: Section 8.2.1
- step 2: **classifying subspace clusters**: Section 8.2.2
- step 3: a **classification** scheme: Section 8.2.3

8.2.1 Step 1: Interesting Subspaces

Interesting subspaces for classifying clusters exhibit a clustering structure in their attributes as well as coherent class label information. Such a structure is reflected by homogeneity in the attribute values or class labels of that subspace. Homogeneity can be measured using Shannon Entropy [SW49], or entropy for short. From an information theoretic perspective, Shannon entropy is the minimum number of bits required for encoding information. More frequently occurring events are encoded with fewer bits than less frequent ones. The sum over logarithmic probabilities weighted by their probability, measures the amount of information, i.e. the heterogeneity of the data.

Definition 8.1 ***Shannon Entropy.*** *Given a random variable X and its possible events $v_1, .., v_m$ the Shannon Entropy $H(X)$ is defined as:*

$$H(X) = -\sum_{i=1}^{m} p(v_i) \cdot \log_2 p(v_i)$$

Transferring the entropy notion to the clustering or classification domain, an attribute can be seen as a random variable whose domain is the set of all possible events. In case of continuous domains, the entropy requires discretization of attributes. Entropy according to a set of attributes with respect to a set of class labels is then:

Definition 8.2 ***Attribute Entropy.*** *Given a set of attributes $X_1, \ldots, X_m$, their possible values $v_1, \ldots, v_m$, and class labels $C = \{c_1, \ldots, c_n\}$, attribute entropy is defined as:*

$$H(X_1, \ldots, X_m|C) = -\sum_{c_i \in C} \sum_{v_1 \in X_1} \cdots \sum_{v_m \in X_m} p(c_i) \cdot H(X_1, \ldots, X_m|C = c_i)$$

Attribute entropy is thus the sum over all conditional attribute entropy value combinations weighted by the class label probabilities. It is a measure for the clustering tendency for all class labels c_i of a subspace in terms of the attributes. To measure the clustering tendency in terms of individual class labels, we define class entropy according to conditional entropy $H(C|X)$ (as e.g. in [Qui92]).

Definition 8.3 ***Class Entropy.*** *Given a set of attributes $X_1, \ldots, X_m$, their possible values $v_1, \ldots, v_m$, and class label C the conditional entropy of a segmentation along these attribute values is defined as:*

$$H(C|X_1, \ldots, X_m) = -\sum_{v_1 \in X_1} \cdots \sum_{v_m \in X_m} p(v_1, .., v_m) \cdot H(C|X_1 = v_1, \ldots, X_m = v_m)$$

Class entropy is thus the sum over all conditional class entropy value combinations for individual class labels C. It corresponds to investigating the data for individual classes instead of aggregated as for attribute entropy.

We are interested in subspaces that exhibit both a distinct class structure as well as a clear clustering structure. Since entropy measures homogeneity, we are interested in low entropy values that reflect a non-uniform distribution of class or attribute values.

However, comparing subspaces using entropy is clearly biased with respect to the number of attributes. Subspaces with more attributes typically have lower entropy values. This is due to the fact that with increasing attribute number, objects tend to be less similar: each attribute contributes potential dissimilarity [AKMS07b]. Thus, we have to normalize entropy with respect to the number of attributes. Normalization to a range of [0,1] can be achieved by

taking the maximum possible entropy value for a given number of attributes into account. Maximum entropy means all values are equally likely, i.e. a uniform distribution. $H_{uniform}(X_1, .., X_m|C)$ for $d = |X_1 \times \cdots \times X_m|$ possible attribute combinations is determined as: $H_{uniform}(X_1, .., X_m|C) = -d \cdot \frac{1}{d} \cdot \log_2 \frac{1}{d} = -\log_2 \frac{1}{d} = \log_2 d$, since in uniform distribution, each attribute value occurs $1/d$ times. For larger numbers of attributes, the theoretical upper bound of $\log_2 d$ cannot be reached, as the actual number of instances is smaller than the number of possible attribute value combinations d. To account for this, we the number of instances $|I|$ is used in this case:

$$H_N(X_1, .., X_m|C) = \frac{H(X_1, .., X_m|C)}{\min\{log_2|I|, \log_2 d\}}$$

In a similar spirit, we use the overall class distribution to normalize class entropy:

$$H_N(C|X_1, .., X_m) = \frac{H(C|X_1, .., X_m)}{H(C)}$$

Since those subspaces are interesting that cover both aspects, we define interestingness as a convex combination of attribute and class entropy, provided that each of the two is within reasonable bounds:

Definition 8.4 ***Subspace Interestingness.*** *Given attributes $X_1, \ldots, X_m$, a class attribute C, and a weighting factor $0 \leq w \leq 1$, a subspace is interesting with respect to thresholds β, λ iff:*

$$w \cdot H_N(X_1, .., X_m|C) + (1 - w) \cdot H_N(C|X_1, .., X_m) \leq \beta$$
$$\wedge\ H_N(X_1, .., X_m|C) \leq \lambda\ \wedge\ H_N(C|X_1, .., X_m) \leq \lambda$$

Thus, a subspace is interesting for subspace classification if it shows low normalized class and attribute entropy as an indication of class and cluster structure. w allows assigning different weights to these two aspects for different applications, while λ is set to fairly relaxed threshold values to ensure that both aspects fulfill minimum entropy requirements.

8.2.2 Step 2: Classifying Subspace Clusters

Having defined interesting subspaces, the next step is detecting *classifying subspace clusters*. On discretized data, clusters can be defined as frequent attribute value combinations. To incorporate class information, these groupings should be homogeneous with respect to class label. We defined the absolute frequency

$$AbsFreq(v_1, \ldots, v_m) = |\{o, o|_S = (v_1, \ldots, v_m)|$$

as the number of objects o which exhibit the attribute values $(v_1, \ldots, v_m)$ in subspace S (projection $o|_S$ contains those attribute values v_i from o where $X_i \in S$).

To ensure that non-trivial clusters are mined, we normalize frequency with respect to the expected frequency of uniformly distributed subspaces. The expected frequency

$$ExpFreq(v_1, \ldots, v_m) = AbsFreq(v_1, \ldots, v_m) * d/|I|$$

is the number of cluster objects in comparison to the number of instances $|I|$ per attribute combination under uniform distribution. Classifying subspace clusters exceed minimum frequency for both absolute and relative (expected) frequency. Note that minimum absolute frequency simply ensures that a cluster exceeds a minimum size even for very small expected frequency values:

Definition 8.5 ***Classifying Subspace Cluster.*** *Given a subspace S of attributes $X_1, \ldots, X_m$, a classifying subspace cluster SC with respect to attribute values $v_1, \ldots, v_m$, minimum frequency thresholds ϕ_1, ϕ_2, and maximum entropy γ is defined as follows:*

- $H_N(C|X_1 = v_1, \ldots, X_m = v_m) \leq \gamma$
- $AbsFreq(v_1, \ldots, v_m) \geq \phi_1$
- $ExpFreq(v_1, \ldots, v_m) \geq \phi_2$

Classifying subspace clusters have low normalized class entropy, as well as high frequency in terms of attribute values. Thus, they are homogeneous in terms of class and show local attribute correlations.

