JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Theory and Implementation of Anticipatory Data Mining

Dissertation zur Erlangung des akademischen Grades

Doctor rerum socialium oeconomicarumque

angefertigt am

Institut für Wirtschaftsinformatik – Data & Knowledge Engineering

Eingereicht von

Dipl.-Wirtsch.-Inf. Mathias Goller

Betreuung

o. Univ.-Prof. Dr. Michael Schrefl
a. Univ.-Prof. Dr. Josef Küng

Linz, Juli 2006

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Dissertation selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe. Diese Dissertation habe ich weder im Inland noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt.

Linz, den 10.7.2006

(Mathias Goller)

Acknowledgements

Thank you very much! Thank you for being interested in the results of my research. As there exist several persons that helped me to bring my ideas into existence, I want to use this page to thank them for their guidance and their support.

First of all, I want to thank my supervisor Michael Schrefl for lots of valuable discussions concerning my approach and alternative approaches. His detailed knowledge about existing work and supervising PhD students helped me to avoid traps. Additionally, a special thank goes to Josef Küng for reviewing my work.

I also want to thank my colleagues at Johannes-Kepler-University Linz for valuable discussions concerning the concepts of my dissertation. Although I could rely on all of them, I want give a special thank to Stefan Berger and Stephan Lechner—Stefan Berger for proof-reading my dissertation and Stephan Lechner for encouraging me when deadlines approached.

I am grateful to my parents for encouraging me in my decision to write a doctoral thesis and to do so in Linz.

Special thanks go to all students which implemented parts of my concept in their master's theses, namely Klaus Stöttinger, Markus Humer, Dominik Fürst, Stefan Schaubschläger, and Julia Messerklinger. I want to thank Dominik Fürst for implementing the kc-method of CHAD and for giving me the clue for a simpler computation of the distance to the bisecting hyperplane. I also want to thank Markus Humer with whom I wrote a paper.

I want to thank the participants of the ECIS doctoral consortium supervised by Virpi Tuunainen and Karl-Heinz Kautz for their valuable feedback. I could benefit of the hints given by Xiaofeng Wang and Jeffrey Gortmaker.

I also want to thank the experts at NCR/Teradata Center of Expertise in Vienna, and in special Karl Krycha, for various valuable discussions concerning data mining in general and my dissertation in special.

(Mathias Goller)

Zusammenfassung

Das Analysieren von Daten mit Hilfe von Data Mining-Methoden ist ein sehr zeitaufwändiger Prozess, besonders dann, wenn die untersuchte Datenmenge sehr groß ist.

Data Mining ist eine wichtige Phase des *Knowledge Discovery in Databases*-Prozesses (*KDD* -Prozess), jedoch nur ein Teil des Prozesses. Verbesserungen der Data Mining-Phase tragen nur teilweise zur Verbesserung des Gesamtprozesses bei, da beispielsweise die Vorbereitungsphase nach wie vor zu langen Prozesslaufzeiten führt. Häufig müssen Vorbereitungsphase und Data Mining-Phase wiederholt werden bis die Ergebnisse der Data Mining-Phase zufriedenstellend sind, was sich wiederum negativ auf die Durchlaufzeit auswirkt.

In dieser Dissertation wird ein neues Verfahren zur Steigerung der Performanz und Qualität des *KDD* -Prozesses vorgestellt. Die Verbesserung beruht auf das Vorbereiten von Zwischenergebnissen, die von keiner konkreten Aufgabenstellung abhängen. Liegt später eine konkrete Aufgabenstellung vor, kann das Ergebnis dieser Aufgabenstellung aus diesen Zwischenergebnissen berechnet werden.

Abstract

Analysing data with data mining techniques needs much time—especially, if the data set that is analysed is very large.

Data mining is an important phase in the *knowledge discovery in databases* process (*KDD* process). Yet, it is only a part of the *KDD* process. Improving data mining also improve the *KDD* process but the improvement can be minor because improving data mining affects only a single phase of a set of phases. Other phases such as the pre-processing phase contribute much to the total time of a *KDD* project. Commonly, it is necessary to iterate the phases pre-preprocessing and data mining before the result of the data mining phase satisfy the analyst's requirements. Again, repeating phases also worsens the performance of total project time.

This dissertation presents a new method to improve performance and quality of the *KDD* process. The idea is to pre-compute intermediate results which depend on no specific setting of any analysis. When the specific setting of an analysis becomes clear, the data mining system can compute the final result of that analysis using the intermediate results.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Introduction

Knowledge discovery in databases (KDD) and data mining offer enterprises and other large organisations a set of methods to discover previously unknown relations among data—which are often called patterns, e.g. [36][73][16]. With these patterns enterprises gain deeper insight into their context and are able to use them for better decision-making.

Frawley, Piatesky-Shapiro and Matheus define *knowledge discovery* as follows: "Knowledge discovery is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data"[73, p. 3]. It is common practise to use the term *knowledge discovery in databases* or short *KDD* instead to stress the most commonly used use case. Yet, it is convenience to use the term *KDD* also for knowledge discovery in data that are not kept in databases but in texts or data streams.

The term *data mining* denotes the step of the *KDD* process in which sophisticated methods analyse the data for patterns of interest [36][16]. The herein used techniques are further called *data mining techniques*. The techniques clustering, classification, and association rule mining are the most-commonly used data mining techniques.

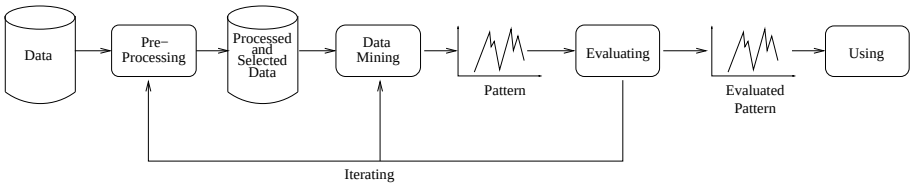


Figure 1.1: KDD process

Quality of patterns found by data mining algorithms and the time needed to find them are the measures of interest in *KDD* projects. The quality of a found pattern determines its usability—for instance, to guide a decision. Additionally, when mining data in large databases, time is an issue, too.

This dissertation focusses on the problem of pattern quality and time of *KDD* projects. Recently, many approaches have been introduced to improve specific *KDD* analyses. However, these approaches try to optimise a single analysis.

In contrast to existing work, this dissertation presents a novel way to improve *KDD* projects in terms of shorter project time and higher quality of results by optimising a set of analyses instead of optimising each analysis individually.

Therefore, the remainder of this section is organised as follows: To be able to show open issues of past work, it first introduces the *KDD* process as is. Later, this section shows where there are open issues of the *KDD* process. Last but not least, this section gives the basic idea of how to improve the *KDD* process in terms of runtime or quality of results.

1.1.1 A First Introduction into the *KDD* Process

This subsection gives a short introduction into the *KDD* process. It gives the information that is necessary to show open issues which is topic of the next subsection. A more detailed discussion of the *KDD* process will follow in Section 2.1.2.

Figure 1.1 shows the *KDD* process with all its phases. The *KDD* process consists of the phases pre-processing, data mining, and evaluating resulting patterns. Using evaluated patterns for decision-making is not part of the *KDD* process but Figure 1.1 contains this item to denote that the *KDD* process does not exist per se but is embedded in a broader economic context.

The first step in a KDD process is selecting data that might contain patterns one is interested in. Some data mining algorithms require data as an input that are consistent with a specific data schema. The *pre-processing* phase consists of all activities that intend to select or transform data according the needs of a data mining analysis.

In the *data mining* phase a data mining expert—further referenced as analyst—analyses the data using sophisticated data mining algorithms. The outcome of these algorithms is a set of patterns that contain potential new knowledge.

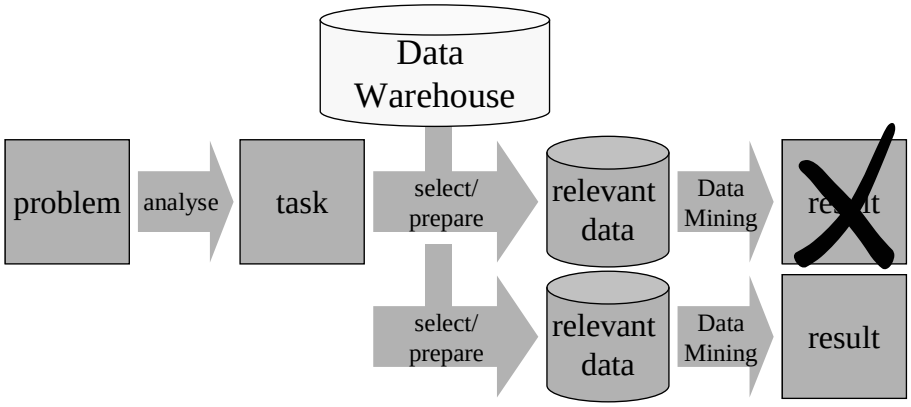


Figure 1.2: Issue A: Insufficient results inflict additional iterations

The resulting patterns must be evaluated before they can be used. This happens during the *evaluating* phase in which an analyst tests whether the patterns are new, valid and interesting or not.

If there are no patterns that fulfill all the above conditions, the KDD process or a part of it has to be repeated once more. The reason why a KDD process has to be iterated can be a bad choice of parameter values or a bad selection of data.

1.1.2 Open Issues of the *KDD* Process

Although there exist approaches that improve the runtime of instances of the *KDD* process, time is still an open issue of the *KDD* process—quality of results is the other one. Hence, we will discuss open issues concerning the runtime of instances of the *KDD* process first before discussing issues related with quality.

Algorithms with long runtime and many iterations during an instance of the *KDD* process increase the runtime of this instance. If the data set which an algorithm is analysing is very large then even the runtime of algorithm of well-scaling algorithms is high. The consequence of a long runtime of an algorithm is that the analyst who started it is slowed down in ones work by the system until the algorithm terminates.

Iterating steps in an instance of the *KDD* process is necessary because the best values of the used algorithm's parameters and the best set of data for a specific analysis are initially unknown. Assuming the opposite would mean that an analyst performs an analysis the result he/she already knows. Yet, that would conflict with the definition of *KDD* that patterns must be new.

Consequentially, it is common practise to run data mining algorithms several times—each time with a different combination of parameter values and data. More precisely, the analyst chooses the data and values that will most likely bring the best results best to one's current knowledge. Even an unsuccessful

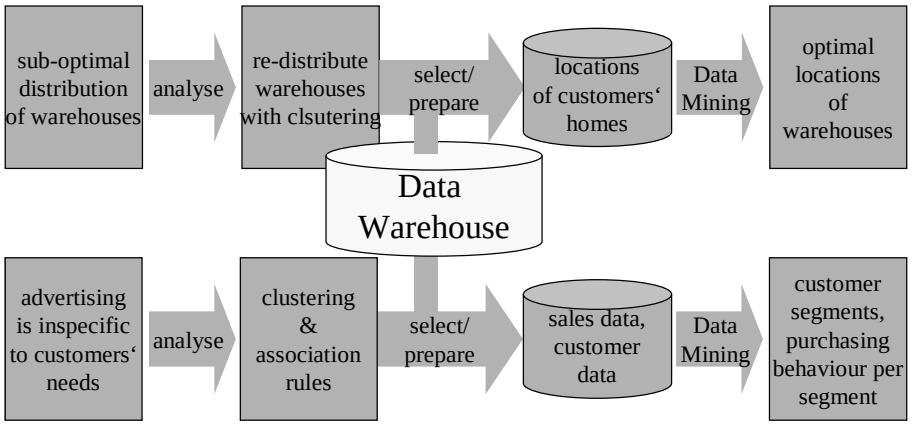


Figure 1.3: Issue B: Multiple isolated *KDD* instances use the same data

attempt commonly gives the analyst a better insight into the relations among the analysed data. Future analyses will be more likely successful. Figure 1.2 depicts an instance of the *KDD* process that requires two iterations.

When an analyst re-does an analysis with different settings, these analyses are not identical but several tasks are very similar. More specifically, both analysts share several intermediate results as later chapters of this dissertation will show. Obviously, processing the same intermediate results more than once means to waste time. Yet, previous work introduces techniques which are able to re-use results of a previous analysis when parameters change. If an analyst uses a different subset of the data he/she must compute all results of this new analysis from scratch.

Two analyses that share the computation of some intermediate results is not limited to analyses of the same instance of the *KDD* process. Moreover, a set of analyses can also share several tasks or parts of them. Especially if these analyses use the same data set and also share the same type of analysis, several intermediate results can be identical. Figure 1.3 shows two instances of the *KDD* process that use the same data set to analyse. Although the goals of both instances are different, both instances need to cluster the same table. Yet, parameter values and used attributes differ. Later chapter will show that is possible to pre-compute common intermediate results for analyses when only attributes and parameters vary.

Re-doing the same steps of analyses would be tolerable if the time needed to compute an analysis is negligible—which is not the case, as illustrated in the next paragraphs.

Pre-processing and most *data mining methods* require accessing all data—which is very time-consuming in large databases exceeding 1GB in size. Hence, algorithms that fail to scale linearly or better in the number of tuples are inapplicable to large data sets.

Although some data mining algorithms like the clustering algorithms BIRCH

[81] and CLARANS [53] are very efficient and scale better in the number of tuples, the minimum time needed for *pre-processing* and *data mining* is still too long for interactively analysing the data. During the tests for this dissertation the Oracle 9i database required few minutes to answer a simple query counting the number of tuples in a table when the size of the table exceeded half a gigabyte—each time the query required accessing values of a specific attribute response time increased significantly to up to ten minutes.

The databases of mail order companies or large warehouses exceed one gigabyte by far. For instance, think of a company operating supermarkets. When there are only 300 small supermarkets in which there are only 600 customers buying 5 items per day and the database requires only 256 byte to store a sold item, the data set that is daily stored requires more than 0.25 GB disc space.

A long runtime of an algorithm or a pre-processing task means that an analyst has to wait longer for a computer to finish its job than he/she can analyse the resulting patterns.

Zhang et alii state that the most time-consuming part of data-mining algorithms is scanning the data while operations in main memory are negligible [81]. This statement is consistent with the test results of this dissertation. As a consequence, approaches of improving data mining algorithms must focus on minimising the number of required database scans.

Avoiding scanning the database at least once is impossible. Even taking a sample requires at least one scan of the data—otherwise, the so-taken sample would not be representative.

Some data mining algorithms already require only the minimum number of one database scan. BIRCH is a hierarchical clustering algorithm requiring only one scan, Bradley et alii have proposed in [59] an EM (expectation maximisation) clustering algorithm, or Kanungo et alii have presented in [40] a k-means-like algorithm, to name only few examples of clustering approaches that require only one scan of the data.

If existing data mining algorithms are nearly as fast as the theoretical limit but even this minimal time is too long for interactive working, the source of major improvements in runtime must be elsewhere. The next subsection gives the basic idea of the approach of this dissertation that is capable to reduce the needed runtime below the current limit. The research question of this dissertation are a direct consequence of this approach.

In many approaches which we will discuss later, one can decrease runtime on behalf of quality of the results of the *KDD* process. Hence, quality is always an additional issue when discussing improvements of runtime. Yet, improving quality and runtime do not exclude each other. Thus, the effect on quality is an additional research question of any approach changing the *KDD* process. Yet, the benefit of improving quality of results depends on the specific instance of the *KDD* process. For instance, if a company can improve the classification of customers who are about to cancel their contracts with the company then each increase in quality, i.e. the accuracy of classification, is beneficial.

1.1.3 Basic Idea of Approach and Research Questions

The *KDD* process as it is shown in Figure 1.1 describes the sequence of steps of a single analysis. Especially, the *KDD* process does not take into account that analysts do several analyses—not necessarily all of them at the same time but many analyses within a year. The discussion of the last subsection indicated that further improvement of analyses is necessary to enable working continuously on data analyses. However, improving a single analysis is improper because the theoretical limit of one scan is not fast enough. Changing this isolated point of view can be a source of major improvement which will be shown later.

The core idea of this dissertation is to re-think the knowledge discovery in databases process. Instances of the KDD process should no longer be regarded as single independent process instances but should be regarded as dependent instances that can profit from each other.

Further, if the specific setting of a future KDD process instance is unknown but several tasks of the instance are known, these known tasks should be done in anticipation.

If two or more *KDD* process instances share the same time-consuming tasks or parts of them then it is reasonable to do those tasks only in the first occurring instance—it would be wasted time to re-do such a task in the second and future instances.

The underlying basic assumption is that there are process instances that have time-consuming tasks in common.

The research questions this dissertation intends to answer are consequents of the above-mentioned basic idea and basic assumption. To be specific, the research questions of this dissertation are as follows:

1. How do *KDD* analyses differ from each other?
2. Are there intermediate results shared by multiple analyses?
3. Can (intermediate) results of an analysis affect the quality of the result of another analysis?

If the variability of analyses is known, it is possible to identify items that are common in all analyses of a specific type of analysis. Additionally, one can identify the range of some parameters. If there are only a few values for a specific parameter the analyst can choose of, one can pre-compute the results for all of these potential values. When the analyst finally chooses the actual value for an analysis, the result has already been computed by the system.

If there are intermediate results of an analysis that other analyses might need than these other analyses can profit of these intermediate results. Hence, the answer to the second research question is needed to improve the *KDD* process:

If common intermediate results are identified, one can pre-compute them in anticipation to save time in succeeding analyses.

Yet, a data set is subject to change, i.e. there will be new, deleted, or updated tuples. Hence, it is necessary to continuously correct pre-computed intermediate results to keep them up-to-date.

The third research question focusses on the aspect of quality of analyses. Taking several analyses into account can also affect the quality of the results of each analysis. For instance, knowledge gained in an analysis can help getting better results in another analysis by biasing the succeeding analysis.

A specific type of analysis needs to perform tasks of a specific type. Some of these tasks depend on parameters of the analyst, others do not. It is possible to pre-compute the results of tasks which are independent of analyst's input.

The data set which the system processes in a specific task is the only differing factor of the same independent tasks of analysis of the same type. As the number of tables in a database is limited, it is possible to pre-compute intermediate results of commonly used types of analyses of all large tables.

However, there still remain tasks depending on parameters which cannot be pre-computed. Thus, it is necessary to split analyses into parts that can be processed anticipatory and tasks that cannot, as illustrated in Figure 1.4. A single instance of a supporting process pre-computes intermediate results for two instances of an altered *KDD* process that consists only of parameter-specific tasks. Later sections will introduce the supported process as parameter-specific *KDD* process and the supporting process as parameter-independent *KDD* process.

The instance of the parameter-independent *KDD* process pre-computes intermediate results and keeps them up-to-date when a table which has been used to pre-process has changed, i.e. there are new, removed, or altered tuples in the table.

The instances of the parameter-specific *KDD* process use the pre-computed intermediate results either to process the final result for higher performance or to bias the computation to receive better results. Improving speed with pre-computed intermediate results corresponds with the second research question, while improving quality by biasing corresponds with the third research question.

The above-mentioned concept of anticipatory pre-computing ideally support analysing data warehouses containing few but very large fact tables. As the combination of data warehouses and *KDD* is a common combination in companies having mass data, this concept offers much potential for many companies.

Some approaches of related work already arise from the same idea of pre-processing but are limited to complete tasks—most of them are tasks of the pre-processing phase. This dissertation introduces an approach that splits tasks into smaller junks and is able to increase the number of pre-computable tasks that way. Additionally, this approach extends the idea of pre-computing to the data mining phase. Hence, it is more broadly applicable than existing work.

This dissertation will present a technique to increase the performance of instance-specific pre-processing tasks to make them so fast that analysts can work on analyses without being slowed down by the data mining system.

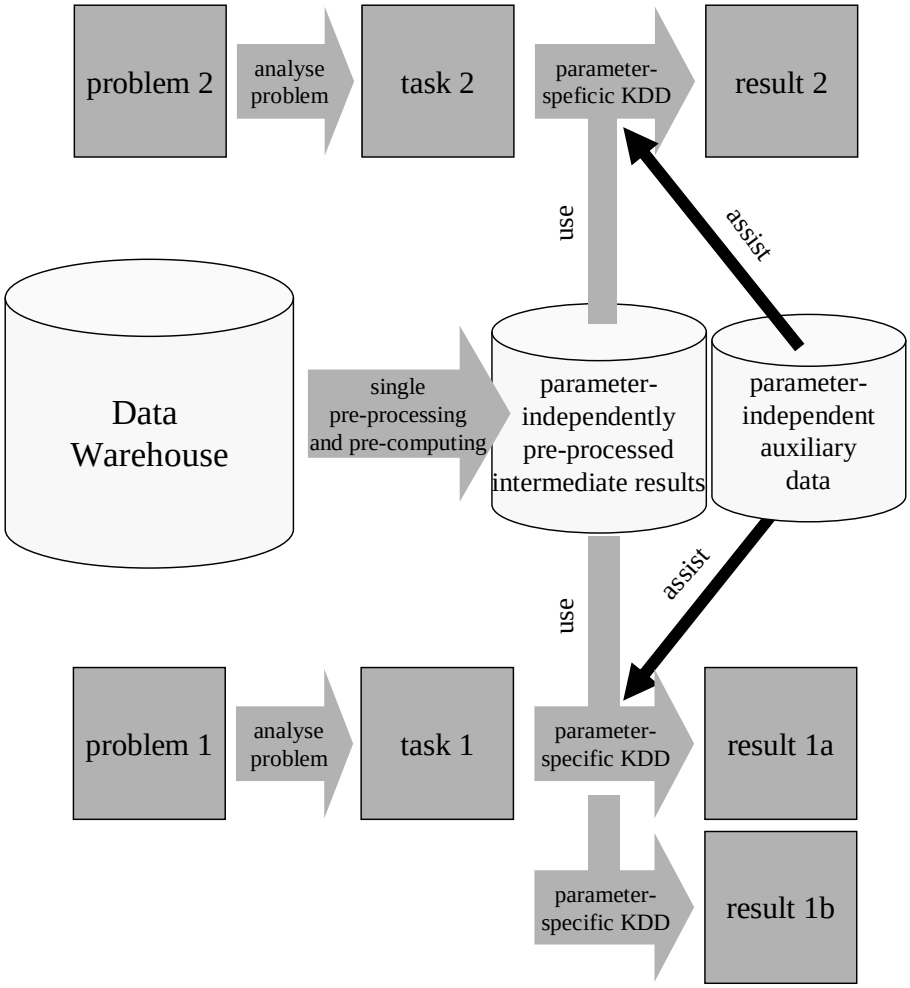


Figure 1.4: Improving the *KDD* process by using pre-computed intermediate results and auxiliary data instead of tuples

In addition to the *pre-processing* phase, the *data mining* phase also has got high potential for improvement.