8.2.3 Step 3: Classification

Classification of a given object o is based on the class label distribution of similar classifying subspace clusters. For nominal values as they occur in our flight data, an object o is typically contained in several subspace clusters and similarity is reduced to containment. Let $CSC(o) = \{SC_i | v_k = o_k \forall v_k \in SC_i\}$ denote the set of all classifying subspace clusters containing object o. Simply assigning the majority class label from this set $CSC(o)$ would be biased with respect to very large and redundant subspace clusters, where redundancy means similar clusters in slightly varying projections [AKMS07b]. We therefore propose an iterative procedure that takes the *information gain* into account to build the decision set $DS_k(o)$.

Just as in the subspace clustering step we measure class homogeneity using the conditional class entropy. Starting with an empty decision set and apriori knowledge about class distribution $H(C)$ we select up to k subspace clusters with maximal information gain on the class label as long as more than ϕ_1 objects are contained in the decision space, i.e. the projection to the union of dimensions of the subspace clusters in the decision set.

Definition 8.6 ***Classification.*** *Given a dataset D, parameter k, an object $o = (o_1, \ldots, o_d)$ is classified to the majority class label of decision set DS_k. DS_k is iteratively constructed from $DS_0 = \emptyset$ by selecting the subspace cluster $SC_j \in CSC(o)$ which maximizes the information gain about the class label:*

$$\begin{aligned} DS_j &= DS_{j-1} \cup SC_j, \\ SC_j &= \left\{ \underset{SC_i \in CSC(o)}{\textbf{argmax}} \{H(C|DS_{j-1}) - H(C|DS_{j-1} \cup SC_i)\} \right\} \end{aligned}$$

under the constraints that the decision space contains at least ϕ_1 objects:

$$|\{v \in D, v|_{DS_k} = o|_{DS_k}\}| \geq \phi_1$$

and that the information gain is positive

$$H(C|DS_{j-1}) - H(C|DS_{j-1} \cup SC_i) > 0$$

Hence, the decision set of an object o is created by choosing those k subspace clusters containing o that provide most information on the class label, as long as more than a minimum number of objects are in the decision space.

o is then classified according to the majority in the decision set DS_k. The decision set is then the set of locally relevant attributes that were used to classify object o. The attributes in the decision set are helpful for users wishing to understand the information that led to classification.

8.3 Algorithmic concept

Our algorithmic concept focuses on step 1 that is the computationally most complex. A simple brute-force search would require evaluating all 2^N subspaces which is not acceptable for high dimensionality N. We thus propose lossless pruning of subspaces based on two entropy monotonicities.

Theorem 8.1 ***Upward Monotony of the Class Entropy.*** *Given a set of m attributes, subspace $S = \{X_1, .., X_m\}$, $e \in \mathcal{R}^+$ and $T \subseteq S$, the class entropy in subspace T is less than or at most equal to the class entropy of its superspace S:*

$$H(C|T) < e \quad \Rightarrow \quad H(C|S) < e$$

Proof. The theorem follows immediately from $H(X|X_i, X_j) \leq H(X|X_i)$ [Gra90].

This theorem states that the class entropy decreases monotonically with growing number of attributes. Conversely, attribute entropy increases monotonically with the number of attributes.

Theorem 8.2 ***Downward Monotony of the Attribute Entropy.*** *Given a set of m attributes, subspace $S = \{X_1, .., X_m\}$, $e \in \mathcal{R}^+$ and $T \subseteq S$, the attribute entropy in subspace T is greater than or at most equal to the class entropy of its superspace S:*

$$H(S|C) < e \quad \Rightarrow \quad H(T|C) < e$$

Proof. The theorem follows immediately from $H(X_i, X_j|C) \geq H(X_i|C)$ [Gra90].

We exploit monotonicity by pruning

- all those subspaces T whose superspaces $S \supset T$ fail the class entropy threshold. This is correct since the normalization factor $H(C)$ is independent of the subspace.

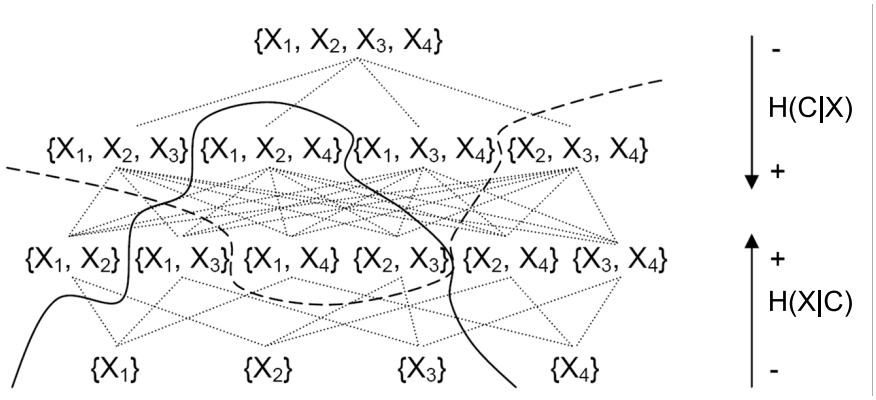

Figure 8.1: Lattice of subspaces and their projections used for up- and downward pruning

- Prune all those superspaces T whose subspaces $S \subset T$ fail the attribute entropy threshold if $log_2|I| \geq log_2|S|$. This is correct since the normalization factor is independent of the subspace if $\min\{log_2|I|, log_2|S|\} = log_2|I|$.

Our proposed algorithm alternately determines lower dimensional and higher dimensional *one-sided homogeneous* subspaces, i.e. subspaces that are homogeneous w.r.t. to class or attribute entropy, respectively. In each step new candidates are created from the set of one-sided homogeneous subspaces mined in the last step.

Figure 8.1 illustrates pruning in a subspace lattice of four attributes. The solid line is the boundary for pruning according to attribute entropy and the dashed line according to class entropy. Each subspace below the attribute boundary and above the class boundary is homogeneous with respect to the entropy considered. The subspaces between both boundaries are interesting subspace candidates, whose combined entropy has to be computed in the next step.

For the bottom up case, the apriori property, originally from association rule mining, can be used to create new candidates [AS94, CWZZ99, KKK04]. Following the apriori approach, we join two attribute homogeneous subspaces of size m with identical prefixes (e.g. in lexicographic ordering) to create a candidate subspace of size $m + 1$. After this, each new candidate is checked for entropy validity, i.e. if all of its possible subspace of cardinality m are contained in the set of attribute homogeneous candidate subspaces.

We suggest a similar method for top down candidate generation using class monotonicity. From the set of class homogeneous subspaces of dimensionality m, we generate all subspace candidates of dimensionality $m-1$. We develop a method that ensures that each subspace candidate is only generated once. Based on the lexicographic order, our method uniquely generates a subspace of dimensionality $m-1$ from its smallest superspace. Note that this guarantees that all candidates but no superfluous candidates are generated (see example below). After this, just as with apriori, we check whether all superspaces containing the newly generated candidates are class homogeneous subspaces. Otherwise the new generated subspace is removed from the candidate set.

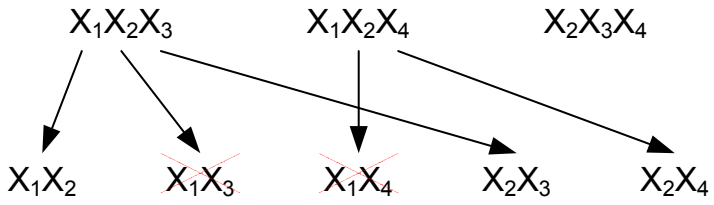

Figure 8.2: Example top down generation

Example. Assume four attributes $X_1, \ldots, X_4$ from the previous step subspaces $X_1X_2X_3$, $X_1X_2X_4$, and $X_2X_3X_4$ that satisfy the class entropy criterion. In order to generate candidates, we iterate over these subspaces in lexicographic order. The first three-dimensional subspace $X_1X_2X_3$ generates the two-dimensional subspaces X_1X_2 (drop X_3), X_1X_3 (drop X_2), X_2X_3 (drop X_1). Next, $X_1X_2X_4$ generates X_1X_4 and X_2X_4. X_1X_2 is not generated by $X_1X_2X_4$ again, because dropping X_4 is not possible, as it is preceded by X_3 which is not contained in this subspace. The last three-dimensional subspace $X_2X_3X_4$ does not generate any two-dimensional subspace since the leading X_1 is not contained; its subsets X_2X_3 and X_2X_4 have been generated by other three- dimensional subspaces. After candidate generation, we check their respective supersets. For example, for X_1X_2, its supersets $X_1X_2X_3$ and $X_1X_2X_4$ exist. For X_1X_3, its superset $X_1X_2X_3$ exists, but $X_1X_3X_4$ does not, so it is removed from further consideration following monotony pruning. Likewise, X_1X_4 is removed as $X_1X_3X_4$ is missing, but X_2X_3 and X_2X_4 are kept.

As we use two entropies, one with downward, one with upward pruning, subspaces may need to be considered twice. Minimizing computations is thus a trade-off. Figure 8.3 illustrates these effects. A missing candidate in S_{Down} (e.g. X_1X_2) means that this candidate has an attribute entropy above β. According to the attribute monotony, superspaces (e.g. $X_1X_2X_3$) have an attribute entropy above β and thus the combined entropy is also greater than β. Even though the subspace could be pruned according to combined entropy, it is still required for valid class entropy candidate generation. There

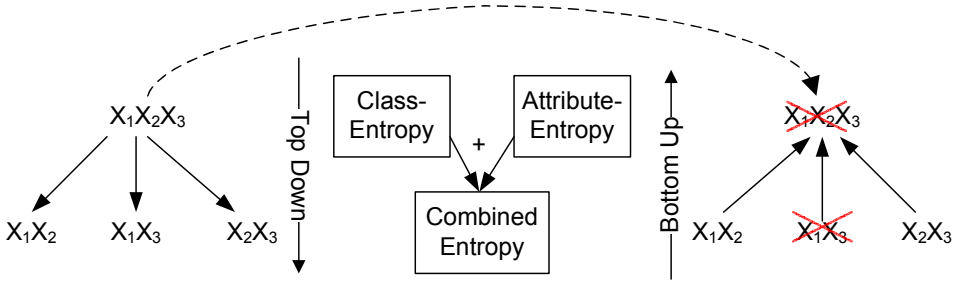

Figure 8.3: Pruning of subspace $X_1X_2X_3$

is thus a trade off between avoiding computations and reducing the search space by pruning high entropy subspaces. A good heuristic is to evaluate the entropy of those subspaces for which larger subspaces already had a high entropy. Randomly picking subspaces for additional evaluation also performs quite well in practice.

If the bottom up approach has not pruned the investigated subspace, the top down approach computes the entropy of the subspace. If the weighted normalized entropy is below β the subspaces is added to the result set and marked as one-sided homogeneous. The algorithm finally computes the combined entropy of all subspaces for which both subspaces are marked one-sided homogeneous in the result sets.

Once subspaces have been evaluated for **step 1**, the most complex algorithmic task has been solved. Having reduced the potentially exponential number of subspaces to the interesting ones, the actual clustering (**step 2**) is performed for each of these subspaces. This is done by computing the frequency and class entropy for all attribute value combinations in these subspaces. The resulting classifying subspace clusters then provide the model that is used for the actual classification (**step 3**). For incoming objects,

compute the most similar classifying subspace clusters according to relative Hamming distance. If tied, compute reverse class entropy. The decision is then based on their class label distribution.

8.4 Experiments

Experiments were run on both synthetic and real world data. Synthetic data is used to show the correctness of our approach. Local patterns are hidden in a data set of 7.000 objects and eight attributes. As background noise, each attribute of the synthetic data set is uniformly distributed over ten values. On top of this, 16 different local patterns (subspace clusters) with different dimensionalities and different numbers of objects are hidden in the data set. Each local pattern contains two or three class labels among which one class label is dominating. We randomly picked 7.000 objects for training and 1.000 objects for testing.

The flight data contains historic data from a large European airport. For a three-month period, we trained the classifier on arrivals of two consecutive months and tested on the following month. Outliers with delays outside [-60, 120] minutes have been eliminated. In total, 11.072 flights have been used for training and 5.720 flights for testing. Each flight has a total of 13 attributes, including e.g. the airline, flight number, aircraft type, routing, and the scheduled arrival time within the day. The class labels are "ahead of schedule", "on time" and "delayed". Finally we use two well-known real world data sets from the UCI KDD archive (Glass and Iris [HB99]), as a general benchmark.

As mentioned before, preliminary experiments on the flight data indicate that no global relevance of attributes exist. Moreover, the data is inherently noisy, and important influences like weather conditions are not collected from scheduling. For realistic testing as in practical application, classifiers can only draw from existing attributes. Missing or not collected parameters are not available for training or testing neither in our experiments nor during the actual scheduling process.