This dissertation presents techniques to enhance the performance of data mining techniques and intends to improve the *KDD* process in terms of quality and speed that way. For being more specific, it is useful to distinguish the different data mining techniques association rule mining, classification, clustering, and regression at this point.

Association rule mining requires finding frequently occurring sets of items before finding association rules among them. If the frequent item sets that are valid in the set of all transactions are once determined it is possible to determine the frequent item sets that are valid in subsets of all transactions. This has been sufficiently shown in other work. This work will discuss those works for reasons of completeness but will contribute only a minor new idea.

Classification algorithms use a set of data to train a function that is called a classifier which is able to predict the class of a tuple using the tuples' attribute values. The set of data that is used for training typically contains only few tuples—consequentially, scaling algorithms is not a problem. The quality of the resulting classifier is the most concerning issue of classification. This work will discuss quality-improving classification approaches and contribute some new ideas how to improve the quality of a classifier by using auxiliary data gained as by-products of other *KDD* process instances.

Clustering algorithms produce a set of clusters. All clustering algorithms but density-based clustering algorithms represent clusters with describing attributes which vary from algorithm to algorithm. For instance, *EM* clustering algorithms generate a probabilistic model consisting of k random variables and store mean and deviation of each variable, while k-means and related algorithms represent cluster by their euclidian mean. It is easy to compute mean and standard deviation of a cluster, when the number of the cluster's tuples, the linear sum of its tuples and their sum of squares is known, which will be demonstrated later.

If the tuples within a subset of the data are very similar to each other, they will be part of the same cluster in most times. Thus, it is possible to build a subtotal of all of these tuples.

Determining the mean by summing up all tuples or summing up subtotals will produce the same result as long as there are no tuples included that are part of a different cluster. If there are such tuples then the result might become erroneous if specific conditions are met—we will discuss these conditions later. The experimental results have shown that the so-generated error only rarely influences the result of a data mining analysis.

This dissertation discusses approaches improving clustering by aggregating tuples to sub-clusters in detail and enhances techniques to re-use results of a cluster analysis for other cluster analyses such that it is possible to find sub-clusters in anticipation.

Additionally, this dissertation will also discuss other popular approaches improving the performance of analyses such as sampling.

A running example shall illustrate the concepts of this dissertation. Therefore, the next section introduces a running example which is used to demonstrate

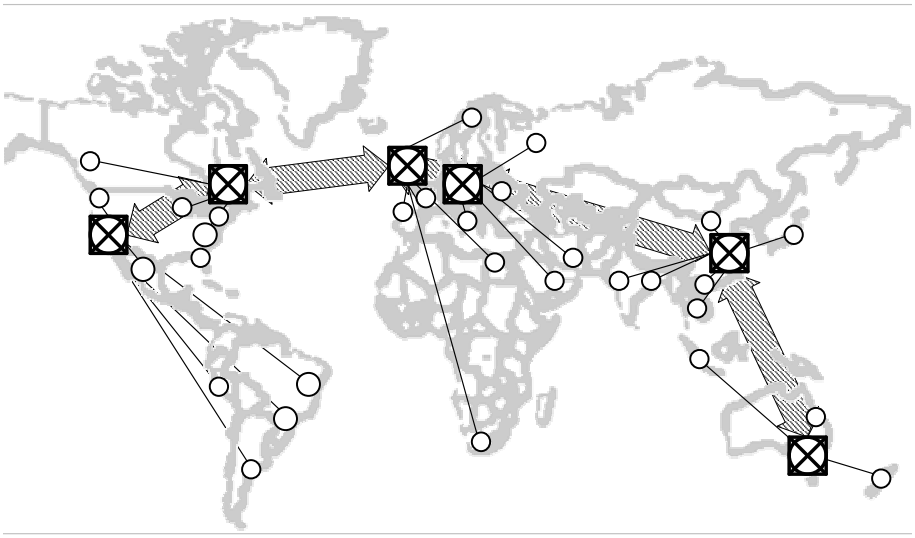


Figure 1.5: network of a globally operating company

more complex parts of this dissertation's concepts.

1.2 Running Example: Sales in a Data Warehouse

For the purpose of illustration this dissertation contains a running example. It is used to demonstrate all concepts presented in this document where it is applicable. Concepts where applying this example does not make sense will be illustrated with their own example—such as the necessary changes for applying the ideas of this dissertation to data streams.

Assume there is a large publishing company maintaining a data warehouse for both strategic and operational decision making. The data warehouse contains a vast set of data mainly concerning the sales of the company's products—books, magazines, journals and on-line articles. Data about transactions with printing companies form another large part of the data warehouse.

The company has optimised its data warehouse for OLAP but not yet for data mining. Later sections of this dissertation will present ways the company can use to improve its performance of *KDD* analyses in terms of quality and speed.

The publishing company is operating globally. Therefore, there are six major locations that administrate the operations of smaller units. Each of these six administration centres operates a data warehouse that contains the data of those units that report to that centre of administration.

The location of the company's headquarter coincides with one of the centres

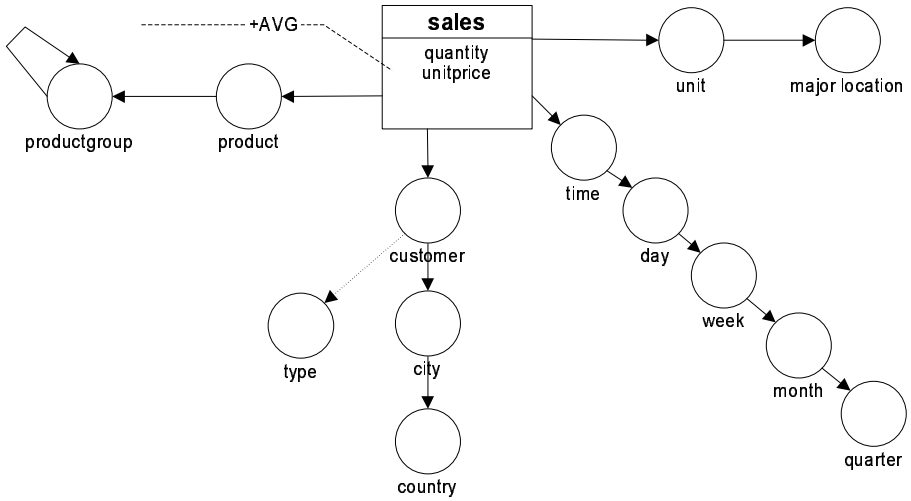


Figure 1.6: schema of cube sales in Golfarelli notation

of administration.

Figure 1.5 shows the distribution of the six major locations, which are symbolised by rectangles with crosses, and their affiliated units, which are represented by circles, around the globe. Arrows and lines denote the amount of traffic of data exchange between locations. We will discuss how to handle distributed data with anticipatory data mining in Subsection 5.2.2.

The global data warehouse and all local data warehouses share the same schema. Each data warehouse contains several cubes but the 'sales' cube is by far the cube having the highest number of tuples—its size exceeds the sum of the sizes of all other cubes. Table 1.1 shows the relational schema of the tables that store the data of the 'sales' cube.

The schema of the 'sales' cube is a starflake schema as it is shown in Figure 1.6. Figure 1.6 uses the notation of data warehouse schemas introduced by Golfarelli in [24]. The 'sales' cube has two measurements, namely the quantity of items sold and the price per sold unit. The cube has the four dimensions *product*, *customer*, *time*, and *unit*.

Products are assigned to product groups which themselves are hierarchically ordered in groups and sub-groups—the number of levels in this hierarchy is potentially unlimited. Hence, dimension *product* has a hierarchy with a recursion.

Dimension *customer* is structured by the location of the customer and alternatively by the type of customer. There are the two types *individual* and *corporate* customer, where the group of corporate customers includes all kind of organisations such as companies but also incorporated societies or non-government organisations. The company delivers books and other printed material only by mail. On-line purchases require a previous registration in which a valid address is required. Thus, there are no anonymous sales.

Relations	sales(units, priceperunit, timestamp, *prid, shopid, cid*) product(prid, name) productgroup(groupid, groupname) isproductofgroup(*prid*, *groupid*) unit(shopid, name, citycode, statecode) customer(cid, name, customertype, citycode, statecode)
Inclusion Dependencies	sales(prid) ⊆ product(prid) isproductofgroup(prid) ⊆ product(prid) isproductofgroup(groupid) ⊆ productgroup(groupid) productgroup(ispartof) ⊆ productgroup(groupid) sales(cid) ⊆ customer(cid)
Legend	relations are shown in the form <relation name>(<attribute list>) attributes of the primary key are underlined, attributes of foreign keys are written in italics

Table 1.1: relational schema of the running example

prid	*name*
1	Theory and Implementation of Anticipatory Data Mining
2	Annual Subscription of KDD Journal

Table 1.2: Sample Entries of Table *product*

Dimension *time* is organised conventionally except years are omitted for simplicity.

A roll-up operation in any dimension causes the quantity measurement being summed-up while an average price per unit is determined.

The data warehouse schema is realised in a set of relations of a relational database. Table 1.1 contains the schema of these relations.

Smaller units daily send their data about sales of that day to the centre of administration they report to. There, a large ETL (Extract-Transform-Load) process inserts these data into the local data warehouse.

The amount of daily arriving data each centre of administration exceeds 1GB.

Beside OLAP-queries the company regularly performs data mining analyses using the data that is kept in the data warehouse. The types of these analyses are two-fold: A lot of these analyses are re-runs of former analyses with updated data to validate their results while others are completely new.

Both analysts residing in the headquarter and local analysts use the contents of the data warehouses for data mining analyses regularly. The analyses of local analysts might require all data stored in all data warehouses or specific parts of

groupid	*groupname*
1	book
2	belletristic
3	scientificmonography
4	article
5	webarticle
6	online journal

Table 1.3: Sample Entries of Table *productgroup*

units	*priceperunit*	*timestamp*	*prid*	*shopid*	*cid*
1	350	2004-01-01 12:00:00	2	1	4711
1	355	2005-01-01 12:00:00	2	1	4711

Table 1.4: Sample Entries of Table *sales*

it. Especially, an analysis is not limited to the content of a local data warehouse.

To be more specific, suppose the following analyses are about to be performed on the current date:

- The number of sale transactions of a specific publication is varying temporally. Many factors such as publication type or reputation of the author influence the demand for a specific publication.

  The publishing company uses multi-variate linear regression method to estimate the current demand of their publications. The resulting regression model predicts the number of purchases for each publication. It is a model of the life-cycle of a given type of publication. As the preferences of customers might change, the company regularly re-constructs the linear regression model to cope with changes in customer purchase behaviour.

- Customers can buy online articles individually or as part of an annual subscription of an online journal. An annual subscription is implemented as a separate product, as shown in Table 1.2.

  The publishing company wants to find a model to predict which subscriber will not extend one's subscription by analysing personal of customers and their purchasing behaviour—stored in relations *customer* and *sales*, respectively. The company pays much attention to those customers because the regular turnover from subscriptions offers high margins. Additionally, retaining unsatisfied customers is known to be generally cheaper than acquiring new customers.

- The marketing division compiles catalogues and hand-outs for specific customer segments that have not been identified, yet.

  An analyst of the company shall find a suitable segmentation of customers. Clustering is his preferred method to complete this task.

1.3 Chosen Research Paradigm and Methodology

The herein-presented approach follows design science as research paradigm. It instantiates the methodology that is suggested in design science literature such as [47], [32], or [43]. This section characterises the paradigm of design research and shows that the pre-requisites for a design research are given. It surveys vital elements of design science and their instantiation in this dissertation.

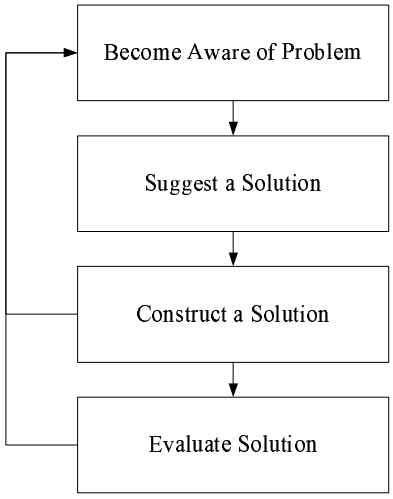
The focus of design science is on creating artefacts which solve specific problems of users. Hence, a problem and an appropriate solution are vital elements of each approach of design science. Thereby, analysing the problem is necessary but not as essential as it is in other sciences such as natural science or behavioural science where understanding the problem with its phenomena is the main focus.

In order to construct an artefact that solves a problem or represents a better solution than existing solutions, the problem to solve must be well-understood or it must be clear what indicates a good solution. In other words, there must exist specific metrics to measure how much an artefact contributes to the improvement of the underlying problem.

The pre-requisites for performing a design science are given for the problem mentioned in the previous section: The *KDD* process is in the scope of research. Section 2.1.2 shows that there is a common conceptualisation of the *KDD* process in literature. The quality of its results and the time needed to run an instance of the *KDD* process represent the problem which are measured by existing quality measures of *KDD* results and runtime, respectively. Hence, solutions of different approaches solving the same problem are comparable with each other.

March and Smith identify four types of artifacts, namely constructs, models, methods, and instantiation [47]. Constructs are artefacts that form a conceptualisation of the domain of a problem. According to March and Smith "a model'is a set of propositions or statements expressing relationships among constructs" [47, p. 256]. Hence, models and constructs express the way we believe or know how things are. Or in other words, they explain the mechanics of a problem and why a solution works. Contrary, methods are artifacts that indicate how to solve a given problem by applying a set of steps. An algorithm is a typical representative of a method, as March and Smith define it. Finally, instantiations are artefacts that utilise the concepts of a design approach. They show that it is possible to solve a problem with a given solution.

All four types of artifacts as defined by March and Smith appear in this dissertation—yet not all of them are stressed equally. Defining constructs properly is needed before constructing a model or a method. Hence, this dissertation includes a chapter to present existing constructs of the *KDD* process. New constructs will be introduced in separate sections when presenting a model or a method, respectively. Finally, an instantiation of the methods is an essential part of this dissertation to show that the herein presented methods work.



adapted from [43, figure 3]

Figure 1.7: Design Cycle

March and Smith mention that conceptualisation are necessary but also can be harmful because it might blind researchers and practitioners to critical issues [47, p. 256].

The basic idea to improve existing methods to analyse data founds on the current model of the *KDD* process. The model of the *KDD* process describes how analysts analyse data in a single project. Yet, the aspect that there are several projects concerning the same source of data is blinded out. Hence, the model of our conceptualisation of *KDD* needs an update to also consider sources of improvement which arise when giving up the isolated view of instances of the *KDD* process.

Hence, the prime resulting artefacts of this dissertation are a set of methods and an updated model of the *KDD* process. As mentioned earlier, the instantiations of these methods aim to proof the effectiveness of the methods. They also justify the update of the model of the *KDD* process.

There exist several presentations of the methodology of design research in literature which differ in detail but share a common structure. For instance, Hevner et al. consider design research as a cyclic search process which consists of the two phases *generate alternatives* and *test alternatives*. Other authors such as Kuechler add several phases to be more detailed. This dissertation uses the design cycle as presented by Kuechler [43]. For a comprehensive survey of research methodology of design research the interested reader is referred to [43].

Figure 1.7 depicts the research methodology of design research which is also the methodology this dissertation uses.

Becoming aware of a problem is necessary for designing any solution to it. This phase includes understanding the problem with all its facettes including

the strengths and weaknesses of solutions that are already available. For design research that is intending to present new models or methods it is necessary to understand the affiliated constructs and the existing model of that problem.

Extensive study of literature and various discussions with data mining practitioners and researchers were the used methods to become aware of the problem of this dissertation. The next chapter surveys the current model of the problem and related constructs.

Suggesting a solution is the phase in which the researcher abductively generates the idea to a new solution. Although experience with previous designs and techniques supporting creativity might help the researcher, this is a very creative task.

In the phase *suggest a solution* of this dissertation intensive study of the model of the *KDD* process and re-thinking all its elements, as well as several discussions with other researchers, and the interaction with an experimental prototype assisted the author of this dissertation to find a good solution.

Constructing a solution once it is suggested is an essential part of design research. This phase includes the creation of all artefacts which are part of a solution. Engineering techniques assist this process. As in this dissertation the resulting artefact is a model of a process and an algorithm, this phase is very similar to a project of business process modeling or a software project—including the techniques to assist specific phases. Yet, evaluation of solution is different in design research and and a software project, for instance.

Evaluating the result of the construction phase is primarily necessary to prove that the suggested solution indeed solves the problem. If there exist other solutions to the same problem, the new solution should solve the problem better according to at least a specific metric. Yet, metrics can be multi-fold: For instance, a solution could be better because it is more broadly applicable than another solution. However, a third solution could be better because it solves a special case of the problem better than a more general solution.

Additionally, constructing and evaluating a solution gives valuable insight into the problem and ways to solve it. Hence, both phases are viable sources of gaining knowledge.

This dissertation utilises several methods of design evaluation which are listed in [32, table 2]. If it is possible to show a property of an artefact analytically, then static analysis is the method of choice. For instance, upper bounds for potential errors are shown analytically. If otherwise it is impossible to show a property of an artefact analytically, then this dissertation uses a controlled experiment.

As seen in the figure 1.7, research design is a process which goes in cycles. As mentioned above, constructing and evaluating a solution gives insight into the problem and ways to solve it—which means in other words that one becomes aware of other aspects of the problem or the solution, respectively. Modifications of the suggested solution and re-constructing parts of artefacts might be the consequence.

1.4 Organisation of this Dissertation

The remainder of this document is organised as follows:

Chapter 2 gives an introduction into the KDD process and the most commonly used data mining methods. To be more specific, it sketches the general principles of these methods. Knowing these principles is necessary for understanding how anticipatory data mining can improve these methods. The intended reader of this chapter is novice or intermediate in knowledge discovery in databases and data mining, respectively.

Chapter 3 shows previous work trying to improve the KDD process by making existing algorithms scalable or using any kind of re-use.

Chapter 4 introduces a novel method to improve the KDD process by splitting the KDD process in two processes: a process that performs user-defined analyses and a supportive process that does most of the preparatory and mining tasks in anticipation.

Chapter 5 describes the concepts of CHAD—a new framework for partitioning clustering. CHAD is short for Clustering Hierarchically Aggregated Data and gives a clue to the main reason why it is so efficient as it is shown in the test chapter. CHAD also implements the majority of concepts shown in the previous chapter.

Chapter 6 shows how to implement the concepts that CHAD has left out such as using pre-computed statistics and pre-selected tuples for association rule analysis or classification.

Chapter 7 contains the results of the test series that tested each feature of CHAD and the other concepts presented in chapter 6. For clarity of description the chapter only summarises the most important results. The interested reader can find all details of each test in the appendix.

Chapter 2

Analysing the *KDD* Process and Existing Algorithms

Contents

2.1 Knowledge Discovery in Databases and Data Mining

Before presenting a solution that is capable to solve a specific problem, it is necessary to become aware of that problem. Moreover, one needs to understand the context of the problem and must know existing solutions.

Hence, this chapter gives an overview of knowledge discovery in databases or *KDD* in short. Thereby, this section introduces the general process of projects that analyse data for interesting patterns. Subsequent sections survey specific techniques used in this process.

To be more specific, this section defines the vocabulary used in the *KDD* community and sketches the *KDD* process.

Due to *KDD* being an interdisciplinary field of research, there exist items of interest denoted by more than one term. Research disciplines such as statistics, machine learning, and databases have their own terms for the same object of interest. For instance, what the database community considers as a *tuple* is an *observation* in statistics and an *instance* in the machine learning community, respectively.

For a high level of understandability, all chapters use the terminology of the database community. Especially, there is no mix of terms originating from different research disciplines. However, if a term of another discipline which is not databases is the only common term, then chapters use this term to comply with the terminology of the *KDD* research community.

2.1.1 Definition of Knowledge Discovery in Databases

Frawley, Piatesky-Shapiro and Matheus define *knowledge discovery* as follows: "Knowledge discovery is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data"[73, p. 3].