We have conducted prior experiments to evaluate the effect of ϕ and γ for minimum frequency and maximum entropy thresholds, respectively. For each data set we used a cross validation to chose ϕ_1 (absolute frequency), ϕ_2 (relative frequency) and γ. For λ we have chosen 0.9. This value corresponds

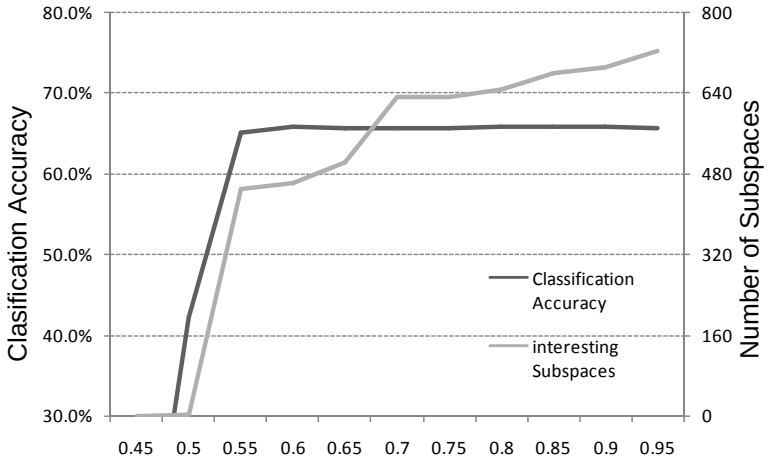

Figure 8.4: Varying β on synthetic data

to a rather relaxed setting as we only want to remove completely inhomogeneous subspaces from consideration. To restrict the search space β can be set to a low value.

In our first experiments we develop a heuristic to set up reasonable parameters for the threshold β of the interestingness and the weight w of the class and attribute entropy, respectively.

Figure 8.4 illustrates varying β from 0.45 to 0.95 on the synthetic data, measuring classification accuracy and the number of classifying subspaces. The weight w for interestingness was set to 0.5. As expected, the number of classifying subspaces (CSS) decreases when lowering the threshold β. At the same time, the classification accuracy does not change substantially or even increases slightly when less subspaces are used. This effect may be related to the effect of overfitting. Using too many subspaces patterns are not sufficiently generalized, and noise is not removed. To set up the threshold β, slowly increasing β until the number of classifying subspace clusters shows a rapid rise, allows adjusting β to a point between generalization and overfitting. For both our data sets, a value around 0.65 obtains produces good results.

The effect of slightly increasing classification accuracy when reducing the number of subspaces can also be observed on the flight delay data (see Figure 8.4). This confirms that the flight data contains local patterns for classifica-

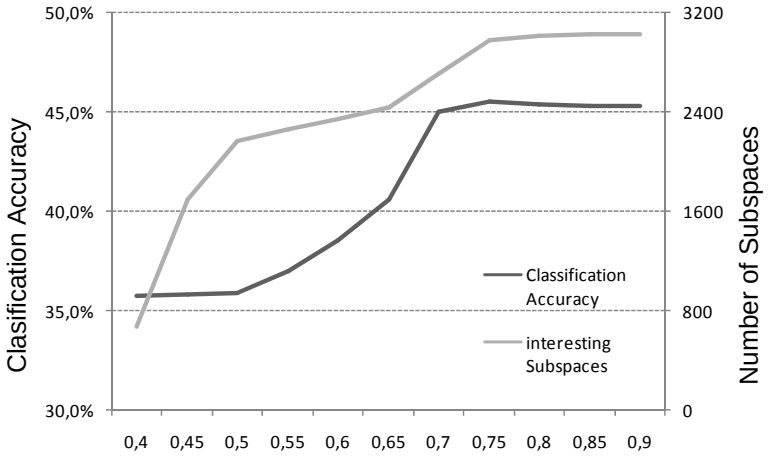

Figure 8.5: Varying β on flight delay data

tion.

Varying parameter w yields the results depicted in the left part of Figure 8.4 and 8.4. The number of classifying subspaces decreases when giving more weight to attribute entropy. At the same time, classification accuracy does not change significantly. This robustness is due to the ensuing subspace clustering phase. As classification accuracy does not change this confirms that our classifying subspace cluster definition selects the relevant patterns. Setting $w = 0.5$ gives equivalent weight to the class and attribute entropy and hence is a good choice for pruning subspaces.

Next, we evaluate classification accuracy by comparing *SubClass* with other well-established classifiers that are applicable on nominal attributes: the k-NN classifier with Manhattan distance, the C4.5 decision tree that also uses a class and attribute entropy model [Qui92], and a Naive Bayes classifier, a probabilistic classifier that assumes independence of attributes. Parameter settings use the best values from the preceding experiments.

Figure 8.8 illustrates the classification accuracy using four different data sets. In the noisy synthetic data set, our *SubClass* approach outperforms other classifiers. The large degree of noise and the varying class label distribution within the subspace clusters make this a challenging task. From the real world experiment on the flight data, depicted in Figure 8.8, we see that the situation is even more complex. Still, our *SubClass* method performs

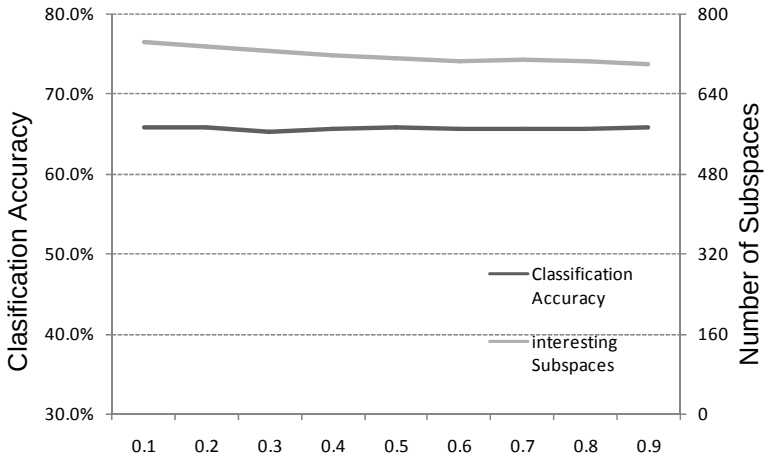

Figure 8.6: Varying w on synthetic data

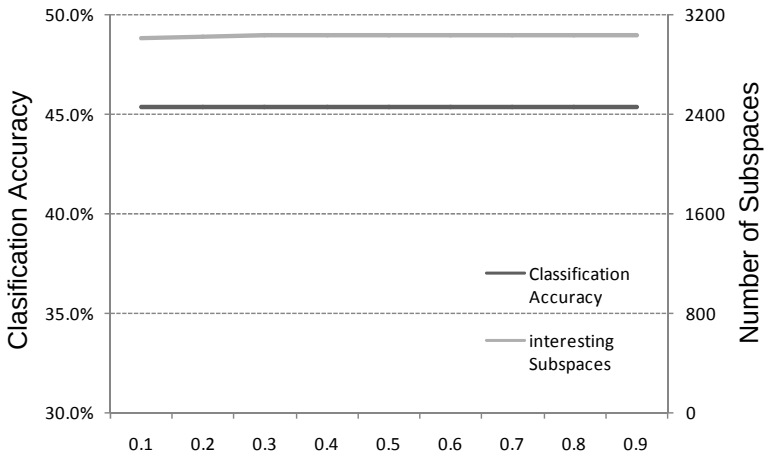

Figure 8.7: Varying w on flight delay data

	Flight Data	Synthetic Data	Iris	Glass
SubClass	**45.4%**	**65.9%**	**96.27%**	70.9%
C4.5	43.9%	58.0%	95.94%	66.8%
K-NN	42.4%	54.3%	93.91%	**71.1%**
Naive Bayes	42.8%	64.1%	95.27%	46.7%