Han presents the following definition for *data mining* in the slides to his textbook [36]: "Data mining is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data". The textbook is free of this definition. According to Han and Kamber in [36], data mining is part the *KDD* process. Yet, several authors cite that textbook with the above-mentioned definition of Han causing conceptual confusion.

This dissertation follows the definition of Frawley, Piatetsky-Shapiro and Matheus because it describes the concepts of knowledge discovery in a clear way and is consistent with the conception of the *KDD* and data mining research community. However, Piatetsky-Shapiro et al. use the term *information* without differentiating data, information, and knowledge. Especially, it is common understanding within the school of which the author of this dissertation is member of that machines process data but humans are able to understand information. Therefore, we discuss each item of the definition and indicate where this dissertation differs from the definition of Piatetsky-Shapiro et al.

nontrivial extraction The criterion *nontrivial extraction* excludes other methods of data analysis such as descriptive statistics methods or OLAP, respectively, which require a human user sophistically using these methods while methods themselves are rather simple.

implicit information The criterion *implicit information* means that the methods of *KDD* use implicit information to derive patterns. Hence, this criterion excludes some sophisticated methods of artificial intelligence that use an explicit knowledge base to derive new knowledge in the form of new facts, e.g. by logical deduction.

previously unknown information It is self-evident that discovering knowledge means to seek unknown patterns in data. Yet, distinguishing automatically between known and unknown information is a very hard problem. Making all the user's knowledge explicit is impractical due to sheer hugeness of individual and organisational knowledge.

Previously unknown but obvious information is a closely-related issue in *KDD* . It is obvious that a typical user calling a company's help desk for installation instructions has previously bought a product although potentially none of that company might ever have thought about this before.

Usually a human expert determines whether a result of a *KDD* process instance is a new piece of information or not. For some *KDD* applications that commonly return many results such as association rule analysis there exist criteria to determine interestingness or triviality of a result. This issue will be discussed later when presenting the specific data mining techniques.

potentially useful information The criterion *potentially useful information* requires a human user that must be interested in the outcome of a *KDD* process instance. Consequentially there must be a user that is able to judge the interestingness of the outcome of the *KDD* process.

2.1.2 The *KDD* process

The *KDD* process is presented in different ways in the literature but there are many things in common. Thus, this subsection presents the most-cited models of the *KDD* process in literature first and discusses common elements later.

We choose to present the models of the *KDD* process of most-cited textbooks due to their high influence on the common sense of the *KDD* research community.

As the introductory section has sufficiently discussed the aspect of iterating the *KDD* process, the subsequent subsections present only the typical sequential path of tasks—where each subsection presents a single model of the *KDD* process.

Note that the model used in this dissertation is the model that expresses the relations of the constructs of knowledge discovery in databases as is, as described in Section 1.3. This model needs an update to reflect the additional aspects on which the approach of this dissertation founds on, as also mentioned in Section 1.3. To be more specific, it focusses only on independent instances. It is the model that the author of this dissertation had in mind when he became aware of the problem and analysed existing solutions. Adding dependency of instances to the model of the *KDD* process offers designing a new improved artefact—a method that performs analyses significantly better in terms of speed or quality.

KDD process according to Han and Kamber

Han and Kamber present a process schema of the *KDD* process consisting of seven phases which are *data cleaning*, *data integration*, *data selection*, *data transformation*, *data mining*, *pattern evaluation*, and *knowledge presentation*[36, p 6f].

Data cleaning is required to handle erroneous data to receive valid results, i.e. to correct inconsistencies in data, to fill missing values with correct values, or simply to remove erroneous data that cannot be corrected.

Data integration is necessary if the analysed data is stored in different data sources such as tables in a distributed database.

It is common practise to do data cleaning and data integration separately[36, p 7] because data cleaning and data integration is required by many different *KDD* instances. The approach presented in this dissertation extends the idea of performing cleaning and integration once for multiple instances by far because this dissertation's approach also includes later phases of the *KDD* process.

Data selection means to select the subset of data that is relevant in respect to the purpose of the current instance of the *KDD* process. This implicitly requires the existence of a purpose or a goal of the process such as "Finding the reasons for canceling contracts to prevent customer churn".

Data transformation is the phase of transforming the selected data to the needs of the *KDD* instance. For instance, algorithms that are intended to be used in the data mining phase might require ordinal attributes instead of continuous ones—making the construction of intervals necessary. Aggregating data is also necessary when data is needed at a different level of aggregation than the data stored in a table, e.g. an analysis needs total sales per customer but sales are stored per individual purchasing event.

Data mining is the phase in which sophisticated algorithms scan the selected and transformed data for patterns. The algorithm influences the type of patterns that are found. Thus, the succeeding sections of this chapter show the different types of algorithms and their types of pattern.

Pattern evaluation and knowledge presentation. Analysing the results happens in the phases *pattern evaluation* and *knowledge presentation*. According to Han and Kamber evaluating patterns happens automatically by determining interestingness according some quality measures. Contrary, knowledge presentation is the process of presenting the so-found interesting patterns to the user. Other authors disagree with Han and Kamber that pattern evaluation is an automatic process, as is shown in the following subsections.

KDD process according to Ester and Sander

Ester and Sander presented a textbook [16] with great influence on the German-speaking *KDD* research community. As this book is written in German its accessibility is limited. When presenting the *KDD* process of Ester and Sander this section uses translated versions of the terms they use. The original German terms are postponed in brackets to enable the reader that also is capable to understand German to reproduce the translation.

Focussing [Fokussieren] is the initial phase of the *KDD* process according Ester and Sander. In this phase the analyst becomes aware of the problem he/she wants to analyse in a *KDD* project. The analyst defines goals of the instance of the *KDD* process. This phase also includes the choice of data sets the analysis of which might contribute to fulfill the goals of this instance of the *KDD* process.

Pre-Processing [Vorverarbeitung] Once the data sets are selected that the analysts wants to analyse, he/she must pre-process the data to remove or correct inconsistencies.

Transformation [Transformation] Ester and Sander distinguish in pre-processing data due to erroneous data and pre-processing data due to requirements of the chosen data mining algorithm. They call the first type of pre-processing *pre-processing* while they call the latter type *transformation*. Data mining algorithm typically require data having a specific

scale. For instance, several clustering algorithms need data having ordinal or rational scale. Algorithms mining association rules need categorical data to search for interesting rules. If the scale of the data has the wrong scale for a specific type of analysis, then specific operations can transform the data to change scale.

The description of the *KDD* process of Ester and Sander also contains the phases *data mining [Data Mining]* and *evaluation [Evaluation]*. As Ester and Sander's definitions of these phases are equivalent in their meaning to the according definitions of Han and Kamber, we omit to discuss them here once more.

KDD process used in this work

This subsection summarises the different aspects of the *KDD* process in order to find a model of the *KDD* process that represents the common understanding concerning the *KDD* process. This common understanding is necessary for finding improved solution which are broadly accepted.

Although the above-presented variations of the *KDD* process have different number of phases, the sequences of tasks that we receive when we split each phase in its sets of tasks are very similar. Han and Kamber present a *KDD* process that consists of more phases than the *KDD* process as Ester and Sander present it.

In both descriptions of the *KDD* process the phases *data mining* and *evaluation* are common elements. They also agree about the necessity of pre-processing the data before analysis. Hence, the model of the *KDD* process used in this dissertation contains the phases *data mining* and *evaluation* which have the same meaning as presented by Han and Kamber, and Ester and Sander, respectively.

Yet, the first phases of both descriptions of the *KDD* process focus on different aspects of the *KDD* process. Han and Kamber stress the common usage of data warehousing and data mining, i.e. several tasks of data warehousing such as data cleaning pre-process the data before selecting them for analysis. Contrary, Ester and Sander emphasise the need to plan the analysis as a project including defining goals and selecting appropriate methods.

As both aspects, defining goals of the process and using a data warehouse, are relevant aspects, the model of the *KDD* process used in this dissertation includes both aspects.

Becoming aware of a problem and defining goals is necessary to be able to evaluate the results properly, as shown in section 1.3.

Using a data warehouse for analysis has many positive aspects on knowledge discovery. Analysing data is time-consuming. As a consequence, if the database system is fully utilised because it scans a huge part of the database for a specific analysis, then this analysis hinders other users to work interactively with the database system. Hence, separating data for analysis and data for accessing them due to regular business processes is necessary due to efficiency reasons. The *Extract, Transform, Load*-cycle of a data warehouse loads data from other

databases into a data warehouse. Thereby, it performs several tasks which otherwise one would have to do during an instance of the *KDD* process. Correcting errors is the most-relevant example of these tasks.

Using a data warehouse means to pre-process data in anticipation. Therefore, it is an analogous approach to the approach of this dissertation. However, the approach of this dissertation includes much more tasks that can be prepared in anticipation. Especially, it also includes parts of the data mining phase—which is the reason we call the approach *anticipatory data mining*.

Thus, we summarise the model of the *KDD* process as listed below. Note that this model comprises the common understanding of the *KDD* process as is. Especially, it is not the model of the *KDD* process of anticipatory data mining as there is no aspect of anticipation in it. Yet, it is the model that we want to improve with the concept of anticipatory data mining.

Integrating data in a data warehouse loads the data from various data sources into a data warehouse to analyse them later. This task also includes operations cleaning the data on loading.

Focussing on a problem means that an analyst plans an analysis. Thereby, one analyses the problem that is about to solve. Additionally, one chooses data sources that contain potential answers to questions related with that problem. Finally, the analyst chooses data mining techniques that are capable to deliver that kind of answers.

Pre-processing includes all tasks that are necessary to bring the data that should be analysed in a state in which a data mining algorithm is able to analyse them. These tasks include data cleaning—if this has not happened when integrating the data—and re-scaling data.

Data mining and pattern evaluating are again identical with the according phases of Han and Kamber, and Ester and Sander, respectively.

The remaining sections of this chapter survey data mining techniques. There exist few basic types of data mining techniques and a large set of techniques for special applications. This chapter presents basic types only as they are the most commonly used techniques. Using the approach of this dissertation improves all basic types of data mining techniques either in terms of speed or quality. Additionally, many techniques for special applications are derived from a basic technique or a combination of basic techniques.

2.2 The Clustering Problem

Clustering is a process that groups a set of objects into several subsets which are called clusters. The goal of the clustering process is to define the clusters in a way that each object and the object that is not more similar to another object are member of the same cluster. In other words, objects within a cluster should

be most similar while objects of different clusters should be most different to each other.

There are many purposes for performing clustering such as data reduction or hypothesis generation. Subsection 2.2.1 describes the different purposes of clustering.

There are several categories of clustering algorithms including partitioning clustering algorithms, hierarchical clustering algorithms, density-based clustering algorithms, or grid-based clustering algorithms. This dissertation sketches partitioning clustering algorithms and hierarchical clustering algorithms in the succeeding subsections because the clustering algorithm presented in chapter 5 is a combined algorithm consisting of a partitioning clustering algorithm and a hierarchical clustering algorithm.

This section concludes with a subsection discussing quality measure of clustering. Especially, that subsection focusses on the quality measures used in the experiments of of the approach of this dissertation.

2.2.1 Purposes of Clustering

Clustering for Data Reduction

Clustering algorithms scan a set of data and return a set of clusters. The set of data is typically very large in size while the number of clusters is small. Using the circumstance that most clustering algorithms do not store the data that are part of a cluster but a small set of describing features of each cluster, it is possible to use the set of features of a cluster instead of the cluster's data. In other words, the features that characterise a cluster are a condensed representation of a subset of data. The features of all clusters form a compact representation of all data.

Several typical applications profit of the data reducing effect of clustering. Customer segmentation is only one of them. The aim of customer segmentation is to group the set of all customers of a company into few sets of customers that are represented by characteristic features—e.g. by an "average customer".

Data reduction is a necessary pre-processing step whenever there is a large set of unique but similar objects that shall be used for data mining. Data mining techniques try to find frequently occurring patterns within the data, Yet, it is impossible to find frequently occurring patterns if each instance is unique because unique items can never be frequent. The usage of clustering can solve this problem. Before being clustered a specific tuple cannot be used for data mining because it is unique but after being clustered the same tuple can be used for data mining when it is treated as a an element of a cluster—there might be many tuples being element of the the same cluster while each tuple is unique. Hence, it is possible to find frequent patterns.

A simple example shall illustrate the necessity of clustering for pre-processing mentioned above:

Suppose a mail order company wants to analyse which costumers buy which kind of products. The company stores sales data and customer data in the two

tables "sales" and "customer", respectively, of a relational database system. If the company performs an association rule analysis on the "sales" table it receives rules of the form "If a customer buys the set of products 'A', he/she will probably buy the set of products 'B', too". Yet, it is impossible to tell what kind of products a potential customer that has never bought anything is interested in. The customer must have bought at least a single item to guess what other good he/she might be interested in.

Assuming that two customers that are in a similar living condition—i.e. they have the same sex, similar age and similar educational and cultural background, also share some of their interests concerning specific type of products. Yet, the tuples representing customers in the "customer" table are unique combinations of attribute values in most times—i.e. usually, no two customers have same name, sex, birthday and address. But when clustering the data in the "customer" table, the company receives a set of typical customers, e.g. the female teenager. The resulting clusters can be combined with the result of the association rule analysis. By doing so, the company is now able to predict the set of products a potential customer is interested in. The company might use this knowledge for several purposes such as marketing events customised to the interests of specific customer segments, e.g. enclosing a catalogue with books that are primarily read by female teenagers into a girls' magazine.

Clustering for Hypothesis Generation

Some features describing a cluster are suited for being interpreted as a statistical hypothesis. Assume that the application of a partitioning clustering algorithm, see Section 2.2.3 for details, such as k-means has returned three clusters with means $\vec{\mu}_1, \vec{\mu}_2$, and $\vec{\mu}_3$. k-means is a partitioning clustering algorithm that partitions a set of data in k disjoint subsets and returns the mean of each subset—k is a user-given parameter. When the best value of parameter k is unknown, k-means is iterated several times to determine the best value of parameter k. A potential hypothesis is that there are three independent statistical variables $\vec{X}_1, \vec{X}_2$, and $\vec{X}_3$ with means $\vec{\mu}_1, \vec{\mu}_2$, and $\vec{\mu}_3$. Frequent co-occurrences of attribute values can be source of hypothesis, too. For instance, if the mean vector $\vec{\mu}_1$ has the dimensions "income" and "total sales per month", then a small value of the mean in dimension "income" and a high value of the mean in dimension "total sales per month" means that persons with low own income buy a lot of things.

It is common practise in statistics to generate a hypothesis before testing whether data supports it or not. Otherwise, the hypotheses would be trimmed to fit a specific data set that might cause the so-called overfitting problem. Overfitting occurs when the underlying statistical model of a hypothesis fits almost exactly to the data that has been used to generate that model but only poorly fits to other data having the same schema.

When clustering is used for hypothesis generation, it returns those hypotheses that best fit the given data—which might cause overfitting. Hence, it is necessary to validate the resulting hypotheses by testing each hypothesis with a set of data that has not been used for clustering. For instance, when one has

used the data of the past to determine a set of hypotheses, one can use current data in order to try to falsify those hypotheses.

Clustering to Improve Quality of Other Techniques

A basic assumption of clustering is that objects in a cluster are similar in structure and behaviour—or, if not, then the behaviour and structure of objects in a cluster are at least more similar to each other than an object of that cluster is to an object of another cluster.

According to this basic assumption it is valid to assume the same distribution of objects in a cluster.

Partitioning a data set in several subsets with similar distribution reduces the expected error of classification. Assume that attribute Y represents the affinity to a specific class. Further, let X denote an attribute that is used to predict the class attribute Y. The ranges of both attributes are arbitrary—they might be discrete or continuous, limited or unlimited. Then, a classifier is a deterministic function $f : X \to Y$ that assigns each value of attribute X an according value of attribute Y. Thus, a classifier assumes a deterministic association between the classifying attribute X and the class attribute Y.

Yet, the association between classifying attribute and class attribute is rarely deterministic but uncertain in most times. One can use a stochastic model to express the degree of this uncertainty. Assume that the statistical variable Θ represents external influences on the association between attributes X and Y. Thus, it summarises all non-observable influences that determine the class of a tuple.

The probability function of of the triple (X, Y, Θ) represents the association between class, classifying attributes, and unknown parameters. The outcome of this function denotes how likely it is to have a tuple $(x, y) \in X \times Y$ in a data set. Thus, one can determine the likelihood that a tuple with attribute value x is part of class y using the probability function.

Yet, the distribution of the probability function can be very complex. Moreover, the distribution can be an arbitrary function and no known function such as binomial or normal distribution. Thus, a classifier can only approximate it.

If one partitions the data set into several clusters which are very similar, then approximating is more promising because one can assume the same distribution for all tuples of a cluster. Or, one can search approximations for each cluster individually—for instance, the best approximation for a the probability function of a cluster might be binomial distribution while the best one of another cluster might be uniform distribution. As these clusters have only a small deviation compared with the total data set, the deviation of a classifier is also smaller. Yet, the lower the deviation of a classifier is the lower is the error. For instance, assume that the deviation of a class y is σ. As clustering decreases the deviation of attributes in clusters because it groups those tuples into the same cluster which are similar to each other, the deviation σ' of class y is typically smaller, too. Decreasing deviation is no necessary condition but a commonly-observed phenomenon.

A classification algorithm that has a smaller deviation of error than another classification algorithm is superior to this other classification algorithm in terms of quality.

If there is a clustering of a data set that significantly reduces the distances within a cluster, then clustering can improve quality of classification algorithms classifying the data set cluster by cluster.

2.2.2 Expressing Dissimilarity with Distance Function or Distance Matrix

A clustering algorithm must be able to determine the similarity of tuples to determine the cluster a tuple is most similar with. Therefore, some kind of measure is required that indicates similarity or dissimilarity of tuples.

Hence, this section introduces distances, distance functions, and distance matrices as measures to express dissimilarity of tuples.

It is common practise to express dissimilarity of tuples with the so-called distance of tuples. The distance between two tuples can be a distance having a meaning but can also be only a fictive measurement. For instance, if two persons earn €10 and €30 per hour, one person earns €20 more than the other. But if three persons have the values 1.2, 1.4, and 2.0 as values of an artificial index, we can only determine that the first two persons are more similar to each other than to the last one.

For clustering tuples, there must exist a measurement of distances for each attribute and a way to combine distances of different attributes. Distance functions fulfill the function of combining distances of several continuously and interval scaled attributes to a common distance. Distance matrices are used in cases where it is impossible to find a suitable distance function. See below for details.

Popular Distance Functions

A distance function is a function that takes two tuples as its input and returns a float variable—the distance—that indicates the dissimilarity of both tuples.

The Euclidian distance, the Manhattan distance, and the Minkowski distance are popular distance functions for interval-scaled attributes.

The Euclidian distance d_E of two tuples is the length of the distance vector of the vectors representing both tuples in a Euclidian vector space, i.e. if two tuples are represented by the vectors $\vec{x}$ and $\vec{y}$ the Euclidian distance is the length of the vector $\vec{x} - \vec{y}$ which is

$$d_E(\vec{x}, \vec{y}) = ||\vec{x} - \vec{y}|| = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}.$$

The Manhattan distance dM is the sum of the absolute values of the differences

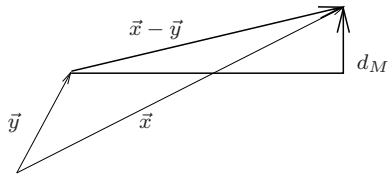


Figure 2.1: Euclidian distance $||\vec{x} - \vec{y}||$ and Manhattan d_M distance of two vectors $\vec{x}$ and $\vec{y}$

of the tuples in each attribute, i.e.

$$d_M(\vec{x}, \vec{y}) = \sum_{i=1}^{d} |x_i - y_i|.$$

The name Manhattan distance originates from the way roads in Manhattan are organised: Streets follow East-West direction while avenues run from North to South. In analogy to that, to get from point $\vec{x}$ to point $\vec{y}$ one has to pass several blocks in one direction before changing the direction orthogonally. The Euclidian distance is the distance taking the straight way from point $\vec{x}$ to point $\vec{y}$, as shown in figure 2.1.

The Minkowski distance is the generalised distance function of Manhattan distance and Euclidian distance. The Minkowski distance has a parameter p that determines the exponent of the difference of the attribute values of the tuples, i.e.

$$d_p(\vec{x}, \vec{y}) = \sqrt[p]{\sum_{i=1}^{d} |x_i - y_i|^p}.$$

The Manhattan distance is a Minkowski distance with parameter $p = 1$. The Euclidian distance is a Minkowski distance with parameter $p = 2$.

The specific requirements of the *KDD* process instance determines which distance function is the one to prefer. The Euclidian distance will be the most-appropriate distance function when analysing geographical data because it represents the real geographical distance between two points. The Euclidian distance is also chosen for many other technical applications with metric data only.

The Manhattan distance is applicable when the Euclidian distance is applicable but the Manhattan distance favours changes in only a single attribute. Let vector $\vec{x}$ assume $(0, 0)$ and vector $\vec{y}$ assume $(2, 2)$. Further let vector $\vec{z}$ assume $(0, 3)$. According to the Manhattan distance, vector $\vec{z}$ is nearer to vector $\vec{x}$ than vector $\vec{y}$ is to vector $\vec{x}$, i.e. a distance of 3 compared to a distance of 4. According to the Euclidian distance, vector $\vec{y}$ is the nearest vector to vector $\vec{x}$ because the distances are $d_E(\vec{x}, \vec{y}) = 2\sqrt{2} \approx 2.8$ and $d_E(\vec{x}, \vec{z}) = 3$, respectively.