Figure 8.8: Classification accuracy on four data sets

better than its competitors. This result supports our analysis that locally relevant information for classification exists that should be used for model building. Experts from flight scheduling confirm that additional information on further parameters, e.g. weather conditions, is likely to boost classification. This information is inexistent in the current scheduling data that is collected routinely. *SubClass* exploits all the information available, especially locally relevant attribute and value combinations, for the best classification in this noisy scenario. Finally we evaluated the performance of *SubClass* on Glass and Iris [HB99]. The results indicate that even in settings containing no or little noise *SubClass* performs well.

8.5 Conclusion

Classification in noisy data with locally varying attribute relevance, as for our project in scheduling at airports, requires an approach that detects local patterns. Our *SubClass* method automatically detects classifying subspace clusters by incorporating class structure into the subspace search and the subspace clustering process. Our experiments demonstrate that local structures are successfully detected and employed for classification, even in extremely noisy data.

Conclusion and future research directions

Conclusion

The knowledge discovery process aims at finding patterns in large databases. Clustering and classification are important subtasks of the KDD process. Since databases grow tremendously in size efficient methods for clustering and classification are required. In this thesis we proposed efficient density-based methods for both clustering and classification. While improving efficiency we always made sure that it was not detrimental for the effectiveness. Consequently, we always discussed both efficiency and effectiveness for the new methods proposed in this work. In the following two sections we summarize the main contributions of this thesis.

Contributions in efficient density-based clustering

In the first part of this thesis we proposed new density-based methods for clustering multi-dimensional data. After discussing some aspects of the so called "curse of dimensionality" which are relevant for clustering higher dimensional data, we presented a possible solution: density-based subspace clustering.

Based on the definitions of subspace clustering we formalized the so called "dimensionality bias" in Chapter 2 and analyzed the consequences for subspace clusters. We developed a dimensionality unbiased density measure based on statistical foundations. Further more, we discussed redundancy effects and the empty space problem for subspace clusters. The main contribution of this chapter is the DUSC subspace clustering model. The DUSC model extends traditional density-based subspace clustering models by an unbiased density measure and a definition for non-redundant clusters. We have proven the high effectiveness of the DUSC model in various experiments on different real world data sets. The DUSC model and parts of the discussion presented in this chapter were also presented at 2007 IEEE international conference on data mining [AKMS07b].

Next we proposed an efficient and complete algorithm for our DUSC subspace clustering model in Chapter 3. To ensure efficient mining of density-based subspace clusters, we derived powerful pruning properties. Based on these pruning properties and a novel density conserving grid we developed a filter-and-refinement architecture to prune the search space. Further on, we explained how to index dense subspace regions and proposed an method

for in-process removal of redundant subspace clusters. We evaluated the efficiency of eDUSC in thorough experiments.

Visualization techniques are essential to present a clustering result to users. This thesis proposed the first method to visualize subspace clusters. Our VISA technique allows for both getting an overview of a subspace clustering and an indepth-analysis of the result set. Excerpts of this chapter were also published in the 2007 special issue on visual analytics of the ACM SIGKDD Explorations [AKMS07a].

The last subspace clustering method proposed in this thesis is motivated by collaborations with researchers from hydrology. A subspace subsequence clustering model has been developed for finding clusters in a multi-dimensional sequence database. Scalability to large databases like the river database from hydrology is guaranteed by our novel index structure for dense sequences. Based on the clustering results hydrologists developed a decision support system for the renaturation of rivers. Parts of this work are accepted for publication in the international journal on knowledge and information systems [AKGS08]. A presentation on this topic was also given at the international workshop on spatial and spatio-temporal data mining [AKGS06].

Summary for efficient density-based classification

In the second part of this thesis two different density-based classification methods are discussed. After introducing preliminary definitions for Bayes and nearest neighbor classification we presented the special requirements of anytime classifiers in Chapter 7. We proposed a novel hierarchy of densities (the BayesTree) and presented different search strategies for efficient anytime classification. For streams of variable inter-arrival we discussed the advantages of anytime classifiers and evaluated the accuracy of our approach in various settings.

Last but not least, we proposed a new classification technique based on classifying subspace clusters. We developed a subspace clustering algorithm which considers information about the class distribution during mining of clusters. The efficiency and effectiveness of our new classification approach is evaluated in collaboration with a local company for operations research. Parts of this work will be presented at the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining [AKW+08].

Future research directions

The results presented in this work open various directions for future research:

Subspace clustering

The DUSC subspace clustering model proposed in this thesis has proven to be a very effective method for subspace clustering. The two aspects responsible for the high quality results are an unbiased density measure and the definition of redundant subspace clusters. Both concepts solve a specific problem but leave room for further research.

Redundancy definitions

The redundancy definition presented in this thesis defines subspace clusters as redundant if a similar higher dimensional cluster exists in the data set. An advantage of this redundancy is its simplicity: for users the definition is easy to understand and the effect of different redundancy parameters on the mined cluster result is comprehensible.

However, more complex redundancy definitions may further improve the quality of the clustering result and reduce the number of identified clusters. We plan to examine different extended redundancy definitions. A first possible extension of our redundancy definition denotes a cluster as redundant if it is represented by a set of higher dimensional clusters. Following this redundancy definition a cluster which splits up into multiple higher dimensional clusters would hence be removed from the result set. Depending on the application such a redundancy definition may be more appropriate.

Moreover, we are working on a redundancy definition inspired by the *set covering problem*. We plan to generalize the definition of redundancy from subspaces to object sets. An "optimal" redundancy free subspace cluster is then defined as a set of clusters which covers all objects. Optimality can be defined in different ways. The expected density, the size and the dimensionality of a cluster are parameters which should be considered to determine clusters of high quality. As the set covering problem is NP-hard a major challenge will be to develop an efficient algorithm which approximates this redundancy definition as good as possible.

Unbiased density assessment

The DUSC subspace clustering model formalizes the requirements for finding comparable subspace clusters which separate clusters from noise. An additional important aspect clustering higher dimensional spaces is to correctly assess the density of objects. As discussed under the name of empty space problem, the area of influence for higher dimensional spaces is often devoid of observations.