The Manhattan distance is also applicable to data sets with ordinal attributes. In such a case, the Manhattan distance of two tuples x and y is the

attribute value	a_1	a_2	a_3
b_1			
b_2			y
b_3	x	$\rightarrow$	$\uparrow$

tuples $x = (a_1, b_3)$, $y = (a_3, b_2)$
Manhattan distance $d_M(x, y) = 3$

Figure 2.2: Manhattan distance of ordinal attributes

minimal number of steps to take to get from the attribute value combination of x to the attribute value combination of y in the matrix of all potential attribute value combinations, as shown in Figure 2.2.

For comparing objects that include a non-empty set of objects, there exists a distance function that is computed as the number of different items in relation to the number of items in both sets, i.e.

$$d_S(X, Y) = \frac{||(X \cup Y) - (X \cap Y)||}{||X \cup Y||}.$$

Distance function d_S can be used to cluster transactions of arbitrary length—what is needed when pre-clustering or post-clustering of an association rule analysis is required.

The distance d_S is also an appropriate distance function for data sets with categorical attributes because a tuple of a data set with categorical attributes can be interpreted as a set of attribute values with a fixed size of items. If we omit the ordering of attribute values in Figure 2.2 we receive a relation with two categorical attributes. Assuming that tuple X assumes (a_1, b_2) and tuple Y assumes (a_2, b_2), the distance d_S equals $d_S = \frac{||\{a_1,a_2\}||}{||\{a_1,a_2,b_2\}||} = 2/3$.

The range of distance d_S is $[0, 1]$ where 0 is the resulting value if and only if both compared sets are identical. Contrary, 1 is the resulting distance if and only if no element of one set is element of the other set.

Distance Matrix

A distance matrix can replace a distance function in cases where there is no appropriate distance function available or computing the result of a distance function would be too expensive. For instance, let the attribute "job" be an attribute of interest in table "person". It is impossible to find a distance function that determines the distance of two different jobs because similarity of jobs is subjective. One person might think that "nurse" and "doctor" are closely-related jobs because nurses and a lot of doctors work together in hospitals. Contrary, another person might think that "advocate" and "doctor" are more closely-related jobs because both jobs require a university degree.

Although it is impossible to give a distance function that represents the similarity a user associates with some pair of objects—e.g. the similarity of

jobs—it is possible to express the user's association of similarity of objects with a distance matrix.

A distance matrix is a matrix that includes the distances of all combinations of tuples, i.e. in the "jobs"-example a distance matrix has an element for each combination of two jobs that indicates how much related the user assumes that these jobs are.

Note that the resulting distance matrix expresses a subjective opinion of a user or a group of users—if they agree with the distance values stored in the distance matrix. Consequentially, the result of a clustering using such a distance matrix would be a subjective result. too. Thus, it is important to present the distance matrix together with the results to prevent users from taking those results for granted.

Using a distance matrix is limited to finite sets of tuples because it must include all combinations of tuples—infinite sets of tuples would require a distance matrix with infinite space. Hence, continuously scaled attributes and ordinal-scaled attributes with unbounded domain prevent the usage of distance matrices.

Using Normalisation When Attributes Vary Significantly in Range

Especially in cases with one attribute having a much broader range than the other ones, the attribute with the broader range dominates the result of the clustering because its distances are higher than the distances of other attributes.

Normalisation of attributes is used to prevent that the attribute with the broadest range dominates the clustering. Normalisation of attributes is a transformation of an attribute so that the ranges of all transformed attributes are approximately identical.

Z-normalisation or z-scoring is a popular way of attribute normalisation. The z-score z of an attribute value is the transformed value we receive by subtracting the mean μ of the attribute of the attribute value first and divide the result of the substraction by the standard deviation σ of that attribute.

$$z := \frac{x - \mu}{\sigma} \tag{2.1}$$

The succeeding subsections introduce partitioning clustering algorithms and hierarchical clustering algorithms which both are used in CHAD, a clustering algorithm that uses two types of clustering algorithms to receive a higher overall performance.

2.2.3 Partitioning Clustering Algorithms

This section gives a brief overview of the characteristics of partitioning clustering algorithms. Yet as the following paragraphs will show, the process of conceptual clarification is still unfinished. The following survey of partitioning clustering algorithms will point out unclear concepts.

Partitioning clustering algorithms partition a set of tuples in several subsets of tuples, i.e. clusters. The number of clusters to be found typically is a user-given parameter—although there exist approaches to determine the optimal number of clusters automatically such as [75]. Typical partitioning clustering algorithms are k-means [46], CLARANS [53], and k-modes [34].

Partitioning clustering algorithm try to find partitions according to a given criterion of optimality. Minimising the sum of squares distances to the clusters' centre is a popular criterion that partitioning clustering algorithms use. k-means is one of them, see Section 2.2.3.

Each partition of the clustered data set must contain at least one tuple. Hence, an empty cluster would be an illegal partition.

Ester and Sander additionally demand that a tuple must be contained in exactly one cluster [16, p. 51]. Yet, according to this demand the EM (Expectation Maximisation) clustering algorithm [12] would be no partitioning clustering algorithm because tuples in an EM cluster can belong to several clusters, although Ester and Sander categorise EM as a partitioning clustering algorithm [16, p. 59f]. According to Han's definition of model-based clustering algorithms [36, p. 348], EM would be a model-based clustering algorithm. However, several authors consider k-means and EM as closely related algorithms such as [62], [67], and [16, p. 59f]. The description of both, k-means and EM, can be found in the following subsections.

Due to the similarity of EM with partitioning clustering algorithms, we categorise EM as a partitioning clustering algorithm and weaken the condition of Ester and Sander by omitting the necessity of one tuple having exactly one associated cluster.

According to the argumentation above, a partitioning clustering algorithm assigns each tuple to at least one cluster. This requirement excludes tuples remaining unassigned. Some non-partitioning clustering algorithms such as density-based clustering algorithms consider only a subset of the data set as relevant. The un-considered data is called noise [15]. Omitting clustering tuples is tolerable if the purpose of a given cluster analysis is only to find concentrations in data. Yet, if the purpose of a cluster analysis is to partition a data set for finding better classifiers it is not.

Partitioning clustering algorithms present the result of the clustering as a set of features of the clusters they found. Yet, the type of feature depends on the algorithm that was used. Therefore, the following subsection surveys the different types of partitioning clustering algorithms which are available at the moment.

Available Types of Partitioning Clustering Algorithms

The type of feature a partitioning clustering algorithm is returning is the most-discriminating factor of partitioning clustering algorithms.

For instance *k-means* returns the centroids of all clusters—the centroid of a cluster is a vector that is the arithmetic mean of the vectors that represent the tuples of that cluster. As k-means assigns each tuple to its nearest mean,

the set of means is sufficient to determine the affiliation of a tuple to a specific cluster.

In analogous way, k-median algorithms such as PAM and CLARANS return the clusters' medoids that can be used to determine the cluster affiliation of a tuple. A medoid of a cluster is a tuple of that cluster that minimises the total distances of all non-medoid tuples to the medoid tuple. In contrast to a centroid, a medoid is an existing tuple.

k-modes is a partitioning clustering algorithm using the modus of a cluster as feature of the cluster. The modus of a cluster is the tuple that has the most-frequent combination of attribute values of all tuples in a cluster.

The *Expectation Maximisation* (EM) [12] algorithm returns a statistical model consisting of k statistical variables with mean and standard deviation. Typically, these variables are assumed to be normally distributed but other distributions could be used alternatively. EM is very similar to k-means but EM also takes the deviation of a cluster into account. Where k-means assigns a tuple to the cluster with the nearest mean, EM assigns a tuple to the cluster having the greatest prior probability. If one would assume that the deviation of all clusters would be identical, k-means and EM would return the same means—given, that both algorithms use the same initialisation. If deviation is identical for all clusters, the condition "greatest probability" and "nearest cluster" are equal.

The scale of attributes typically determines the type of partitioning clustering algorithm to use. As an arithmetic mean of a cluster is only available for continuously scaled attributes, k-means is limited to data sets with continuously scaled attributes. Due to EM's similarity with k-means the same argumentation is true for EM. Again, determining the distances to a medoid requires attributes being scaled at least ordinal. Thus, k-medoid algorithms are limited to ordinally or continuously scaled attributes. k-modes is applicable to data sets having any type of scale but the expressiveness of the modus of a cluster is too low for many applications.

General Functioning of Partitioning Clustering Algorithms

Although partitioning clustering algorithms differ in the type of result they return, they share a common pattern how they compute results. This section presents the common functioning of partitioning clustering algorithms.

Partitioning clustering algorithms improve an initial solution in several iterations. Algorithm 1 shows the pseudo code of a generalised prototype of a partitioning clustering algorithm. A set of features represents the current solution of a partitioning clustering algorithm. Depending on the type of algorithm as presented in the previous subsection, these features are either centroids, medoids, modes, or statistical variables of clusters. In each iteration, the algorithm assigns tuple to clusters and re-computes the features of the current solution.

The way a partitioning algorithm finds an initial solution (line 2 of Algorithm 1) is not fixed. Hence, there exist several heuristics to find good initial solutions. Applying the algorithm on a small sample is a very common technique to find

an initial solution [69]. Here, the resulting solution of the run with the sample becomes the initial solution of the second run with the original data set.

Some algorithms re-order the steps shown in Algorithm 1 to enhance the algorithm's performance. For instance, the MacQueen variant of k-means performs the update of features (line 20) right after re-assigning a tuple to a different cluster (line 16).

Algorithm 1 Generalised pseudo code of a partitioning clustering algorithm

Require: set of tuples represented by a set of points $X = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\}$, number of clusters k
Ensure: set of k features $\Theta = \{\theta_1, \ldots, \theta_k\}$
1: $\Theta \leftarrow$ `determine_initial_solution`(X, k)
2: $X_P \leftarrow \{X_1, \ldots, X_d\}$ /* create d initially empty partitions */
3: **for all** $\vec{x} \in X$ **do**
4: $\quad i \leftarrow$ `determine_index_most_similar_feature`$(\vec{x}, \Theta)$
5: $\quad X_i \leftarrow X_i \cup \{\vec{x}\}$
6: **end for**
7: **for all** $\theta_j \in \Theta$ **do**
8: $\quad \theta_j \leftarrow$ `update_solution`(X_j)
9: **end for**
10: **repeat**
11: $\quad$ **for all** $X_j \in X_P$ **do**
12: $\quad\quad$ **for all** $\vec{x} \in X_j$ **do**
13: $\quad\quad\quad i \leftarrow$ `determine_index_most_similar_feature`$(\vec{x}, \Theta)$
14: $\quad\quad\quad$ **if** $i \neq j$ **then**
15: $\quad\quad\quad\quad X_i \leftarrow X_i \cup \{\vec{x}\}$
16: $\quad\quad\quad\quad X_j \leftarrow X_j \backslash \{\vec{x}\}$
17: $\quad\quad\quad$ **end if**
18: $\quad\quad$ **end for**
19: $\quad\quad$ **for all** $\theta_j \in \Theta$ **do**
20: $\quad\quad\quad \theta_j \leftarrow$ `update_solution`(X_j)
21: $\quad\quad$ **end for**
22: $\quad$ **end for**
23: **until** stop criterion is met
24: **return** Θ

Partitioning clustering algorithms iterate until a stop criterion is met. Typical stop criteria of partitioning clustering algorithms are

- a given number of iterations is complete—CLARANS has this type of stop criterion; additionally, several implementations of algorithms in commercial systems use a maximum number of iterations to guarantee termination in a reasonable amount of time
- the quality of the current iteration and the former iteration does not improve—k-means, k-modes, and EM have this type of stop criterion

- the estimated cost of a further iteration exceeds the estimated benefit of a further iteration.

k-means and EM are part of many commercial data mining tools such as SAS Enterprise Miner, SPSS Clementine, or Teradata Warehouse Miner. Additionally, both algorithms are broadly discussed in literature. As both algorithms require an Euclidian vector space to operate within, many improvements exploiting conditions given in an Euclidean vector space are applicable to both algorithms. Thus, it is sufficient to discuss only one of them in detail.

Due to the usage of k-means in commercial products and the large set of articles discussing improvements of k-means, we will focus on k-means to show how to implement the concepts presented in this dissertation for k-means application.

The significant contribution of this dissertation is to pre-process more tasks than in the conventional way for potential future applications such as k-means clustering applications. k-means fulfills the function of a running example demonstrating the concepts presented in this dissertation. It is left to the reader to traduce the herein shown way of implementing these concepts to other clustering algorithms.

The k-means algorithm

The k-means clustering algorithm is a partitioning clustering algorithm that partitions a set of data into k subsets and returns the centroid of each subset. Each tuple is represented by a vector in a Euclidean vector space. Consequentially, only attributes with continuous, numerical scale are available for k-means clustering because other attributes are improper to span an Euclidean vector space. The centroid of a cluster is the arithmetic mean of all vectors of a cluster.

Parameter k, which denotes the number of clusters to be found, is the only parameter of k-means. It is also the parameter that gives k-means its name.

The general proceeding of k-means is like the proceeding of partitioning clustering algorithms as presented in the previous subsection. The following paragraphs describe how k-means implements the generic pattern of a partitioning clustering algorithm.

Like other partitioning clustering algorithms, k-means starts with an initial solution which it improves in several iterations. Hereby, the initial solution consists of a set of k centroids.

The distance of a tuple to a cluster's centroid determines the affiliation to a specific cluster: k-means assigns each tuple to that cluster that minimises the distance between tuple and centroid.

k-means terminates when there were no re-assignments of tuples from one cluster to another cluster within an iteration. Hence, the minimum number of iterations is two: One iteration to assign tuples to clusters and a second iteration to detect that the initial assignment was optimal.

There are two major variants of k-means, namely Forgy's variant of k-means [17] and MacQueen's variant of k-means [46]. Both are identical except for the time of updating a cluster's centroid.

Forgy's variant of k-means updates centroids at the end of each iteration. In other words, the location of a centroid is constant during an iteration. Not so MacQueen's variant of k-means.

MacQueen's variant of k-means updates a centroid each time the cluster of that centroid either gains or looses a tuple, i.e. the location of a centroid can change during an iteration. Consequentially, updating centroids at the end of an iteration is unnecessary. The continuous update of centroids has a single exception.

During the first iteration there is no update of a cluster's initial centroid when that cluster gains tuples. This exception is necessary to avoid that the tuple that is read first biases the result of the clustering. If the algorithm would perform the first iteration in the same way as it does in succeeding iterations, after reading the first tuple it would update the centroid of that tuple in a way that the centroid and the location of the first tuple coincide in the same point—making any initialisation obsolete.

MacQueen's variant of k-means is said to require only a few iterations, which are typically five up to ten iterations, confer [16, p. 54]. However, some of our tests contradict this observation when the number of tuples exceeds a million and there is no significant concentration of tuples, i.e. the concentration of tuples is only somewhat higher in the vicinity of a centroid than in the rest of the data set. In such a case we observed few tuples that permanently changed their affiliation to clusters from iteration to iteration. As their contribution to the resulting clusters was minor we terminated k-means after a fixed number of iterations when k-means did not terminate otherwise.

The number of iterations needed by MacQueen's variant of k-means is less or equal the number of iterations needed by Forgy's variant of k-means because the centroids move faster to their position. However, updating a centroid at the end of an iteration can also be beneficial under some circumstances. Schaubschläger has demonstrated in his master's thesis that Forgy's variant of k-means is easier to compute in a distributed environment because it needs less messages between participating computers for communicating intermediate results [63].

2.2.4 Hierarchical Clustering Algorithms

All clustering algorithms that produce clusters that themselves consist of clusters are hierarchical clustering algorithms. Hence, their result is a tree of clusters, or as it is also called, a dendrogram.

Unlike partitioning clustering algorithms, hierarchical clustering algorithm do not optimise their clusters according to a specific measure. Hence, comparing the quality of two dendrograms is difficult.

Although many hierarchical clustering algorithm have a parameter indicating the maximal number of clusters to keep the resulting dendrogram small, the actual number of clusters a hierarchical clustering algorithm finds depends only on the data.

As hierarchical clustering algorithm operate independently of user-chosen parameters fixing number of clusters and/or criterion indicating optimality, the

approach of this dissertation uses a hierarchical clustering algorithm to pre-cluster data sets. As many applications need some kind of criterion of optimality, the approach of this dissertation also uses partitioning clustering on top of the results of the hierarchical clustering. Therefore, this section presents the basics of hierarchical clustering algorithms.

There exist two ways to construct a tree of clusters: top-down or bottom-up. An algorithm following the top-down strategy initially assigns all tuples to a single cluster which it recursively tries to split in sub-clusters. Contrary, an algorithm using the bottom-up strategy handles each tuple as its own cluster and tries to add the nearest clusters to a higher-level cluster over and over again until there is only a single cluster left.

Algorithms following the top-down strategy are called divisive clustering algorithm as they divide a single cluster in smaller sub-clusters. Analogously, algorithms of the bottom-up strategy are called agglomerative clustering algorithm because they agglomerate sub-clusters to higher-level clusters.

Both, divisive and agglomerative clustering algorithms, need to determine the distance between a pair of clusters. A divisive clustering algorithm needs this distance to determine the best candidates to split a cluster into a set of sub-clusters, while an agglomerative clustering algorithm needs this distance to detect the nearest clusters to determine candidates to group them into clusters.

For expressing the distance between two clusters there exist several distances, namely

minimum distance The minimum distance between two clusters is the minimum of of the distances of all pairs of tuples where one tuple is part of one cluster and the other tuple part of the other cluster. *Single-link* [16, p. 77] is an alternative name of the minimum distance that indicates that there exists a single link of two points between two clusters having the minimum distance.

average distance The average distance between two clusters is the arithmetic mean of the distances of all pairs of tuples where one tuple is part of one cluster and the other tuple part of the other cluster. *Average-link* [16, p. 77] is an alternative name of average distance.

maximum distance The maximum distance between two clusters is the maximum of the distances of all pairs of tuples where one tuple is part of one cluster and the other tuple part of the other cluster. *Complete-link* [16, p. 77] is an alternative name of the minimum distance that indicates that all pairs of tuples are linked together by a link having the maximum distance.

For comparing the above-mentioned distances, assume there are three clusters the distances of their centroids are identical but which differ in deviation. Hence, average distance is identical for all distances between clusters. Yet, minimum distance favours clusters with high deviation because there more likely exist tuples far away of the clusters' centroid which are closer together. Hence, a clustering algorithm using minimum distance would merge clusters with high

deviation more often than clusters with low deviation. For the same reason, the opposite is true for maximum distance. As deviation is higher there more likely exist tuples of different clusters which are far away from each other. Thus, maximum distance favours small clusters.

Divisive and agglomerative clustering algorithms differ in the way they compute results. Hence, the following sections describe common characteristics and popular representatives of divisive clustering algorithms and agglomerative clustering algorithms. Section 2.2.4 presents the agglomerative clustering BIRCH in more details than the other algorithms because the major contribution of this dissertation uses an extension of BIRCH for data compression due to its superior performance.

Divisive Clustering Algorithms

Divisive clustering algorithms generate a dendrogram top down. Thus, they start with a single cluster that has all tuples assigned and try to split it into several sub-clusters. The algorithm adds these sub-clusters as child nodes of the original cluster such that a dendrogram emerges. Thereby, the original cluster forms the root of the cluster tree. Then it recursively splits the so-created sub-clusters until each sub-cluster of the current leaf node level of the cluster tree fulfills a stop criterion. The way a cluster is split and the stop criterion depend on the specific divisive clustering algorithm that an analyst uses.

DIANA is a divisive clustering algorithm which is presented in [41]. DIANA is short for *DI*visive *ANA*lysis. DIANA generates a binary tree of clusters by recursively splitting a set of tuples in two disjoint subsets. For each cluster that is about to be split DIANA starts with a subset containing all the cluster's tuples and an initially empty subset. Then it moves tuple by tuple from to the initially full subset to the initially empty subset—beginning with the tuple that is most dissimilar to the others. This process lasts as long as the dissimilarity within both subsets decreases. When moving tuples is finished, DIANA recursively splits the so-generated subsets.

The recursive splitting of clusters ends if each cluster at leaf level has only a single tuple associated with.

As DIANA needs to scan the data each time it generates a new level in the cluster tree, it is an algorithm that scales ill in the number of tuples [36, p. 356]. With respect to their ill performance we consider divisive clustering algorithms no further.

Agglomerative Clustering Algorithms

Agglomerative clustering algorithm start with each tuple representing its own cluster. They group similar clusters together to receive a super-ordinate cluster of each group of similar clusters. This process recursively continues until there is only a single cluster left. Hence, the construction of a dendrogram using an agglomerative clustering algorithm happens bottom up.

For determining similar clusters, an agglomerative clustering algorithm uses one of the distance measures for clusters as described in the introduction of section 2.2.4. Minimum distance is a commonly used distance between clusters [5].

There exist several strategies to group clusters to a super-ordinate cluster, namely merging clusters if their distance is lower than a given threshold and merging the k nearest clusters. By inserting tuples in a B^+-like tree where each node has the capacity k is a simple solution to group k clusters together—yet, there might be nodes having less than k entries. Some algorithms like BIRCH use a mixture of both strategies, confer the next subsection.