One solution for estimating the density of objects in higher dimensions is to extend the area of influence (i.e. to adapt ε). For this approach the weak monotonicity property proposed in Section 2.6 does not hold anymore. Further on, the size of the connectivity border would also depend on the dimensionality of the subspace (see Section 3.4). The development of efficient algorithms for subspace clusters based on an adaptive area of influence is an interesting area of research. Without any monotonicity property for pruning the search space heuristics for finding subspace clusters might be a promising approach.

Subspace outliers

In this thesis we discussed the effectiveness of subspace clustering methods in high-dimensional spaces. Outlier detection in high-dimensional spaces also suffer from the "curse of dimensioanlity". Typical approaches for outlier detection are often based on clustering methods, e.g. objects which do not belong to a cluster or are far away from a cluster center are very likely outliers. Since subspace clusters are capable of identifying locally relevant patterns, a promising approach is to use subspace clusters for outlier mining.

Open questions are how to handle overlapping clusters (e.g. objects belonging to multiple clusters) or contradictory evidences (e.g. an object may belong to a subspace cluster and at the same time be an outlier w.r.t. another cluster). Mining outliers is clearly influenced by the quality and degree of redundancy of a clustering result. Hence, detecting subspace outliers could be an interesting field of research.

Anytime classification

This thesis presented an anytime classification approach using a hierarchy of densities (the Bayes tree). First results indicate the usefulness of this

approach especially for stream classification. The performance of the Bayes tree depends on the quality of the mixture densities stored in the tree. In this thesis the dynamic insertion strategy of the R*-tree is used which makes efficient learning of new objects possible.

Evaluating the effects of different bulkloading techniques on the quality of the mixture models could be an interesting approach for future work. Bulkloading techniques analyze the data before the hierarchy is created. The development of bulkloading techniques based on hierarchical mixture models could further improve the classification performance of the Bayes tree.

Another interesting research direction is to develop a Bayes tree which does not separate the classes at root level. For a Bayes tree containing multiple classes different questions have to be investigated.

- Which information about the class distribution should be stored in the inner nodes of the Bayes tree? Detailed information about the classes decreases the branching factor and hence the query performance. The classification accuracy on the other hand also depends on information given about the classes (more details typically increase the classification accuracy).

- How should the Bayes tree organize multiple classes? Mixture densities which reflect the data distribution very well may not necessarily separate the classes. However, separating the classes on a higher level of the tree would allow for an early class decision.

- Which anytime classification strategy is reasonable for a Bayes tree containing multiple classes? Global best first methods may only refine one class over and over again. However, detailed information about other classes is also necessary for an effective classification decision.

Further on, developing a method for streams with constant inter-arrival times is be a challenging task. One possibility is to derive a confidence measure for a classification decision at any time point. The overall classification performance of a stream with constant inter-arrival times could then be improved by optimizing the overall confidence.

Bibliography

[ABKS99] M. Ankerst, M.M. Breunig, H.P. Kriegel, and J. Sander. OPTICS: ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM ACM SIGMOD International Conference on Management of Data*, pages 49–60, 1999.

[ADGK07] B. Arai, G. Das, D. Gunopulos, and N. Koudas. Anytime measures for top-k algorithms. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, pages 914–925, 2007.

[AFGY02] J. Ayres, J. Flannik, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery in Databases*, pages 429–435, 2002.

[AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 94–105, 1998.

[AHLP06] N. Agarwal, E. Haque, H. Lie, and N. Parsons. A subspace clustering framework for research group collaboration. *International Journal of Information Technology and Web Engineering*, 34:35–57, 2006.

[AKA91] D.W. Aha, D. Kibler, and M.K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[AKGS06] I. Assent, R. Krieger, B. Glavic, and T. Seidl. Spatial multidimensional sequence clustering. In *Proceedings of the 1st In-*

ternational Workshop on Spatial and Spatio-temporal Data Mining (SSTDM), in conjunction with the 2006 IEEE International Conference on Data Mining (ICDM), pages 343–348, 2006.

[AKGS08] I. Assent, R. Krieger, B. Glavic, and T. Seidl. Clustering multidimensional sequences in spatial and temporal databases. *International Journal on Knowledge and Information Systems (KAIS) (to appear)*, 2008.

[AKKS99] M. Ankerst, G. Kastenmüller, H.P. Kriegel, and T. Seidl. Nearest Neighbor Classification in 3D Protein Databases. In *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 34–43, 1999.

[AKMS07a] I. Assent, R. Krieger, E. Müller, and T. Seidl. VISA: Visual Subspace Clustering Analysis. *ACM SIGKDD Explorations Special Issue on Visual Analytics, Vol. 9, Issue 2*, pages 5–12, 2007.

[AKMS07b] I. Assent, R. Krieger, E. Müller, and T. Seidl. DUSC: Dimensionality Unbiased Subspace Clustering. In *Proceedings of the 2007 IEEE International Conference on Data Mining (ICDM)*, pages 409–414, 2007.

[AKSS06] I. Assent, R. Krieger, A. Steffens, and T. Seidl. A Novel Biology inspired Model for Evolutionary Subspace Clustering. In *Proceedings of the Annual Symposium on Nature inspired Smart Information Systems (NiSIS 2006)*, 2006.

[AKW+08] I. Assent, R. Krieger, P. Welter, J. Herbers, and T. Seidl. SubClass: Classification of Multidimensional Noisy Data Using Subspace Clusters. In *Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2008) (to appear)*, 2008.

[ALV03] C. Archambeau, J.A. Lee, and M. Verleysen. On Convergence Problems of the EM Algorithm for Finite Gaussian Mixtures. In *11th European Symposium on Artificial Neural Networks*, pages 99–106, 2003.

[AN07] A. Asuncion and D. Newman. University of California Machine Learning Repository, http://www.ics.uci.edu/∼mlearn/MLRepository.html, 2007.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, pages 487–499, 1994.

[AS95] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the 1995 IEEE International Conference on Data Engineering (ICDE)*, pages 3–14, 1995.

[AS04] D. Andre and P. Stone. Physiological Data Modeling Contest (ICML-2004): http://www.cs.utexas.edu/ users/pstone/Workshops /2004icml/, 2004.

[AWY+99] C. Aggarwal, J. Wolf, P. Yu, C. Procopiuc, and J. Park. Fast algorithms for projected clustering. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 61–72, 1999.

[AY00] C. Aggarwal and P. Yu. Finding generalized projected clusters in high dimensional spaces. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 70–81, 2000.

[Bar05] S. Bartusseck. Regelbasiertes Entscheidungsunterstützungssystem (DSS) zur Bewertung von Maßnahmenplänen gemäß EG-WRRL. *Forum für Hydrologie und Wasserbewirtschaftung*, 10, 2005.

[Bay63] T. Bayes. An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society*, 53:370–418, 1763.

[BGRS99] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbors meaningful. In *Proceedings of the 7th International Conference on Database Theory (ICDT)*, pages 217–235, London, UK, 1999.