The merging strategy using a threshold merges clusters as follows: There exists a threshold for all clusters of the same level of the dendrogram. The algorithm determines to each cluster the most similar cluster of the same level. If the distance of this pair of clusters is below the threshold, the algorithm merges both clusters by adding them to the same super-ordinate cluster.

The merging strategy using the nearest neighbours of a cluster detects the nearest neighbours of a cluster and adds them to the same super-ordinate cluster. The clustering algorithm CURE [28], for instance, initially creates several partitions of the data set to keep the resulting dendrogram small. Yet, after initial partitioning CURE iteratively merges pairs of nearest clusters.

BIRCH is a very efficient agglomerative clustering algorithm because it needs a single scan of the data to build a dendrogram. It is also the algorithm on which the algorithm presented in this dissertation founds on. For both reasons, the following section exclusively introduces the agglomerative clustering algorithm BIRCH.

BIRCH

The clustering algorithm CHAD [81][80] uses a modified version of the first phase of the hierarchical clustering algorithm BIRCH as its first phase. Hence, this section presents the unmodified version of BIRCH before presenting the modifications in Section 5.2.

Zhang created BIRCH as vital part of his dissertation [80]. BIRCH is short for Balanced Iterative Reducing and Clustering using Hierarchies. It consists of four phases as described below.

Building a clustering feature tree (CF-tree) With a single scan of the data, BIRCH builds a dendrogram. The inner nodes of the dendrogram are organised in the same way as the inner nodes of a B^+-tree. They build up the tree in the same way except that BIRCH has no key values but uses so-called clustering features (CF) as entries of a node. We will discuss clustering features separately in the subsequent paragraphs.

As mentioned in the previous section, BIRCH uses a threshold to merge clusters. However, there is only a single threshold which it applies on entries of leaf nodes. For inner nodes, BIRCH uses the insertion operation

of B^+-tree which always groups at least k and at maximum $2k-1$ sub-nodes to a super-ordinate node. The capacity k of inner nodes is chosen so that a node fits a page of the machine which performs the clustering.

The dendrogram of BIRCH is memory-optimised which means that BIRCH automatically chooses the threshold that the resulting dendrogram fits in the main memory that is available on the machine which is used for clustering.

Pruning the CF-tree As the resulting clustering feature tree can be very large—too large for analysing it—, BIRCH can prune the clustering feature tree optionally to receive a smaller tree.

Global clustering In phase *global clustering* BIRCH applies another clustering algorithm on the clustering features of the leaf nodes of the CF-tree. In doing so, the used algorithm interprets a clustering feature as an n-ary point in a multi-dimensional vector space. Therefore, the clustering algorithm used in this phase must operate in a vector space.

Refining of clusters Due to the aggregation of tuples in the CF-tree, the result of the global clustering might be imprecise. To circumvent this problem, one can optionally re-scan critical sections of the data, i.e. sections in which tuples are located that the global clustering failed to assign them clearly to a single cluster.

The dissertation of Zhang [80] focusses primarily on the first phase of CHAD. Additionally, the source code of Zhang to test BIRCH includes the functions for the global clustering methods such as k-means only as dummy functions.

Therefore, we refer only to phase 1 of BIRCH when we consider BIRCH although the concept of BIRCH includes also pruning, global clustering, and re-fining.

Several approaches use the CF-tree of BIRCH's phase 1 for another type of clustering such as a k-means clustering. In other words, they implement phase 3, the global clustering. These approaches show that the quality of results is only a little worse than using all tuples but their results are better than other techniques using sampling, e.g. [6]—given, both techniques use the same amount of main memory. They can be found in the related work section of this dissertation.

This dissertation introduces in chapter 5 a similar algorithm to BIRCH but uses a different second phase. While BIRCH focusses on compressing the data, the algorithm presented in chapter 5, CHAD, focusses on making the dendrogram of the first phase applicable to different analyses. Additionally, chapter 5 goes more into the problems concerned with global clustering—such as initialising clustering without scanning the data—as BIRCH does.

As the first phase of BIRCH is very similar to the first phase of CHAD, the remainder of this section describes BIRCH's first phase in detail. To be more specific, it surveys structure and elements of a CF-tree and its construction.

When surveying the construction of a CF-tree, it focusses on BIRCH's automatic adaption of parameters such that the resulting tree always fits into the main memory of the machine that is running BIRCH.

Definition 2.1 *A clustering feature* $cf = (N, \vec{LS}, SS)$ *of a set of tuples is a triple containing number* N*, linear sum* $\vec{LS}$*, and sum of squares* SS *of all tuples represented by this clustering feature.*

Cluster feature is an alternative name for *clustering feature*—e.g. Bradley et al. [59] use the term cluster feature. Both terms denote the same triple as defined above. We will use the term clustering feature to denote an element of BIRCH's CF-tree. In contrast to that, we will use the term general cluster feature to denote an element of CHAD's CFG-tree. General cluster feature and CFG-tree are both terms that we will introduce when presenting CHAD in chapter 5.

We will extend Definition 2.1 in chapter 5 to stress the difference between BIRCH and CHAD. Yet, the definition above is sufficient to present BIRCH and approaches using BIRCH.

The elements of a clustering feature are sufficient to compute many other statistics of the data set which the clustering feature represents. For instance, one can determine the location of the centroid of the data set by dividing the linear sum of a clustering feature by the number of tuples in that clustering feature. The sum of squares is needed to determine some common quality measures of clustering such as the total distance to the cluster's centroid.

The additivity of clustering features is a characteristic of clustering features that is important for BIRCH. Additivity of clustering features means that one can determine the clustering feature of a data set that is the union of two disjoint data sets by adding the according elements of the clustering features of the disjoint data sets.

2.2.5 Measuring Clustering Quality

Quality is an interesting feature of clustering when the purpose of an analysis is any other purpose than data reduction, i.e. the purpose of a cluster analysis is either *clustering for hypothesis generation* or *clustering to improve quality of other techniques*.

This section gives a brief overview of quality measures of clustering in general and the measures used in the experiments in special. Vazirgiannis et al. give a comprehensive overview of quality measures of clustering [72, pp. 93-121].

As clustering groups similar objects together, it is obvious to express quality of clustering in terms of similarity, i.e. a measure indicating the quality of a clustering must indicate higher quality the more similar the tuples of a cluster are. As distances express similarity and dissimilarity, respectively, quality measures of clustering are functions of distances.

As the experiments of this dissertation testing clustering include only experiments with partitioning clustering algorithms, we focus our discussion of quality measures on quality measures for partitioning clustering.

Quality measures of partitioning clustering use the distances of tuples in a cluster to a central point of the cluster to compute a measure for all clusters. Depending on the type of algorithm, the central point is centroid or medoid of a cluster. Hence, this point is a tuple in the data set if it is a medoid. It is a virtual point in a vector space if it is a centroid. Yet, it any case it is a point in a vector space. Thus, we denote this point as $\vec{M}$.

Partitioning clustering algorithms searching for medoids try to minimise the sum of differences between tuples and the medoid of their cluster. Hence, this sum is a quality measure for medoid-based clustering algorithms, which we further reference as *total distance*.

Definition 2.2 *The* total distance TD *of a clustering* C *consisting of* k *clusters* $C = \{C_1, \ldots, C_j, \ldots, C_k\}$ *and a distance function dist is the sum of all sums of the distance of each tuple* $\vec{x}$ *and a central point* $\vec{M}_j$ *of the cluster* C_j *in which the tuple is part of, i.e.*

$$TD = \sum_{\forall C_j \in C} \sum_{\forall \vec{x}_i \in C_j} dist(\vec{x}_i, \vec{M}_j).$$

Partitioning clustering algorithms searching for centroids try to minimise the sum of squared distances. Analogously, we refer this quality measure as *total distance of squares* TD^2.

Definition 2.3 *The* total distance of squares TD^2 *of a clustering* C *consisting of* k *clusters* $C = \{C_1, \ldots, C_j, \ldots, C_k\}$ *and a distance function dist is the sum of all sums of the distance of each tuple* $\vec{x}$ *and a central point* $\vec{M}_j$ *of the cluster* C_j *in which the tuple is part of, i.e.*

$$TD^2 = \sum_{\forall C_j \in C} \sum_{\forall \vec{x}_i \in C_j} \left(dist(\vec{x}_i, \vec{M}_j) \right)^2.$$

The smaller the total distance or the total distance of squares, respectively, of a clustering are the higher is the quality of the clustering. Yet, total distance and total distance of squares depend significantly on the number of clusters in a clustering. If the number of clusters increases, total distance and total distance of squares decrease accordingly. If the number of tuples equals the number of clusters then optimal total distance and total distance of squares. Hence, one can use total distance and total distance of squares for comparing cluster analyses only if the number of clusters is fixed. If the number of clusters is variable, i.e. the analyst or the data mining system tries a partitioning clustering algorithm several times to find the optimal number of clusters, then quality measures that are independent of the number of clusters are needed.

The experiments in the experiments chapter of this dissertation testing clustering use total distance of squares as quality measure. This is a valid option because the number of clusters is fixed for a given series of tests. For a comprehensive overview of other quality measures of clustering that are capable to determine the quality of clusterings with differing number of clusters, the interested reader is referred to Vazirgiannis et al. [72].

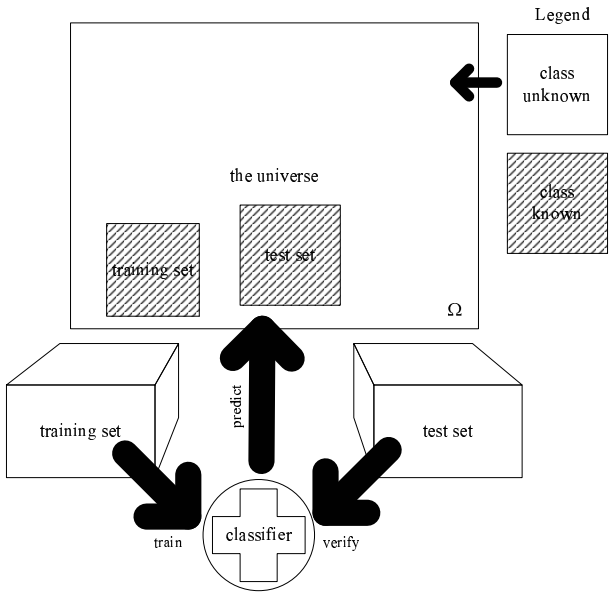


Figure 2.3: Using a trained and verified classifier to predict tuples' class where class is unknown

2.3 Classification

This section gives a brief overview of the data mining method of classification. It first introduces the major concepts and terms of classification. Second, it surveys how to measure the quality of the result of a classification. Finally, it presents the major types of classification algorithms in short.

Knowing quality measures and classification algorithms is necessary to understand how the concepts of this dissertation improve classification.

Classification is one of the main data mining methods. The aim of classification is to train the so-called classifier which is a function that is capable to determine the affiliation of an object to one class out of a pre-defined set of classes. The classifier uses the attribute values of the object that has to be classified to determine the according class. In the context of classification attributes are also called features. In the following the term "feature" will be used because it is the more common term.

Classification consists of two phases: training a classifier with a set of data where class affiliation is known for all of its tuples and using the classifier to determine the class affiliation of tuples where class affiliation is unknown.

The set of data that is used for training the classifier is called training set. It consists of one subset for each class.

In the common case that a classifier is required to determine whether a tuple is part of a specific class or not, the subsets of the training sets are called positive set and negative set. The positive set contains only those tuples that

are affiliated with the only interesting class while the negative class contains only tuples that are not affiliated with that class.

The sizes of positive and negative set influence the quality of the resulting classifier. Usually, the larger a training set is the better is the classifier. Sometimes it is more tolerable having false positives than having false negatives. If that is the case an analyst would use a positive set which is larger in size compared to the negative set. Tests for early diagnosis of different types of cancer are good examples where it is more tolerated to send a healthy person to further examination than overlooking an ill person. Subsection 2.3.1 discusses quality criteria that are able to distinguish between tolerable and intolerable errors.

Several classification algorithms are capable to train only classifiers that have a single class of interest. Yet, if there are more than two classes it is still possible to use these algorithms by training iteratively a set of classifiers—one classifier for each class. In such a case the positive set of each class is the subset of those tuples affiliated with that class while the negative set is a subset of the union set of all other subsets.

For determining the quality of a classifier, an additional set of data where class affiliation is known is required—this set of data is called test set. Subsection 2.3.1 describes available methods for determining classification quality in detail.

The classification problem can be formally written as follows:

Assume there is a set consisting of objects that have to be classified according the classes $\{c_1, \ldots, c_k\} = C$. Each object is represented by a d-dimensional feature vector $x_i \in X, X = X_1 \times \cdots \times X_d$. Let X_j denote a single feature and x_{ij} denote the value of this feature of the object that is represented by x_i.

Definition 2.4 *A classifier f is a function $f : X \to C$ that assigns a class to each object that is represented by a feature vector $x_i \in X$.*

Definition 2.5 *A training set $T \subseteq X \times C$ is a set of pairs $(x_i, y_i) \in T$ that consists of a feature vector x_i and a class assignment $y_i \in C$. The value of y_i denotes the real class of the object represented by x_i and is known before classification.*

If the set of classes C consists only of the classes c_1 and c_2 it is common to call one of these classes positive and the other one negative. Let c_1 denote the positive class in the following two paragraphs. Accordingly, c_2 denotes the negative class.

Definition 2.6 *A positive set $P \subset T$ is a subset of the training set T that consists of all feature vectors that are affiliated with the positive class c_1, i.e. $P = \{(x_i, y_i) \in T | y_i = c_1\}$.*

Definition 2.7 *A negative set $N \subset T$ is a subset of the training set T that consists of all feature vectors that are affiliated with the negative class c_2, i.e. $N = \{(x_i, y_i) \in T | y_i = c_2\}$.*

cid	average sales	sales current quarter	customer type	churn flag
1	130	0	0	1
2	500	250	1	1
3	350	360	1	0
4	1'000	700	1	0
5	170	40	0	0
⋮	⋮	⋮	⋮	⋮

Table 2.1: Clip of the pre-processed data for a churn analysis

Definition 2.8 *A test set $\hat{T} \subseteq X \times C$ is a set of pairs $(x_i, y_i) \in \hat{T}$ that consist of a feature vector and a class assignment $y_i \in C$. The value of y_i is the real class of the object represented by x_i and is known before classification.*

The remainder of this section is organised as follows: Subsection 2.3.1 introduces quality measures of classification and describes how to use them to test the quality of a classifier. Each of the following subsections focusses on a specific category of classification algorithms which are Bayes classification algorithms, decision tree algorithms, and support vector machines—in this order.

For illustrative purposes each subsection is exemplified with a sub-problem of the running example. To be more specific, the succeeding subsections use the analysis with the aim to find the reasons *"why customers cancel their standing orders of journals"* to exemplify the concepts presented within them. Such an analysis is called a *churn analysis* in literature [78] as customers that cancel their contracts are often called *churners*.

The sales data set is a potential data source for churn analyses but it needs some pre-processing. As the trained classifier of a churn analysis shall be used several times—e.g. once per quarter—the classifier must not be restricted to a static time frame. This means that the resulting classifier may use attributes such as the *current year's sales* and *last year's sales* but must not use attributes such as *sales in 2005*. Hence, one must derive new features relative to the current date. Additionally, all relevant features must be determined per customer. Hence, aggregation of sales grouped by each customer is needed.

Assume that after pre-processing the resulting training set looks like as depicted in Table 2.1. Attribute *cid* stores a unique customer number. Attribute *average sales* represents the average sale per quarter in monetary units of the current customer over all quarters since beginning storing data about that customer. Contrary, attribute *sales current quarter* is the sum of all sales of this customer in the current quarter. Attribute *customer type* has either value 0 or value 1, where value 0 indicates that the current customer is an individual, while value 1 indicates that the customer is a corporate customer. Finally, if the churn flag assumes value 1 then the current customer is a churner.

		estimate	
		positive	negative
real class	positive	true positives	false negatives
	negative	false positives	true negatives

Table 2.2: confusion matrix of binary classes

		estimate				
		c_1	c_2	c_3	c_4	c_5
real class	c_1	$\mathbf{f_{11}}$	f_{12}	f_{13}	f_{14}	f_{15}
	c_2	f_{21}	$\mathbf{f_{22}}$	f_{23}	f_{24}	f_{25}
	c_3	f_{31}	f_{32}	$\mathbf{f_{33}}$	f_{34}	f_{35}
	c_4	f_{41}	f_{42}	f_{43}	$\mathbf{f_{44}}$	f_{45}
	c_5	f_{51}	f_{52}	f_{53}	f_{54}	$\mathbf{f_{55}}$

Table 2.3: confusion matrix with five classes

2.3.1 Measuring Classification Quality

Quality is the most interesting feature of classification because it influences the potential benefit of the resulting classifier. Hence, this subsection surveys how to test the quality of a classifier and presents quality measures of classification.

Measuring the quality of a classifier needs a set of data in which the real class affiliation of all of its tuples is known—which is called the test set. When comparing the class of a tuple that the classifier estimates for that tuple with the tuple's real class, one is able to detect misclassification.

The test set and the training set should be disjoint if possible. Otherwise, a classification algorithm can train a classifier that exactly predicts all tuples of the training set but is unable to correctly predict other tuples. The phenomenon that a classifier predicts the training set very well but is unable to do so for the rest of the universe is called *over-fitting*. If major parts of training set and test set overlap, then it is impossible to detect over-fitting.

Analogously, the phenomenon that a classifier correctly predicts only a minority of tuples is called *under-fitting*. A good classifier is free of both phenomena.

As mentioned in the previous subsection, there exist applications of classification where one is interested in the correct classification of one or only a few classes. In such a case, misclassification of other classes is uninteresting. Early diagnostics of hazardous diseases is an example where users are more interested in finding all ill persons and take the risk of erroneously diagnosing healthy individuals as ill. More intensive diagnostics can correct a wrong positive diagnosis but omitting to treat a wrongfully negatively classified individual in time might complicate medical treatment when diagnosed later—or medical treatment even might be too late.

As the interest in correct classification of specific classes may vary from analysis to analysis, there exist different quality measures of classification such

as accuracy, specificity and sensitivity, as defined below.

The so-called confusion matrix contains all information that is needed to compute quality measures of classification. It is a matrix in which the classes are positioned in horizontal and vertical dimension. One dimension denotes the class the classifier has estimated, while the other dimension denotes the real class of the classified tuples. Each element m_{ij} of the matrix contains the number of tuples that are classified as class c_j and have real class c_i—where the indices i and j may but need not be identical. Tables 2.2 and 2.3 depict two confusion matrices. The matrix in Table 2.2 is a matrix of a classifier with only a single class of interest. Contrary, table 2.3 shows a matrix of a classifier with some classes.

Obviously, the diagonal of the confusion matrix contains the numbers of those tuples that are classified as the class they really are. Hence, the sum of the numbers in the diagonal is the number of correctly classified tuples. Consequentially, all other elements of the confusion matrix denote misclassified tuples. The next paragraphs show how to compute quality measures using a confusion matrix.

Definition 2.9 *The* accuracy *of a classifier determines the estimated ratio of correctly classified tuples with regard to all tuples classified by a given classifier. It is the quotient*

$$accuracy = \frac{number\ of\ correctly\ classified}{number\ of\ tuples}.$$

According to its definition, accuracy is the quotient of the sum of the diagonal of the confusion matrix and the number of tuples of the test set. One can determine the number of tuples by adding all row sums or by adding all column sums of the confusion matrix.

There exist analyses in which the accuracy of classification is unsuitable to express the quality of a classifier. If a test set consists predominantly of tuples of a single class and only a few tuples of other classes, a classifier that predicts all tuples as member of the predominant class has a very high accuracy although the analyst might be interested more in the other classes. Analysing patients with a rare but severe disease is such an analysis where the analyst is interested in correctly classifying this rare class. Predicting customers that are likely to quit their contract within the next weeks is another example where the interesting class is rare compared to the uninteresting class of non-quitters.

Definition 2.10 *The* sensitivity *of a classifier denotes the portion of a specific class which the classifier detects as member of this class. It is the quotient*

$$sensitivity = \frac{number\ of\ correctly\ classified\ tuples\ of\ a\ specific\ class}{number\ of\ tuples\ of\ a\ specific\ class}.$$

In other words, the sensitivity of a classifier with respect to a specific class only examines correctly classified tuples of that class. For instance, the sensitivity of the classifier shown in confusion matrix of Table 2.3 with respect to class c_3 is $\frac{f_{33}}{\sum_{j=1}^{5} f_{3j}}$.

training set (a)

		estimate	
		churn	no churn
real class	churn	215	35
	no churn	3	247

test set (a)

		estimate	
		churn	no churn
real class	churn	15	113
	no churn	55	237

Table 2.4: Confusion matrices of the first classifier of the running example

training set (b)

		estimate	
		churn	no churn
real class	churn	198	52
	no churn	15	235

test set (b)

		estimate	
		churn	no churn
real class	churn	75	43
	no churn	25	267

Table 2.5: Confusion matrices of the second classifier of the running example

If there is only a single class the sensitivity is the quotient of true positives and positives in total. If the negative class is the class of interest, the term *specificity* denotes the quotient of true negatives and negatives in total.

Assume that an analyst of the fictive company of the running example has completed the training of three classifiers. Each time he scored the classifier against the training set and the test set. The tables 2.4, 2.5, and 2.6 contain the confusion matrices he received that way.

The accuracy of the classifier shown figure in 2.4 is very high in the training set but very low in the test set, namely $\frac{215+247}{500} = 92.4\%$ compared to $\frac{15+237}{15+237+113+55} = 60\%$. When we only know the frequency of churners and non-churners we would have a similar accuracy by only guessing the class of a tuple randomly. Thus, the classifier performs poorly in the test set. As mentioned above, the phenomenon where a classifier performs significantly better in the training set but only poorly in the test set is called *over-fitting*. The sensitivity of attribute *churn* indicates over-fitting even more significantly with a sensitivity of $\frac{215}{250}$ in the training set versus a sensitivity of $\frac{15}{128}$ in the test set.

Contrary, the second classifier the confusion matrices of which is depicted in Table 2.5 has a somewhat worse accuracy in the training set than the first

training set (c)

		estimate	
		churn	no churn
real class	churn	245	5
	no churn	80	170

test set (c)

		estimate	
		churn	no churn
real class	churn	113	5
	no churn	65	227

Table 2.6: Confusion matrices of the third classifier of the running example

classifier but its accuracy in the test set with approximately 81% is much better than the first classifier's accuracy of 60% in the test set. Hence, the second classifier is superior to the first classifier because it is less specialised to a given training set. Therefore, its estimated accuracy in an additional, yet unknown data set is higher than the first classifier's expected accuracy in the same data set.

However, if we assume that the company is interested in loosing as few customers as possible, then the sensitivity of the second classifier which is approximately 64% should be better—even if the accuracy decreases that way. Yet, if misclassification of non-churners as churners is tolerable then a decreasing accuracy is also tolerable as long as sensitivity increases. Table 2.6 includes a set of confusion matrices of a classifier that is worse in accuracy than the second classifier but outperforms the second classifier in terms of sensitivity. To be specific, the accuracy of the third classifier in the test set is only approximately 68% but its sensitivity of almost 98% is the highest one of all classifiers.

Summarising, due to the highest sensitivity value the third classifier is the best classifier when assuming that detecting churners is more important than misclassification of non-churners as churner. However, if this assumption does not apply then the second classifier is the best one because it is the classifier with the highest accuracy in the test set. The accuracy in the training set is irrelevant for the quality of a classifier. Yet, it is a good for indicating overfitting, as exemplified with the first classifier.

The remaining subsections of this section focus on how to construct such a classifier.

2.3.2 Bayes Classifier

Bayes classification is a category of classification algorithms that assigns that class to an object that maximises expectation. They use the theorem of Bayes to compute the probability that a class c_j is the real class of an object represented by a specific feature vector that has been observed—which is also called the

posterior probability. In other words, a Bayesian classifier determines the class that is the most likely class for an observed feature vector.

According to the theorem of Bayes, the posterior probability of class c_j being the right class of an object X is determined by

$$P(c_j|X) = P(X|c_j)P(c_j) \tag{2.2}$$

The term $P(X|c_j)$ is called the prior probability. The prior probability determines how likely it is that a specific feature vector occurs if c_j is the current class.

The probability of a class c_j ($P(c_j)$) can be estimated by its frequency in the training set.

Yet, Bayesian classifiers differ in the way the prior probability $P(X|c_i)$ is computed. Naïve Bayes classifiers assume independency between each pair of attributes to simplify the computation of the prior probability. Contrary, optimal Bayes classifiers use a Bayesian network to express conditional dependencies among attributes.

A naïve Bayesian classifier assumes that features are independent in pairs. According to this assumption, the prior probability of an object X can be determined by multiplication of the prior probabilities of each feature, i.e.

$$P(X|c_i) = \prod_{j=1}^{d} P(x_j|c_i) \tag{2.3}$$

Hence, a naïve Bayesian classifier is fully defined if there exists a $P(x_j|c_i)$ for each combination of a value of an attribute and a class.

The prior probability of an individual feature can be estimated by the according frequency in the training set.

If a dimension has numerical scale, the number of combinations of classes and potential values of this dimension is unlimited. Thus, the probability that a tuple is of any class is zero because the probability to receive a specific point is zero. In such a case, the value of a density function at point x_i replaces the prior probability, where x_i is the value in dimension with numerical scale. The naïve Bayes classifier then assigns a tuple to that class where the probability density is highest.

In contrast to naïve Bayesian classifiers, Bayesian Networks take the conditional dependencies of features into account. A Bayesian Network is a directed acyclic graph where each vertex denotes a value of a feature and each edge between two vertices denotes the conditional probability that the feature value of the succeeding vertex occurs given the feature value of preceding vertex has occurred.

As continuous attributes would mean having a graph with unlimited vertices and edges, Bayesian Networks are unable to handle continuous attributes.

Again, the running example shall illustrate the construction of a naïve Bayes classifier. For classification we need to split the pre-processed data into a training set and a test set. The tables 2.7 and 2.8 show samples of training set and test set, respectively.

cid	average sales	sales current quarter	customer type	churn flag
2	500	250	1	1
4	1'000	700	1	0
5	170	40	0	0
⋮	⋮	⋮	⋮	⋮

Table 2.7: Clip of the training set for a churn analysis

cid	average sales	sales current quarter	customer type	churn flag
1	130	0	0	1
3	350	360	1	0
⋮	⋮	⋮	⋮	⋮

Table 2.8: Clip of the test set for a churn analysis

To train a naïve Bayes classifier we must determine the conditional probabilities of the classes *churn* and *no churn* given that any attribute value of one of the other attributes has occurred. Additionally, the probabilities of each class without any condition must be known. We can use the according frequencies in the training set to estimate these probabilities. A single scan of the training set is sufficient to determine those frequencies. Once these frequencies are known, the naïve Bayes classifier is fully trained.

When scoring a tuple, i.e. when we assign a class to a tuple where the class is initially unknown, we have to compute the probability of each class by multiplying the according probabilities. If we consider the tuple of the training set with identifier cid assuming 3, we can easily determine the probability of churn given customer type is 1 (corporate customer). However, the attribute value of attribute *average sales* might be unique. Hence, the probability that would be necessary to determine the combined probability of all attributes would be missing as there is no such tuple in the training set.

If a value of an attribute we need for scoring is missing in the training set, we can either determine the class of that tuple without this attribute or we can determine the probability density function of that attribute. In order to determine the probability density function we need to analyse the joint distribution of the attribute and the class attribute.

As form and all parameters of the probability density function are unknown we can only estimate it. Estimating includes choosing a type of distribution and determining the optimal parameters according to the chosen type of distribution. We can use a McNemar test for testing if a specific type of distribution is appropriate to approximate the probability density function's distribution. If the type of distribution is known we can perform a least square estimate to determine its optimal parameters.

Section 6.3 will show a way of constructing a joint probability density function as a by-product of other tasks. The there-presented algorithm uses the

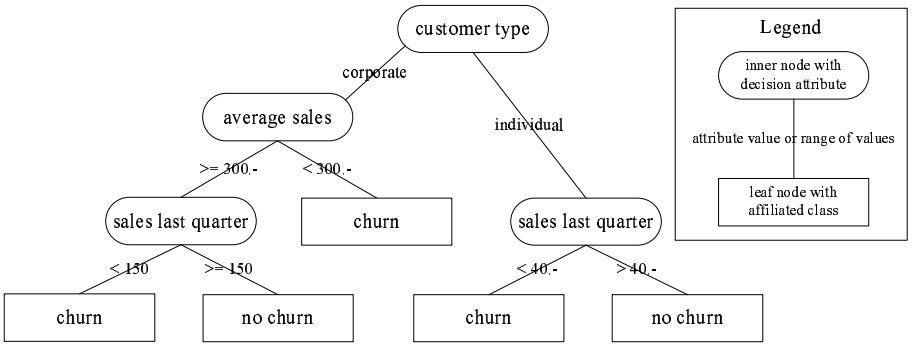


Figure 2.4: Decision tree that predicts whether a customer is about to churn or not

observation that an arbitrary shaped probability density function can be approximated by a set of normal distributions.

2.3.3 Decision Trees

A decision tree is a tree in which each leaf node is associated with one of a set of classes. Each inner node of a decision tree has a decision criterion which is a single attribute that is used for decision. Each edge between two nodes—regardless if inner node or leaf node—represents a decision of the anteceding inner node. Hence, each path within the decision tree is a sequence of decisions.

Classification algorithms generating decision trees take the training set to derive a decision tree. A decision tree algorithm iteratively identifies a split criterion using a single attribute and splits the training set in disjoint subsets according to the split criterion. For doing this, the algorithm tries to split the training set according to each attribute. Finally, it uses that attribute for splitting that partitions the training set best to some kind of metric such as entropy loss.

A split criterion shatters the training set into several subsets. Depending on the type of decision tree algorithm, the split is binary or n-ary. If the attribute that is used for splitting is a continuous or ordinal attribute, algorithms typically use a binary split at a specific attribute value. Tuples with a smaller or equal value in the split attribute are assigned to one branch of the tree, greater values are assigned to the other branch. However, if the split attribute is categorical, it is possible to create a branch for each distinct attribute value of the split attribute. Alternatively, it is also possible to do a binary split by partitioning the distinct attribute values in two disjoint subsets. However, the type classification algorithm determines the type of split that is used. The Rainforest algorithm [21] which is used as sample classification algorithm in this dissertation is a generic algorithm that might use any of the above-mentioned strategies for splitting. As it can use several strategies instead of a single strategy, Rainforest is ideally suited to be used as sample algorithm.

Regardless which split strategy a classification algorithm uses, the algorithm scores potential splits according to a given metric. Entropy loss, information gain, and the Gini index are metrics for choosing the best split.

According to Ester and Sander [16, p.129], the entropy of a set of tuples T of the training set is the minimal number of bits that are necessary to code a message to indicate the class of a random tuple, or according to Shannon's theory of communication

$$entropy(T) = -\sum_{i=1}^{k} p_i \log_2 p_i,$$

where p_i is the probability of a tuple being an element of the i-th class of a set of k classes.

The entropy loss is the difference in entropy before and after a split. Yet after a split the training set is scattered into multiple sets of tuples. Therefore, the entropy after a split consists of the entropies of each subset. The total entropy after split is the sum of the weighted entropies of each subset. Thus, the *loss* in entropy according the split attribute a and the partitioning in subsets $\{T_1^a, \ldots T_i^a, \ldots T_t^a\}$ the split attribute induces is

$$entropy\_loss(T, a) = entropy(T) - \sum_{i=1}^{t} \frac{|T_i^a|}{|T|} entropy(T_i^a).$$

Information gain is an alternative name of entropy loss. Hence, both metrics are computed identically. The partitioning in subsets which has the highest information gain is used for splitting.

The Gini index or Gini coefficient [22] is a metric that indicates the inequality of objects within a set. It is the area between the Laurenz curve and the bisecting line of the first quadrant divided by the total area under the bisecting line. The Laurenz curve typically determines which percentage of persons owns which percentage of resources but it can also be used to measure concentrations of attribute values.

Among a set of objects and a set of k classes, the Gini index is

$$gini(T) = 1 - \sum_{i=1}^{k} p_i^2 \ [16, p.129],$$

where p_i is the probability of a tuple being an element of the i-th class. Again, as the training set is shattered in multiple subsets, the Gini index of the set of subsets is the weighted sum of the Gini indices of all subsets, i.e.

$$gini(T_1, \ldots, T_i \ldots, T_m) = \sum_{i=1}^{m} \frac{|T_i|}{|T|} gini(T_i) \ [16, p.129].$$

The Gini index is zero if all tuples have the same attribute value. Hence, the smaller the Gini index after a potential split is the better is that split. Thus, finding the best split with the Gini index means to minimise the Gini index.

A sample decision tree is shown in figure 2.4. For instance, the root node is an inner node that uses the attribute *customer type* for decision. *Customer type* has two potential attribute values, *corporate* and *individual*, respectively. Thus, when classifying a tuple of corporate customers the scoring algorithm follows the left branch of the sample decision tree. Otherwise, it would choose the right branch.

It is easy to derive classification rules from a decision tree as each path within the tree represents a sequence of decisions. Hence, by joining the criteria of all visited inner nodes of a path in the tree with conjunctions one receives a logical rule that indicates when a tuple is part of that path. As all paths must be disjoint in pairs, one can join the rules of paths that share the same assigned class with disjoints to receive a rule that determines when a tuple is member of that class. For instance, in the decision tree which is depicted in figure 2.4 all tuples of class *no churn* fulfill the rule *customer type* = *corporate* $\wedge$ *average sales* $\geq 300 \wedge$ *sales last quarter* $\geq 150 \vee$ *customer type* = *individual* $\wedge$ *sales last quarter* $\geq 40 \rightarrow$ *no churn*.

Although the sequential alignment of decisions allows to derive rules is an advantage of decision trees, this sequential alignment is also a major drawback of decision trees—especially if there is at least one pair of attributes that is strongly correlated. If attributes are correlated then the algorithm would have to consider correlated attributes simultaneously to find an optimal decision tree. As it is impossible to do so, the resulting tree tends to have several additional layers with alternating order of correlated attributes—yet, always with different split points. If an analyst detects alternating sequences of the same set of attributes he or she can re-run the classification but replaces the set of correlated attributes with a single computed attribute.

Assume that in our example there is a strong correlation between *sales last quarter* and *average sales*. As might be seen in Figure 2.4, there are many nodes needed to express that if the sales of the latest quarter significantly decrease below the average level of sales, then the customer is about to churn. Contrary, if we use an attribute that we compose by dividing both correlated attribute, then we can simplify the decision tree to a single decision which is $\frac{\text{sales last quarter}}{\text{average sales}} \leq$ *constant* $\rightarrow$ *churn*. However, due to considering attributes sequentially, a decision tree algorithm is unable to find such rules.

The running example shall illustrate the construction of a decision tree. Table 2.9 shows the training set to construct a decision tree.

In order to determine the entropy of the training set T, we estimate the probability of classes churn and no churn by the according frequencies of these classes in the training set. That way we receive the entropy of the training set as follows:

$$entropy(T) = -\frac{2}{6}\log_2\frac{2}{6} - \frac{4}{6}\log_2\frac{4}{6} \approx 0.918.$$

In order to determine the attribute which splits the training set best to reduce entropy we test to split the training set according to each attribute. Attribute *cid* is unsuitable to be a decision criterion of a decision tree because its values are unique. Using a unique attribute would make a decision tree prone

cid	average sales	sales current quarter	customer type	churn flag
2	500	250	1	1
4	1'000	700	1	0
5	170	40	0	0
6	440	220	1	1
7	1'100	650	1	0
9	175	24	0	0

Table 2.9: Training set T to construct a decision tree for a churn analysis

to over-fitting because it can only classify known tuples. Hence, we omit testing attribute *cid*.

By splitting the training set according to attribute *customer type* we receive two subsets: one subset storing all tuples in the training set where customer type is individual and another subset storing all tuples of corporate customers. The entropy of the subset of individual customers is

$$-\underbrace{\frac{0}{2}\log_2\frac{0}{2}}_{\lim_{x\to 0} x\log_2 x=0} -\frac{2}{2}\log_2\frac{2}{2} = 0 \tag{2.4}$$

, while the entropy of the subset of corporate customers is

$$-\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4} = 1.$$

Weighted by the size of both subsets, the entropy after a split according to customer type is $\frac{2}{3}$. Thus, the reduction in entropy is approximately $0.918 - 0.667 = 0.215$.

Statement 2.4 needs further discussion. In the training set there are no individual customers which are also churners. Thus, our estimate for the probability of individual customer being churners is zero. Yet, the logarithmus dualis is undefined for this value. Hence, we have to use the limit of the term $\lim_{x\to 0} x\log_2 x$ to find the appropriate entropy.

Trying to split according to one of the remaining attributes is more complicated than splitting according customer type as the number of distinct values of these attributes is much greater. Hence, there are many potential splits for each attribute. Using an automatically generated histogram is a solution to determine potential split points. Then, the borders of each interval of the histogram is a potential split point. When considering attribute *average sales* we might have built a histogram consisting of the intervals $[0; 200)$, $[200; 600)$, and $[600; +\infty)$—with the potential split points 200 and 600. If we determine the entropy loss for these potential split points, we receive the same entropy loss as the entropy loss according to customer type. Other split points either have less or the same entropy loss. Hence, there are multiple potential split points having the same entropy loss. Depending on the classification algorithm it chooses either the one or the other candidate of potential splits. In the example decision

tree depicted in Figure 2.4 the algorithm has chosen customer type as initial split.

The approach of anticipatory data mining can improve the construction of decision trees in two ways: On the one hand, it can assist the selection of the training set by selecting only tuples that are typical representatives of the data set. This results in better classification accuracy because the training set contains less noise, as shown in Section 7.5. On the other hand, it can simplify finding good split points for numerical attributes as shown in Section 7.5.

2.4 Association Rule Mining

Association rules determine the statistical relation of specific events. They are used to measure the co-occurrence of events such as customers buying several items in common or users looking at the same set of web pages during a session.

This section surveys association rule analysis as far as its concepts are needed to understand approaches that pre-compute intermediate results for various association rule analyses. Chapter 3 includes previous work focussing on re-using results of an association rule analysis in another association rule analysis. Section 6.6 also shows how to apply the general principle of this dissertation to pre-compute results of an association rule analysis in anticipation.

This section first introduces the general concept of association rules before it presents algorithms for association rule analysis.

Analogous to previous sections, the running example shall illustrate the concepts of association rule mining. Assume that the publishing company wants to know which products sell well together and which do not. In order to achieve this goal, an analyst of the company performs a so-called *market basket analysis* that finds patterns of events that frequently occur together such as products that are shopped together in a single sales transaction—or more literally, that lie together in a market basket.

For a market basket analysis, the company needs only those data that indicate which products are purchased together with other products. Hence, the product identifier and an identifier of the sales transaction are sufficient attributes for a market basket analysis. In the running example attribute *prid* identifies the sold product. The combination of *timestamp* and identifier of the customer *cid* identify the individual sales transaction of a specific customer.

Assume that the analyst has pre-processed the data of the sales-cube according to the new schema

```
salespruned(cid, timestamp, prid),
```

which is just a sub-cube of the sales-cube. Table 2.10 shows a sample of the fact table of this cube. The right side of this table shows the according transactions in the fact table and places the items of a transaction in horizontal order.

cid	timestamp	prid
4711	2005-01-05 16:15	1
4711	2005-01-05 16:15	2
4711	2005-01-05 16:15	3
4711	2005-04-01 09:23	1
4711	2005-04-01 09:23	3
0815	2005-04-01 09:27	1
0815	2005-04-01 09:27	4
1704	2005-05-05 13:13	1
1704	2005-05-05 13:13	3
1704	2005-05-05 13:13	4
1704	2005-06-05 10:30	3
1704	2005-06-05 10:30	4

transactions
1, 2, 3
1, 3
1, 4
1, 3, 4
3, 4

Table 2.10: Clip of the tuples of the table that an analyst has pre-processed for a market basket analysis

2.4.1 Association Rules

An Association rule is a logical implication that consists of a condition and a consequence. Both, condition and consequence, consist of sets of statistical events. These events are called items in literature, c.f. [36, p. 228][2]. Analogously, an itemset is a set of items. Association rules are often-presented as logical implications of the form *condition* $\Rightarrow$ consequence.

An association rule indicates a frequent co-occurrence of events or items in a set of transactions. A transaction is an atomic process in which several events or items might occur. For instance, when analysing market baskets, each individual sale is a relevant transaction. The sold goods are the items to be analysed.

For indicating strength and relevance of a rule, there exist several measures such as support and confidence.

The support of a rule indicates how often the events mentioned in that rule occur together with respect to a total number of observations. In other words, support measures the frequency that all items in a rule occur together.

Analogously, the support of an itemset $s(itemset)$ measures the frequency that all items of an itemset occur together. Thus, the support of a rule is identical with the support of the itemset of the union set of condition and consequence.

$$s(condition \Rightarrow consequence) = s(condition \cup consequence)$$

Contrary, the confidence of a rule measures how often the events of the consequence occur when the events of the condition occur. Thus, confidence is the conditional frequency that the consequence occurs provided that the condition has occurred.

It is possible to determine the confidence of a rule $c(condition \Rightarrow$

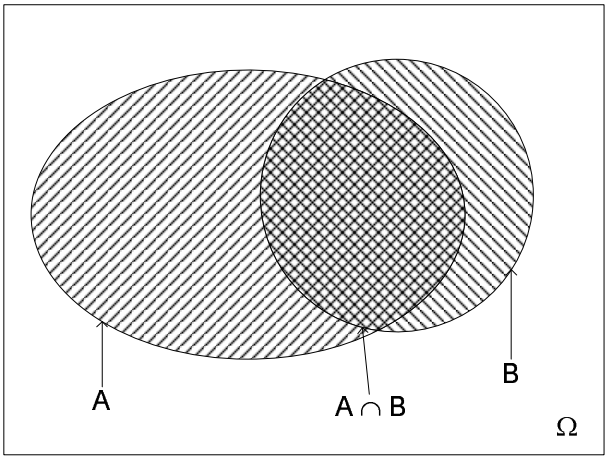


Figure 2.5: Frequencies of an association rule example where only $B \rightarrow A$ satisfies minimum confidence

c*onsequence*) using only the frequencies of the itemsets of that rule, as indicated below.

$$\begin{aligned} c(condition \Rightarrow consequence) &= s(consequence|condition) \\ &= \frac{s(condition \cup consequence)}{s(condition)} \end{aligned}$$

For instance, "customer buys product A" and "customer buys product B" would be relevant events when analysing market baskets of customers. If a company sells product A in 40 different transactions in a set of 100 sales transactions and among these 30 times in combination with product B, then the rule "if customer buys product A, this customer will also buy product B" has a support of 40 % and a confidence of $\frac{30}{40} = 75\%$.