[BKK96] S. Berchtold, D. Keim, and H.P. Kriegel. The X-Tree: An Index Structure for High-Dimensional Data. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB)*, pages 28–39, 1996.

[BKSS90] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990.

[Bol00] A. Bolat. Procedures for providing robust gate assignments for arriving aircrafts. *European Journal of Operational Research*, 120:63–80, 2000.

[Bou04] R.R. Bouckaert. Naive Bayes Classifiers that Perform Well with Continuous Variables. In *Proceedings of the 2006 Advances in Artificial Intelligence (AI)*, pages 1089–1094, 2004.

[BPS06] C. Böhm, A. Pryakhin, and M. Schubert. The Gauss-Tree: Efficient Object Identification in Databases of Probabilistic Feature Vectors. In *Proceedings of the 2006 IEEE International Conference on Data Engineering (ICDE)*, pages 9–17, 2006.

[Bur98] C.J.C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[Bur05] Bureau of Transportation Statistics. Airline on-time performance data. Available from http://www.transtats.bts.gov, 2005.

[CJ02] J.W. Chang and D.S. Jin. A new cell-based clustering method for large, high-dimensional data in data mining applications. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC)*, pages 503–507, 2002.

[CW02] Y. Cao and J. Wu. Projective art for clustering data sets in high dimensional spaces. *Neural Networks*, 15(1):105–120, 2002.

[CWZZ99] C.H. Cheng, A. Waichee, F. Zhang, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 84–93, 1999.

[Def77] D. Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 1977.

[Den04] A. Denton. Density-based Clustering of Time Series Subsequences. In *Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM)*, 2004.

[DHS01] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, New York, 2001.

[DLR+77] A.P. Dempster, N.M. Laird, D.B. Rubin, et al. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[DPG02] C. Domeniconi, J. Peng, and D. Gunopulos. Locally Adaptive Metric Nearest-Neighbor Classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1281–1285, 2002.

[DPGM04] C. Domeniconi, D. Papadopoulos, D. Gunopulos, and S. Ma. Subspace clustering of high dimensional data. *Proceedings of the 4th SIAM International Conference on Data Mining (SDM)*, 2004.

[EKSX96] M. Ester, H.P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. In *Proceedings of the 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[EM05] S. Esmeir and S. Markovitch. Interruptible anytime algorithms for iterative improvement of decision trees. In *Proceedings of the 1st international workshop on Utility-based data mining (UBDB)*, pages 78–85, 2005.

[Epa69] V.A. Epanechnikov. Non-Parametric Estimation of a Multivariate Probability Density. *Theory of Probability and its Applications*, 14:153, 1969.

[ESBB98] M.B. Eisen, P.T. Spellman, P.O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. In *Proceedings of the National Academy of Science of the USA*, volume 95, pages 14863–14868, 1998.

[Eur05] Eurocontrol Central Office for Delay Analysis. Delays to air transport in europe. Available from http://www.eurocontrol.int/eCoda, 2005.

[FdOL03] M.C. Ferreira de Oliveira and H. Levkowitz. From visual data exploration to visual data mining: a survey. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):378–394, 2003.

[FGW02] U. Fayyad, G.G. Grinstein, and A. Wierse, editors. *Information visualization in data mining and knowledge discovery*. Morgan Kaufmann Publishers Inc., 2002.

[FRM94] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 419–429, 1994.

[FvDFH96] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics (2nd ed. in C): Principles and Practice*. Addison-Wesley, 1996.

[GFC05] Weather Data Retrieval Georgia Forestry Commission. http://weather.gfc.state.ga.us, 2005.

[GM03] Alexander G. Gray and Andrew W. Moore. Nonparametric Density Estimation: Toward Computational Tractability. In *Proceedings of the 4th SIAM International Conference on Data Mining (SDM)*, 2003.

[GR92] J.J. Grefenstette and C.L. Ramsey. An Approach to Anytime Learning. *Workshop on Machine Learning*, pages 189–195, 1992.

[Gra90] R. M. Gray. *Entropy and Information Theory*. Springer, 1990.

[GRS99] S. Guha, R. Rastogi, and K. Shim. A robust clustering algorithm for categorical attributes. In *Proceedings of the 1999 IEEE International Conference on Data Engineering (ICDE)*, pages 512–521, 1999.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[GZ04] G. Grahne and J. Zhu. Mining frequent itemsets from secondary memory. In *Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM)*, pages 91– 98, 2004.

[HB99] S. Hettich and S. Bay. The uci kdd archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science, 1999.

[HD02] G. Hulten and P. Domingos. Mining complex models from arbitrarily large databases in constant time. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 525–531, 2002.

[HG07] A. Hinneburg and H.H. Gabriel. Denclue 2.0: Fast clustering based on kernel density estimation. In *Proceedings of the 7th International Symposium on Intelligent Data Analysis (IDA)*, pages 70–80, 2007.

[HK98] S. Hinneburg and D.A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of the 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 58–65, 1998.

[HK01] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[HPY00] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD*

International Conference on Management of Data, pages 1–12, 2000.

[HTF02] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2002.

[HTF07] T. Hastie, R. Tibshirani, and J. H. Friedman. Datasets for "The Elements of Statistical Learning": http://www-stat-class. stanford.edu/ tibs/elemstatlearn/, 2007.

[HZ96] E. A. Hansen and S. Zilberstein. Monitoring anytime algorithms. *SIGART Bulletin*, 7(2):28–33, 1996.

[Ins85] A. Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(4):69–91, 1985.

[JD88] A.K. Jain and R.C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1988.

[JL95] G. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 338–345, 1995.

[Jol86] I.T. Joliffe. *Principal Component Analysis*. Springer, New York, 1986.

[KCMP01] E.J. Keogh, K. Chakrabarti, S. Mehrotra, and M.J. Pazzani. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 151–162. ACM, 2001.

[KDS+08] P. Kranen, Kensche D., Kim S., Zimmermann N., E. Muller, Quix C., Li X., Gries T., T. Seidl, Jarke M., and Leonhardt S. Mobile mining and information management in healthnet scenarios. In *Proceedings of the 9th International Conference on Mobile Data Management (MDM)*, 2008.

[Kei02] D.A. Keim. Information Visualization and Visual Data Minig. *IEEE Transactions on Visualization and Computer Graphics*, 8:1–8, 2002.

[KKK04] K. Kailing, H.P. Kriegel, and P. Kröger. Density-Connected Subspace Clustering for High-Dimensional Data. In *Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM)*, pages 246–257, 2004.

[KKKW03] K. Kailing, H.P. Kriegel, P. Kröger, and S. Wanka. Ranking interesting subspaces for clustering high dimensional data. In *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 241–252, 2003.

[KKRW05] H.P. Kriegel, P. Kröger, M. Renz, and S. Wurst. A Generic Framework for Efficient Subspace Clustering of High-Dimensional Data. In *Proceedings of the 2005 IEEE International Conference on Data Mining (ICDM)*, pages 250–257, 2005.