The higher the values of support and confidence of a given association rule are, the more relevant is that rule because the association between condition and consequence is stronger when both values are high. However, support and confidence are concurring items in most times. If the support of a rule is small, the confidence of the rule is typically high. If there is only one transaction that fulfills the rule then the rule's confidence is 1 because in one of one transactions the consequence is true for the given condition. Yet, such a rule lacks of generality. Hence, the rule should be true for at least a reasonable number of times.

For being valid for large sets of data, we consider an association rule only valid if its support and confidence exceed user-given thresholds for support and confidence, further denoted as minimum support and minimum confidence, respectively.

Choosing the right value of minimum support is crucial for an association rule analysis: Choosing a too small value means that there might be association

rules that originate by pure chance and not as a result of statistical correlation. Contrary, choosing a too high value means that potentially interesting association rules remain unconsidered.

Thus, the minimum support should be at least as high as needed to avoid association rules that represent results of pure chance. Additionally, the minimum support should be as small as possible. Ideally, the minimum support should be only somewhat higher as needed for being statistically relevant.

Yet, there can be associations that are statistically relevant but uninteresting. Consider a association rule between two items each of which is very likely. Then, if there is no correlation the probability that both items occur together is also very high. In the extreme case in which items A and B always occur, both items also occur always together. Thus, support and confidence assume their maximum value of 1. However, one cannot tell whether both items are associated or not because the estimated probability is 1, too. Interesting measures exist to solve this problem.

Interesting measures indicate how much stronger the association of condition and consequence of an association rule is than the expected strength of their association.

Interesting measures take the expectation or the belief of how items are correlated into account. One can distinguish interesting measures into objective and subjective measures. Objective measures base on the statistical expectation of the correlation of items. Accordingly, a rule is interesting if the association of its items exceeds the statistically expected value of association. Contrary, subjective interesting measures take the belief of the analyst into account by trying to formalise what an analyst expects to be the relation of items. Hence, a rule is subjectively interesting if it contradicts the analyst's belief. The remainder of this section only surveys objective interesting measures. For a good overview of subjective interesting measures the interested reader is referred to [68]. Further, for a comprehensive discussion when to favour which interesting measure consider the reader is referred to [71].

The lift of a rule is an interesting measure that is defined as the quotient of the actual frequency of condition and consequence occurring together and the estimated probability of both occurring together under the assumption of statistical independence. In other words, lift indicates how many times measured probability exceeds estimated probability. Thus, the range of lift is the interval $[0; \infty)$. As the conditional probability cannot be measured it is approximated by its frequency.

Again, the support values of all concerned itemsets is sufficient to determine the lift of a rule $l(condition \Rightarrow consequence)$.

$$\begin{aligned} l(condition \Rightarrow consequence) &= \frac{s(condition \cup consequence)}{s(condition)s(consequence)} \\ &= \frac{c(condition \Rightarrow consequence)}{s(consequence)} \end{aligned}$$

Lift and support are contradictory in most cases. As all probabilities have

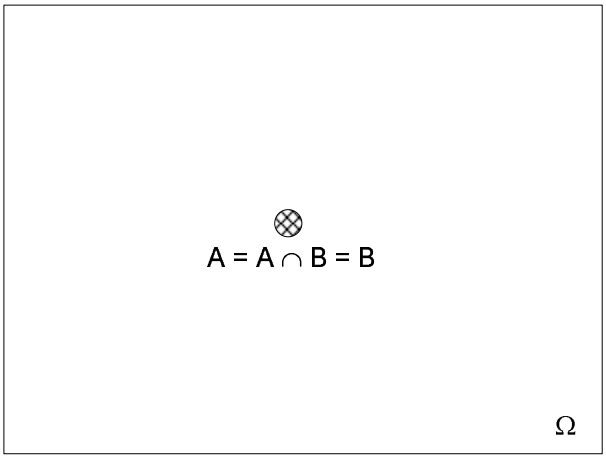


Figure 2.6: Frequencies of a set of infrequent itemsets where lift is at maximum but support is too low

1 as upper limit, the quotient that determines the lift has a maximum value that is lower the higher the probabilities of condition and consequence are. As probability of consequence or condition correlates with the support of a rule, support of a rule indirectly influences the maximum value of the lift of that rule. For instance, let the probability of condition assume 0.5 as is the probability of consequence. Consequentially, the expected probability of condition and consequence occurring together assumes 0.25. Due to the upper limit of the actual probability the lift of the according association rule may range from 0 to 4. Would the probabilities of condition and consequence be smaller, e.g. 0.25, one could receive higher lift values. Therefore, when considering lift one must also consider the value of support.

Alternatively, one can use other correlation measures such as the correlation coefficient to indicate interestingness of an association rule.

2.4.2 Determining Association Rules from Frequent Itemsets

As mentioned in the previous subsection, the support of an association rule must exceed a user-given minimum support to be considered statistically valid. As the support of an association rule equals the support of itemset that includes all items of a rule, the support of this itemset must also satisfy the minimum support—for shortness we will use the term itemset of a rule for this itemset.

According to Han and Kamber an itemset that satisfies the minimum support is referred to as frequent [36, p. 228].

If the itemset of a rule is frequent then the itemsets of condition and consequence must be frequent, too. If condition and consequence occur n times together, then each itemset must occur at least n times. Therefore, determin-

ing all frequent itemsets is necessary to find all association rules that fulfill the minimum support.

Once the frequent itemsets are determined, one can find association rules by trying to split a frequent itemset into two subsets: a subset of items for the condition and the rest of items for the consequence. As all subsets of a frequent itemset must also be frequent, all relevant elements to compute the association rule's confidence are available.

2.4.3 Apriori Algorithm

Agrawal and Srikant introduced the Apriori algorithm in [2]. As shown in algorithm 2, the Apriori algorithm determines the frequent itemsets in a set of transactions. Until FP-growth ([30] or see section 2.4.4) was introduced, Apriori was known as very efficient algorithm to find frequent itemsets because it avoids the generation of unnecessary candidates of itemsets.

For clarity of description, we denote an itemset consisting of k items a k-itemset. The smallest subset of a frequent itemset is an itemset with a single element, i.e. an 1-itemset.

Apriori consists of two steps that it iterates several times. The *join* step takes the frequent $(k-1)$-itemsets as an input and generates potential candidates of k-itemsets. The *prune* step scans the data once and determines the frequency of each candidate. Candidates that satisfy the minimum support remain in the set of frequent k-itemsets. The algorithm continues to perform *join* and *prune* steps until either the *prune* step finds no frequent candidates or the *join* step fails to generate candidates.

When combining two frequent $(k-1)$-itemsets to a candidate k-itemset, the *join* step takes those $(k-1)$-itemsets into account that differ only in a single item. That way, two new k-itemsets emerge: The one itemset has the item that has no pendant in the other $k-1$-itemset of the first $k-1$-itemset, while the other one has the according item of the second $(k-1)$-itemset.

In order to avoid combinatorial explosion of candidates, the *join* step uses the condition that each subset of a frequent itemset must be frequent, too. Thus, each subset of a candidate itemset must be frequent. Otherwise, creating that candidate is invalid. It is sufficient to test only if the subsets with $k-1$ elements are frequent because subsets of these subsets have already been tested in previous iterations.

Determining the 1-itemsets needs special treatment because it is impossible to derive candidates of 1-itemsets using 0-itemsets. Therefore, Apriori determines the 1-itemsets in an initial scan of the data set and counts the frequency of each item. It keeps those items as 1-itemsets that satisfy the minimum support.

Measuring the quality of Apriori result is irrelevant because Apriori always finds all frequent itemsets that satisfy the minimum support. Hence, runtime and space required memory are the only remaining measures of interest.

The runtime of Apriori depends on the number of transactions in the data set, the number of different items, and the minimum support. The number of iterations and the number of tuples influence the runtime of Apriori. Yet, the

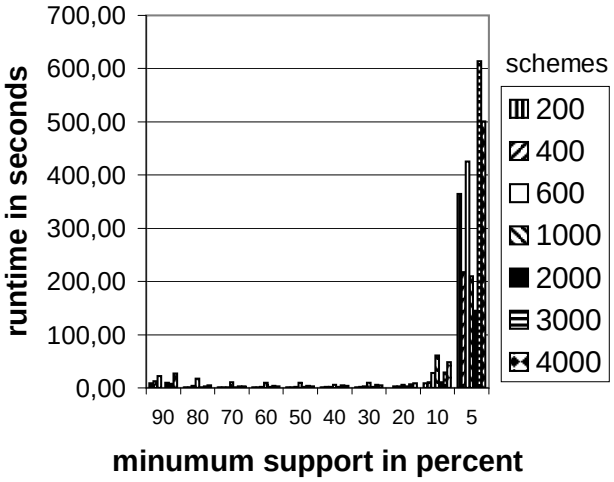


Figure 2.7: Combinatorial explosion due to low minimum support

number of iterations significantly depends on the chosen minimum support. The number of items causes as many iterations as the transaction with the largest number of items has when the worst case that minimum support is zero occurs.

However, choosing such a small minimum support means to choose a minimum support that might produce invalid and uninteresting results, as discussed in Section 2.4.1. Hence, one should avoid choosing a too low minimum support.

The combination of candidate itemsets can result in combinatorial explosion when the chosen minimum support is too low. In such a case, the condition that each subset must be frequent is true for too many candidates. If in addition to that the number of items is also reasonable then the main memory might be insufficient to store the candidate itemsets. Extensive swapping is the result. The runtime of a series of tests with a hundred different items shows exponential runtime with minimum support decreasing below 10 %, as shown in Figure 2.7. The tests analysed the associations of re-occurring items in process schemes. Therefore, the number of transactions, i.e. schemes, is small compared to other applications such as market basket analysis.

As presented in the experimental results, the basic principles of this dissertation are applicable to improve the runtime of Apriori by pre-computing the 1-itemsets. By doing this, Apriori saves exactly one scan. As discussed above, if the minimum support is sufficiently high, then this saving causes a significant reduction of runtime.

However, there is no general threshold that indicates when support is sufficiently high. Moreover, the data set influences whether a chosen support value is sufficiently high or not. A support of one of thousand can be high if there are millions of transactions. To be more specific, only the frequent itemset with the largest number of items determines the maximum number of scans. It typically increases with decreasing support. Yet, this happens gradually and depends on

the correlation of items. Thus, it is possible to have a support of 10 percent and higher and one still faces combinatorial explosion as depicted in Figure 2.7. Yet, it is also possible to have support that is more than a hundred times smaller and Apriori needs only a few scans.

When applying Apriori on the data set of the market basket analysis of the running example, Apriori would perform the analysis as follows. Assume that the minimum support is 40 %. Further assume the minimum confidence is 100 %.

After a first scan, Apriori would return the set of candidate 1-itemsets C_1 which is

$$C_1 = \{\{1\}, \{2\}, \{3\}, \{4\}\}$$

in the running example. All itemsets but itemset $\{2\}$ are frequent according to the minimum support of 40 %. Hence, the set of frequent 1-itemsets L_1 is

$$L_1 = C_1 \backslash \{2\}.$$

Before starting the second scan of the data, Apriori determines the candidates of 2-itemsets the frequency of which it will test during the second scan. By combining all possible combinations of frequent 1-itemsets to $1+1$-itemsets, we receive the candidates of 2-itemsets C_2 which is

$$C_2 = \{\{1,3\}, \{1,4\}, \{3,4\}\}.$$

By counting the frequency of 2-itemsets we observe that all candidates are frequent, i.e.

$$L_2 = C_2.$$

Combing the 2-itemsets to 3-itemsets we can find only one candidate itemset which is

$$C_3 = \{\{1,3,4\}\}.$$

As this itemset is not frequent, there are no 3-itemsets. Hence, Apriori is ready with finding frequent itemsets.

As there are only frequent 1-itemsets and 2-itemsets finding association rules is easy. The only itemsets that can be split into subsets are the 2-itemsets $\{1,3\}$, $\{1,4\}$, and $\{3,4\}$. By doing so, we receive the rule candidates $1 \rightarrow 3$, $3 \rightarrow 1$, $1 \rightarrow 4$, $4 \rightarrow 1$, $3 \rightarrow 4$, and $4 \rightarrow 3$. However, as the minimum confidence is 100 % none of the candidates satisfies minimum confidence. Hence, there are no association rules in the data of the running example according to the analyst-given parameters.

2.4.4 FP-growth Algorithm

FP-growth [30] is an algorithm that generates association rules. It works more efficiently than Apriori does because it needs no candidate generation. According to Han candidate generation is the bottleneck of Apriori and Apriori-like algorithms because the number of potential candidates might be subject to combinatorial explosion. Our tests with Apriori support his statement, as shown in

Algorithm 2 Pseudo code of Apriori algorithm

Apriori

Require: **items** $I = \{i_1, \ldots, i_d\}$, **set of transactions** $D = \{t_1, \ldots, t_n\}$ **with** $t \in D : t \in I^k$, **minimum support** s
Ensure: **frequent itemsets** $L = \{L_1, \ldots, L_k\}$ **with** $L_j \in L : L_j \subseteq I^j : \forall l_m \in L_j : \frac{|\{t \in D | l_m \subseteq t\}|}{|D|} \geq s, \forall c \in I^j \wedge c \notin L_j : \frac{|\{t \in D | c \subseteq t\}|}{|D|} < s$

1: $L_1 \leftarrow \left\{ l_m \in I^1 \,\middle|\, \frac{|\{t \in D | l_m \subseteq t\}|}{|D|} \geq s \right\}$;
2: $k \leftarrow 2$;
3: **while** $L_{k-1} \neq \{\}$ **do**
4: /* **join step** */
$C_k = \texttt{Join}(L_{k-1})$;
5: **if** $C_k \neq \{\}$ **then**
6: /* **prune step** */
$L_k \leftarrow \left\{ l_m \in C_k \,\middle|\, \frac{|\{t \in D | l_m \subseteq t\}|}{|D|} \geq s \right\}$;
7: **else**
8: $L_k \leftarrow \{\}$;
9: **end if**
10: $k \leftarrow k + 1$;
11: **end while**
12: **return** $\bigcup_k L_k$

Join (translated from [16, p. 162])

Require: **items** $I = \{i_1, \ldots, i_d\}$, **frequent itemsets** $L = \{L_1, \ldots, L_{k-1}\}, L_j \in L : L_j \subseteq I^j$
Ensure: **candidates** $C_k \subseteq I^k : \forall c \in C_k : \forall s \subset c : \exists L_j \in L : s \in L_j$

1: **insert into** C_k **select** $p.item_1, p.item_2, \ldots, p.item_{k-2}, p.item_{k-1}, q.item_{k-1}$ **from** $L_{k-1}\, p$, $L_{k-1}\, q$ **where** $(p.item_1 = q.item_1), (p.item_2 = q.item_2), \ldots, (p.item_{k-2} = q.item_{k-2}), (p.item_{k-1} \neq q.item_{k-1})$;
2: **for all itemset** $c \in C_k$ **do**
3: **for all subset** s **of** c **with** $k-1$ **elements do**
4: **if** $s \notin L_{k-1}$ **then**
5: **remove** c **from** C_k
6: **end if**
7: **end for**
8: **end for**
9: **return** C_k

Figure 2.7. Additionally, FP-growth always needs exactly two scans of the data which is less than Apriori typically needs.

Finding association rules with FP-growth is two-fold: Initially, the algorithms constructs a *Frequent Pattern* tree (FP-tree). Finally, it searches the FP-tree for frequent itemsets.

An FP-tree is a tree where each node has an associated item and a frequency. The frequency of a node determines how often the combination of the item of this node and the items of all ancestor nodes up to the root node occurs in a given data set.

The initial construction of the FP-tree first scans the data once to find all items that are frequent according to a user-given minimum support. This proceeding avoids that the FP-tree becomes too large due to too many infrequent items.

In addition to that, the algorithm also orders the items according their frequency with the most-frequent item first. The algorithm needs this ordering when it scans the data for the second time.

During the second scan the algorithm constructs an FP-tree as follows: The algorithm inserts a path into the FP-tree consisting of a sequence of frequent items for each tuple containing frequent items. If a path that is about to being inserted is already part of the FP-tree, the algorithm adds 1 to the frequency of occurrence of each node of that path. If a path does not exist in the FP-tree but a front part of the path is identical with another path, the algorithm inserts the non-overlapping part of the new path as a new branch of the last overlapping node into the tree. Additionally, it increases the frequency of overlapping nodes by 1. Hereby, the sequence of frequent items is in the order of the frequency of the items as determined in the initial scan. The ordering of items avoids that the same combination of frequent items is inserted more than once into the FP-tree.

The second phase of FP-growth finds all frequent itemsets as follows. It recursively scans the tree for frequent itemsets beginning at the root node of the FP-tree using depth-first traversal method.

For each visited node it performs the following steps: If the node has only a single path[1] then the algorithm tests the frequency of each combination of the items of that path for being greater than the minimum support. If so, it inserts the combination into a list of frequent itemsets. If the current node has more than a single path, the algorithm calls itself on a conditional subtree for each child node of the current node.

When constructing a conditional subtree of the FP-tree, FP-growth takes a subtree of the FP-tree but updates the frequencies of nodes where there are identical sub-paths in other parts of the FP-tree. This is necessary to find all frequent itemsets. When we exemplarily calculate frequent itemsets of the

[1] A path in a FP-tree is a sequence of nodes from a leaf node to the root node by iteratively ascending the parent axis of nodes. If a currently visited node is only part of only one path, we say that this node has only a single path. In Figure 2.8 all nodes but the root node and and the node labeled with item 1 have a single path.

(a)

item	frequency
1	4
3	4
4	3
2	1

(b)

item	frequency
1	4
3	4
4	3

Table 2.11: Frequency table of items of FP-growth after initial scan (a) before pruning and (b) after pruning

running example with FP-growth, we will see where updating frequencies is necessary.

Each time a step of recursion finishes, FP-growth adds the item of the root node of the currently traversed subtree to the frequent itemsets found in that subtree. This is necessary because all items in a subtree are occurring under the condition that the item of the root node of the subtree and all items of ancestor nodes of the root node in the FP-tree have occurred. In other words, as subtrees of the FP-tree are conditional, the frequent itemsets must include the condition of the subtree. If for instance the itemset $\{A, B\}$ is frequent in a subtree where the root node of this tree has item C associated with it, then the unconditioned itemset would be $\{A, B, C\}$ as long as the node of C has no ancestors. Otherwise, all items of ancestor node would have to be included in this frequent itemset additionally. FP-growth does this by adding the items of ancestor by ancestor to an itemset each time a recursion ends.

When applying FP-growth to the data set of the running example we should receive the same result as Apriori returned. For showing this, we compute the frequent itemsets with FP-growth. Minimum support is 40 % again.

After FP-growth has scanned the data once to determine the frequency of each item, it orders them according their frequency and removes non-frequent items, as depicted in table 2.11.

Let us consider the insertion of paths into FP-tree when scanning the data for the second time, as depicted in figure 2.8. When the first transaction $\{1, 2, 3\}$ is read, FP-growth eliminates the non-frequent item $\{2\}$ first before sorting the remaining items according to their frequency. As they are already in descending order of frequency, the sequence $(1, 3)$ is the result of this step. For the FP-tree being initially empty, there is no such path in the FP-tree. Hence, FP-growth inserts a node with item 1 and a node with item 3 into the tree and links the node with item 1 with the root node and inserts the other node as child of the node with item 1. When scanning the second transaction, we observe that the pruned version of the second transaction is identical with the first transaction. Hence, we would insert the same path $(1, 3)$ once again which is forbidden. Hence, we increase the frequency of each node in path $(1, 3)$ by 1. When reading the third transaction we receive a path $(1, 4)$ where the beginning of which is identical with a path already existing in the FP-tree. Consequentially, we add only the non-identical part of the path to the FP-tree. In other words, we add a new node for item 4 as a child node of the node of item 1. We update the frequency

of the common nodes of the existing path and the new path by 1 and set the frequency of the new node to 1. Inserting the path of the forth transaction needs to insert a new node to the existing path $(1, 3)$. Finally, the last transaction is a sequence $(3, 4)$ which is already part of the FP-tree. However, the paths $(3, 4)$ and $(1, 3, 4)$ are not identical in their initial elements. Hence, we must insert the path $(3, 4)$ as a new path in the tree because it is not yet part of it.

When traversing the root node of the FP-tree of the running example, we observe there are more than a single path in the tree. Hence, we create a subtree for each of the two child nodes of the root node and call the FP-growth algorithm on both of them.

When we construct the conditional subtree for the right branch of the unconditioned FP-tree, we have to correct the frequencies of the nodes in that branch because the same path also exists in the left branch of the unconditioned FP-tree, as illustrated in Figure 2.9. Otherwise, we would consider a too small frequency when determining the frequency of an itemset. Probably, we might exclude a frequent itemset when its frequency in a single branch is too small. By doing so, the frequency of the node of item 3 is now 4, while the frequency of the node of item 4 is 2. We see there is only a single path in this subtree. Hence, we can detect frequent itemsets in this subtree.