[KKSS04] K. Kailing, H.P. Kriegel, S. Schönauer, and T. Seidl. Efficient Similarity Search for Hierarchical Data in Large Databases. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT)*, pages 676–693, 2004.

[KKZ07] H.P. Kriegel, P. Kröger, and A. Zimek. Detecting Clusters in Moderate-to-high Dimensional Data. In *Proceedings of the Tutorial in conjunction with 2007 IEEE International Conference on Data Mining (ICDM)*, 2007.

[KMSZ06] D.A. Keim, F. Mansmann, J. Schneidewind, and H. Ziegler. Challenges in Visual Data Analysis. In *Proceedings of the 2006 IEEE International Conference on Information Visualization*, pages 9–16, 2006.

[Kru64] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. In *Psychometrika*, volume 29, pages 1–27. Springer New York, 1964.

[KS02] I. Kotenko and L. Stankevitch. The control of teams of autonomous objects in the time-constrained environments. In *Proceedings of the 2002 IEEE International Conference on Artificial Intelligence Systems (ICAIS)*, pages 158–163, 2002.

[KS04] B. Kovalerchuk and J. Schwing. *Visual and Spatial Analysis - Advances in Data Mining, Reasoning, and Problem Solving.* Springer, 2004.

[KWX+06] E. Keogh, L. Wei, X. Xi, S.H. Lee, and M. Vlachos. LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance-with Arbitrary Representations and Distance Measures. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*, pages 882–893, 2006.

[Lau95] S. Lauritzen. The EM algorithm for graphical association models with missing data. *Comp. Statistics & Data Analysis*, 19:191–201, 1995.

[LKLC03] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery (DMKD)*, pages 2–11, 2003.

[LMG03] D.J. Lizotte, O. Madani, and R Greiner. Budgeted Learning of Naive-Bayes Classifiers. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 378–385, 2003.

[LUA03] LUA NRW. River quality data. http://www.lua.nrw.de, 2003.

[Luc94] M. Lucente. *Diffraction-Specific Fringe Computation for Electro-Holography.* PhD thesis, Massachusetts Institute of Technology, 1994.

[LWY04] J. Liu, W. Wang, and J. Yang. A framework for ontology-driven subspace clustering. *Proceedings of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–628, 2004.

[Mac67] J.B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathemtical Statistics and Probability*, pages 281–297, 1967.

[MKSW00] K. Myers, M. J. Kearns, S. P. Singh, and M. A. Walker. A boosting approach to topic spotting on subdialogues. In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, pages 655–662, 2000.

[MP43] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–137, 1943.

[MSE06] G. Moise, J. Sander, and M. Ester. P3C: A Robust Projected Clustering Algorithm. In *Proceedings of the 2006 IEEE International Conference on Data Mining (ICDM)*, pages 414–425, 2006.

[Mur98] S. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.

[NGC99] H. Nagesh, S. Goil, and A. Choudhary. MAFIA: Efficient and scalable subspace clustering for very large data sets. In *TR 9906-010, NWU*, 1999.

[Par62] E. Parzen. On the estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076, 1962.

[PF70] E. Patrick and F. Fischer. A generalized k-nearest neighbor rule. *Information and Control*, 16(2):128–152, 1970.

[PJAM02] C.M. Procopiuc, M. Jones, P.K. Agarwal, and T.M. Murali. A Monte Carlo algorithm for fast projective clustering. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 418–427, 2002.

[PKLL02] P. Patel, E. Keogh, J. Lin, and S. Lonardi. Mining motifs in massive time series databases. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)*, page 370, 2002.

[Pla98] J. Platt. Fast Training of Support Vector Machines using Sequential Minimal Optimization. In *Advances in Kernel Methods*. MIT Press, 1998.

[Qui86] J. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Qui92] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992.

[Rob04] J.C. Roberts. Exploratory visualization using bracketing. In *Proceedings of the ACM Working Conference on Advanced Visual Interfaces*, pages 188–192, 2004.

[SD02] T. Soukup and I. Davidson. *Visual Data Mining: Techniques and Tools for Data Visualization and Mining*. Wiley, 2002.

[SdSA05] L. Silva, J. Marques de Sa, and L. Alexandre. Neural Network Classification using Shannon's Entropy. In *ESANN*, 2005.

[Sib73] R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[Sil86] B.W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, 1986.

[SK98] T. Seidl and H.P. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 154–165, 1998.

[Ste03] I. Steinwart. Sparseness of Support Vector Machines. *Journal of Machine Learning Research*, 4:1071–1105, 2003.

[SW49] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, Illinois, 1949.

[SZ04] K. Sequeira and M. Zaki. SCHISM: A New Approach for Interesting Subspace Mining. In *Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM)*, pages 186–193, 2004.

[UXKL06] K. Ueno, X. Xi, E. Keogh, and D.J. Lee. Anytime Classification Using the Nearest Neighbor Algorithm with Applications to Stream Mining. In *Proceedings of the 2006 IEEE International Conference on Data Mining (ICDM)*, pages 623–632, 2006.

[VC74] V. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition*. Nauka, Moscow, 1974.

[WF05] I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., 2005.

[YWKT07] Ying Yang, Geoffrey Webb, Kevin Korb, and Kai Ming Ting. Classifying under computational resource constraints: anytime classification using probabilistic estimators. *Machine Learning*, 69(1):35–53, 2007.

[Zhu05] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.

[Zil96] S. Zilberstein. Using anytime algorithms in intelligent systems. *The AI magazine*, 17(3):73–83, 1996.

[ZPAS05] M. Zaki, M. Peters, I. Assent, and T. Seidl. CLICKS: An Effective Algorithm for Mining Subspace Clusters in Categorical Datasets. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 355–356, 2005.

Lebenslauf - Curriculum Vitae

Persönliche Angaben

Dipl. Inform. Ralph Krieger
Am Rollefer Berg 54
52078 Aachen
Telefon: 0241-9908002
Email: krieger@cs.rwth-aachen.de
Geburtsdatum und -ort: 04.01.1973 in Köln
Familienstand: verheiratet, eine Tochter

Schulische Ausbildung

1979-1983	Grundschule Luisenstrasse
1983-1989	Realschule Hugo-Junkers
1989-1992	Gymnasium Mies-van-der Rohe

Grundwehrdienst

1992-1993	Wehrdienst

Studium

1993-2002	Informatikstudium mit Nebenfach Elektrotechnik, an der RWTH Aachen

Berufserfahrung

1999-2002	MICOS GmbH, Large Scale Security Systems (studentische Hilfskraft)
2002-2003	MICOS GmbH, Large Scale Security Systems (Systementwickler)
seit 2003	RWTH Aachen, Lehrstuhl für Informatik 9, Datenmanagement und Exploration, Prof. Dr. Thomas Seidl (Wissenschaftlicher Mitarbeiter)