The itemsets $\{3\}$, $\{4\}$, and $\{3, 4\}$ are the only potential frequent itemsets in the conditional subtree of the right branch of the FP-tree. They are all frequent because their frequencies of $\frac{2}{5}$ and $\frac{4}{5}$, respectively, are greater than or equal the given minimum support.

Note that if we would have omitted updating the frequencies of items in the conditional subtree appropriately, we would have lost the frequent itemsets $\{4\}$ and $\{3, 4\}$.

In our example, constructing the conditional sub-tree is simple. Therefore, Figure 2.10 illustrates the construction of a bit more complex conditional subtree by adding an additional node into the left-most branch of the FP-tree. By doing so, the path of nodes 3-5-4 is no longer identical with the path 3-4 although path 3-5-4 contains path 3-4. As item 5 might be part of a frequent itemset, we must not omit it. Hence, we merge paths 3-4 and 3-5-4 as shown in Figure 2.10.

Analogously, we create a conditional subtree of the left branch of the FP-tree. Here, we need a second recursion because the subtree of the left branch has two paths. Determining of frequent itemsets happens analogously to the subtree of the right branch. We must update the frequency of item 4 in the right sub-subtree because the sequence "item 1 followed by item 4" exists transitively in the left sub-subtree. Contrary, we must not update items of the left branch because there is no "item 1 followed by item 3 followed by item 4" in another branch. FP-growth finds the frequent itemset $\{3\}$ in the left sub-subtree and frequent itemset $\{4\}$ in the right sub-subtree.

When considering the end of the second recursion of the left branch of the FP-tree, we can add the item of the node that is the root node of both sub-subtrees to the frequent itemsets the algorithm has found in each subtree. Hence, we receive the frequent itemsets $\{1, 3\}$ and $\{1, 4\}$ in addition to the already-found itemsets $\{3\}$ and $\{4\}$. Obviously, the root element $\{1\}$ is a fre-

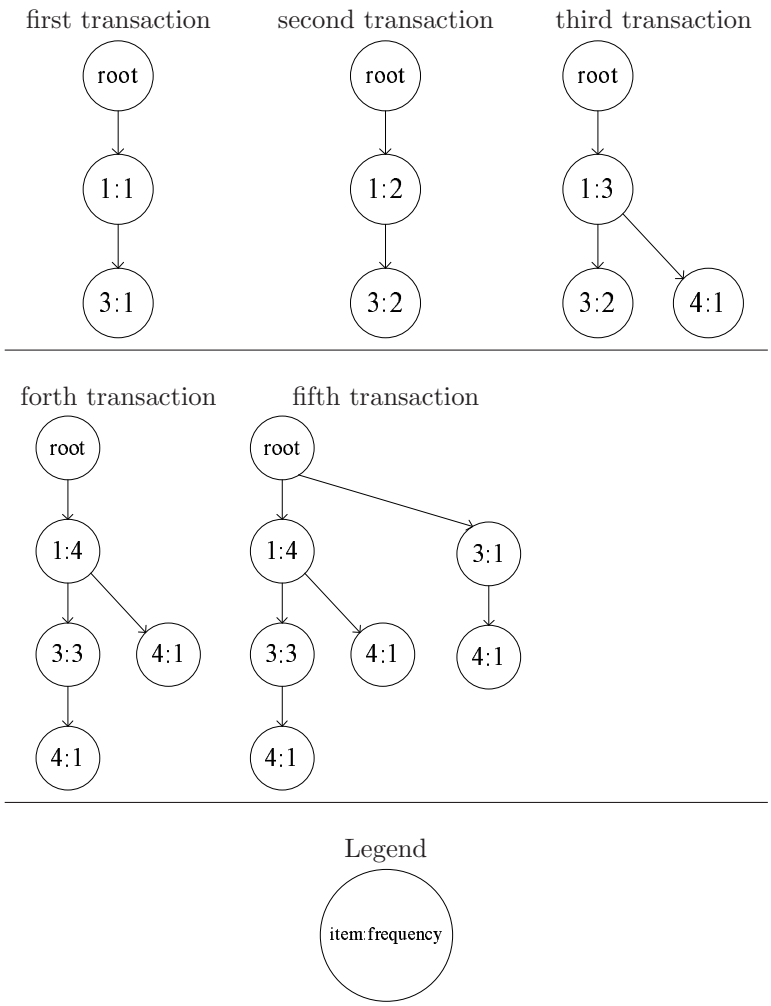


Figure 2.8: Growth of FP-tree when scanning transactions one by one

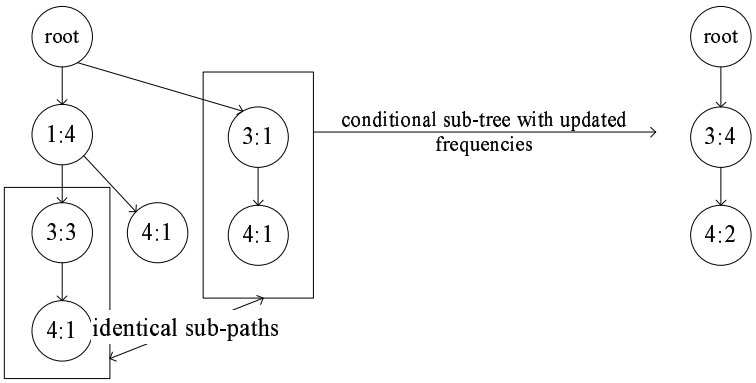


Figure 2.9: Creating conditional subtree

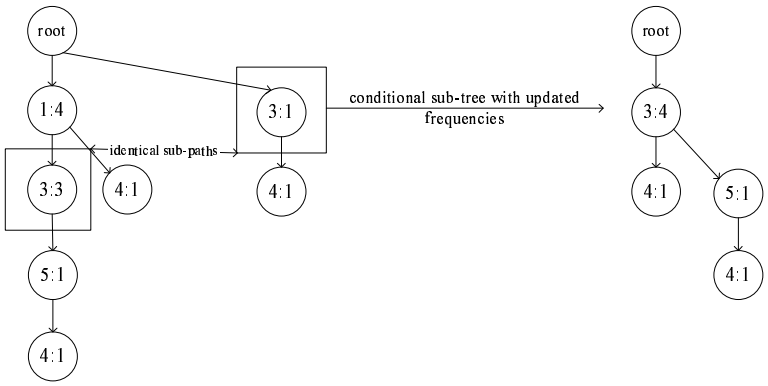


Figure 2.10: Creating conditional subtree with a single identical sub-path

quent itemset on its own.

When the recursion that was started first finally ends, the algorithm terminates. It needs not to add any item to the itemsets found in the subsets because the root node of the FP-tree has no item assigned with it.

Hence, FP-growth has found the frequent itemsets

$$\{\{1\}, \{3\}, \{4\}, \{1, 3\}, \{1, 4\}, \{3, 4\}\},$$

which is the identical set of frequent itemsets as found by Apriori.

Chapter 3

Discussing Existing Solutions

Contents

3.1 Introduction

The following sections give an overview of the state of the art of approaches trying to improve the performance of the *KDD* process or parts of it.

As mentioned in the introductory chapter, this dissertation presents techniques that are similar to techniques used for data warehousing.

Thus, Section 3.3 discusses the state of the art of data warehousing techniques that focus on improving OLAP sessions to demonstrate the differences as well as the analogous elements between the concepts used for data warehousing and the concepts presented in this dissertation, respectively.

As techniques such as sampling and data compression are applicable to speed up any kind of analysis, the description of these techniques can be found in separate sections, namely sections 3.2 and 3.4.

The other sections of this chapter describe previous work on improving data mining techniques—whereas each section presents approaching concerning the same data mining technique. To be more specific, Section 3.5 surveys related clustering approaches, Section 3.6 surveys related classification approaches, and last but not least Section 3.7 surveys related approaches of association rule mining.

3.2 Sampling

Sampling is the most-commonly used technique in data analyses when the amount of data is so large that performing an analysis would be too expensive otherwise.

Many *KDD* approaches use sampling in any kind of way to scale algorithms to large databases. For instance, CLARANS [53] iteratively takes samples from a data set to test the tuples of each sample if one of them is a medoid of the data set. Hence, this section discusses sampling method in general including its benefits and potential drawbacks.

The idea of using sampling is to take a small excerpt of a data set that is significantly smaller but has the same characteristics as the data set of which the sample is taken from. We will use the term *population* to denote the original data set. Hence, finding significant patterns in samples means that these patterns also exist in the population if the sample is representative for the population, i.e. sample and population share the same characteristics.

A sample can be biased or unbiased. In the average case, the distribution of tuples in an unbiased sample and in the population are identical—except some minor differences due to sampling error. Contrary, a biased sample favours specific subsets of the population, i.e. the biased sampling process selects them more likely than others. Favouring tuples with specific features can be beneficial for some analyses. For instance, when in a classification analysis a specific class is more interesting than another class, then favouring the interesting class can increase the prediction accuracy of the interesting class on behalf of the uninteresting class.

Yet, unintended biased sampling can worsen the quality of analyses using so-taken samples because the samples no longer share the same distribution of tuples with the population.

Uniform sampling method creates unbiased samples by selecting each tuple of a data set with the same *uniform* likelihood. Depending on the size of the sample, scanning the database or accessing tuples using point queries is more efficient. If a sample is very small compared to the population, it is better to retrieve the tuples of a sample tuple by tuple using point access. To do this, scanning the index is necessary. Yet, if the number of tuples in the sample is high scanning the database as a whole might be more beneficial than accessing tuples individually because efficient bulk accesses can decrease the total time needed for disk accesses.

When not further specified, we refer to uniform sampling when using the term *sampling*.

3.3 Analogous Solutions for OLAP

Data warehouses are designed for efficient querying them with OLAP. Ideally, the execution time for queries is so fast that analysts can interactively perform OLAP queries without being hampered by slow answering time of the OLAP system. However, the execution time of a query might be too long for interactive querying when the data warehouse is very large.

This section presents approaches that efficiently pre-compute data structure for interactive OLAP processing. Some of these techniques are very similar to approaches used for scaling of data mining algorithms.

Lakshmanan et al. present a data structure that stores aggregates of facts for efficient drill-down and roll-up operations in a data cubes which they call Quotient Cube tree [44], or QC-tree in short.

Other approaches such as the approach of Cuzzocrea [10] use a subset of the data to estimate the result of OLAP queries. The aim of these approaches is to give answers to queries early when only a part of tuples is scanned. Cuzzocrea's approach gives a probabilistic boundary for the approximated answers [10]. Yet, these approaches have to face the problem of representativeness of the subset of data they use, as we discussed in the previous section concerning samples. If an OLAP processer approximates a distributed query due to some processors have not answered the query yet, then the subset used for estimation is only representative if the distribution of tuples is the same for all processors—especially, there must not exist an ordering of tuples, i.e. a processor stores the tuples with a specific feature while another processor stores the tuples with another feature.

In contrast to that, the approach of this dissertation is always representative because it uses all tuples to derive statistics, i.e. it uses the entire population. Moreover, some of its statistics allow deterministic boundaries for approximated answers. Yet, we will discuss these statistics later in chapters 4 and 5.

3.4 Data Compression

While data analyses using samples use a small but representative subset of the data of which the samples are taken from, data analyses using any kind of data compression technique use another representation of the data but with redundant elements of the data removed.

For many *KDD* analyses it is sufficient to know some characteristics of the analysed data but not the tuples itself. For instance, to determine the k means of a run of the k-means clustering algorithm it is sufficient to know linear sum and number of tuples of each of the k clusters. In order to derive a linear regression model it is sufficient to know the correlation between each pair of attributes the analyst has selected for analysis.

Deligiannakis et al. present in [11] a data compression technique for data streams that approximates a data stream with a set of signals. Unlike to wavelet transformations there exists a dominant signal called base signal that functions in analogous way as the carrier way of a radio transmission does for the signals carrying the transferred information. The base signal stores the long-term trend while other statistics represent short-term deviations of that trend.

When compressing a data set, the compressed data represents the state of the database at the moment of compression. Hence, the compressed version of a data set is only a valid replacement of a data set for a given analysis, when the state of the database does not change in the interval between the moment of compression and the moment of doing the analysis.

If a compressed data set shall serve as replacement of a data set which might be subject to change then it is necessary to keep the compressed data up-to-date, too.

Algorithms that analyse data streams also require a mechanism that continuously updates either a solution or a compressed data set. Algorithms analysing streams using a compressed data set are capable to re-compute results when parameters change. Hence, they are superior to algorithms that incrementally update a single solution. Thus, all stream analysing algorithms using compressed data sets must be able to incrementally update a compressed data set. Moreover, one can use these algorithms to compress a database—instead of a data stream which is the type of data source they were designed for,

Clustering features, which we have discussed when introducing BIRCH, are common statistics to represent a compressed set of tuples. For keeping a clustering feature $(N, \vec{ls}, ss$ of a set of tuples up-to-date that has gained a tuple $\vec{x}$, one needs to add the clustering feature $(1, \vec{x}, \vec{x}^2)$ which represents tuple $\vec{x}$ to the clustering feature, i.e. we receive the updated clustering feature $(N+1, \vec{ls}+\vec{x}, ss+\vec{x}^2)$. BIRCH automatically updates the most similar clustering features when inserting new tuples [81].

Data bubbles are quadruples of summarising statistics that represent a set of compressed tuples. A data bubble consists of a representative of a set of tuples, the number of tuples in the data set represented by the data bubble, the radius of a sphere around the representative in which the majority tuples are located, and the estimated distance between any tuple of the data set and its

k nearest neighbours—k is a parameter the user has to specify when constructing data bubbles. Moreover, the representative is the centroid of the tuples represented by a data bubble. Nassar et al. show how to keep data bubbles up-to-date [52]. Updating data bubbles is similar as updating clustering features. Yet, the update of the estimated average distance is different. By assuming normal distribution of tuples around the centroid, Nassar et al. estimate the average nearest neighbour distance. If the estimated quality of compression of an updated data bubble is low, the algorithm of their approach re-builds a data bubble by re-scanning the tuples of that data bubble.

Data bubbles and clustering features represent the same type of information in two different ways because data bubbles can be transformed to clustering features and vice versa. The representative of a data bubble is the quotient of linear sum and count of tuples of a clustering feature. Additionally, the estimate of Nassar et al. uses the sum of squares to estimate the average nearest neighbour distance. By inverting the formulae of representative and average nearest neighbour distance, we receive the formulae to compute linear sum and sum of squares of the clustering feature that corresponds with a given data bubble. Hence, all algorithms using clustering features can use data bubbles that have been converted to clustering features. We will discuss several approaches using clustering features in the following sections.

3.5 Related Clustering Approaches

This section discusses related approaches focussing on improving clustering algorithms in terms of runtime and quality. Most of these approaches use some kind of index structure, compression technique, and/or sampling technique to increase the speed of algorithms.

Quality of clustering depends significantly on the used clustering algorithm. For instance, Hamerly and Elkan discuss the influence of chosen clustering algorithm on the quality of results [29].

Yet, the focus of this dissertation is on presenting a strategy that improves existing algorithms and not to argue in favour of a specific clustering algorithm.

The experimental results chapter of this dissertation shows that initial solutions have the most significant influence on quality of clustering results. Thus, we will discuss approaches to initialise clustering in a separate section, namely Section 5.10.

In the remainder of this section, we will discuss approaches that try to significantly improve runtime of a specific clustering algorithm without decreasing its quality—or with an insignificant reduction of quality. As the effect of initialisation on quality outweighs the other effects on quality, this reduction is tolerable.

Using index structures is a good way to improve the runtime of clustering. Depending on the type of chosen index structure, one can use it to access relevant tuples more easily. Some index structures are suited to be used as replacements of the tuples. Thus, accessing tuples is no longer needed. Hence, the following

subsections introduce index structures which are commonly used to improve clustering algorithms.

Another category of approaches uses sufficient statistics of the analysed data set to increase the speed of clustering algorithms. A statistic of a data set is sufficient when it contains the same information to determine a feature of interest as the tuples do. Hence, one can keep only summarising statistics and can neglect the summarised tuples. As many index structures use sufficient statistics, we distinguish in approaches using some kind of index structures which might use summarising statistics and approaches that use summarising statistics without index structures.

Finally, algorithms clustering streams must be able to consider tuples only once. Hence, they must be very fast and efficient. Therefore, algorithms clustering streams use only a single scan of the data set and are consequentially faster than partitioning clustering algorithms which usually take several scans of the data set. As a consequence of this, this section concludes with a discussion of algorithms clustering streams.

3.5.1 Using Sampling for Clustering

We discussed sampling as general technique to scale data mining algorithms for large data sets in Section 3.2, i.e. an analysts uses a sample as a replacement of the data set of which the sample is taken from. Additionally, several partitioning clustering algorithms use samples for finding an initial solution. As this is a very common method of initialisation, we will discuss initialisation separately in Section 5.10.

In contrast to these ways of using samples for clustering, this subsection discusses approaches where sampling is essential part of presented algorithms.

CLARANS [53] and CLARA are medoid-based partitioning clustering algorithms which use sampling. They return a set of k medoids of a data set. Hence, they produce the same type of result as the medoid-based partitioning clustering algorithm PAM [41] does without using sampling. Yet, CLARANS and CLARA perform significantly better than PAM.

PAM initially generates a candidate set of k medoids and determines the total distance of each tuple to its nearest medoid. In several iterations, it replaces a single medoid by a tuple which has not been a medoid, yet. If total distance decreases, this replacement becomes permanent. It repeats this replacement strategy until there is no improvement possible. Due to this working, PAM has to scan all tuples many times—making it an ill-scaling algorithm.

Contrary, CLARANS independently takes a fixed number of samples from the data set which it searches for medoids. Like PAM, CLARANS generates candidates of medoids and tests the total distance. For computing the total distance, it uses all tuples. Yet for selecting medoids, it uses only the sampled data. For limiting the maximum number of replacing medoids, it additionally uses a maximum number of tries of replacements that do not improve the quality of the result. CLARA also works with samples. Yet, CLARA iteratively takes samples and applies PAM on the sample.

3.5.2 Using *kd*-trees for Clustering

A *kd*-tree is a binary tree each node of which represents a choice in a single dimension of k dimensions, i.e. the left branch of that node represents a subspace that contains tuples with an attribute value lower than a given threshold value while the right branch of that node represents a subspace with tuples with higher attribute values than the the threshold value. Hence, the *kd*-tree iteratively splits a vector space into several disjoint and exhaustive subspaces.

The 'filtering algorithm' of Kanungo et al. [40] is an adaption of the k-means algorithm of Forgy that uses a *kd*-tree to improve the runtime of k-means.

First, the algorithm constructs a binary tree upon the data. It recursively divides the vector space that is spanned by a set of tuples into two subspaces that have approximately the same number of tuples in them. The result is a binary tree in which each node represents a subspace of the vector space that is spanned by all tuples. A k-means-like function uses this tree to compute a set of means more efficient than k-means does.

The k-means-like function of the filtering algorithm scans the tree several times and determines the means of k clusters—which is similar to the k-means version of Forgy [17]. Yet, k-means scans tuples, not a tree.

Additionally, the assignment of nodes to clusters is different, too. Initially, each node of the tree is assigned to all clusters.

When scanning the tree, the filtering algorithm tests if it can exclude one or more clusters as nearest cluster of all tuples in a node. If a mean of a cluster is always farer away from the tuples in a node than another cluster's mean, then the filtering algorithm can exclude the cluster that is farer away. As long as there is still more than a cluster left as potential nearest cluster of a node, the filtering algorithm also examines the children of that node.

If there is only one candidate cluster left in a specific node, the filtering algorithm assigns that node and all sub-nodes to that cluster.

The filtering algorithm is very efficient to reduce the cost of assigning tuples to means because it tests a complete subspace at a time instead of testing each tuple individually.

Alsabti et al. present almost the same approach in [4].

As the filtering algorithm uses the Forgy variant of k-means instead of the McQueen variant, the number of iterations which the filtering algorithm needs is typically higher than approaches that use the McQueen variant.

Additionally, the time needed for constructing the tree might exceed the runtime needed for the k-means application. The runtime complexity of constructing a *kd*-tree is $\mathcal{O}(n \log n)$ [3]. Thus, if $\log n$ exceeds the number of iterations needed by k-means, the filtering algorithm becomes insufficient.

For the reasons given above, the approach of this dissertation uses other index structures than *kd*-trees for clustering, namely extended versions of clustering feature trees.

Using SS-trees for Clustering

A *Similarity Search*-tree or short SS-tree [76] is a multidimensional hierarchical index structure where each tuple is assigned to the leaf node the centroid of which is nearest to the tuple. Thus, the space in which all tuples of a node are located is a sphere in a multidimensional vector space. Inserting tuples into an SS-tree is analogous to R^+-tree. R^+-trees and SS-tree are very similar but all tuples of a node of an R^+-tree are in a cube in a multidimensional vector space while all tuples of a node of an SS-tree are in a sphere.

Trees of data bubbles, as approaches such as [52] produce them, are SS-trees enriched with additional elements. The representative of a data bubble is the centre of a sphere. Additionally, a data bubble includes the radius of that sphere.

3.5.3 Using Sufficient Statistics to Improve k-means or *EM* clustering

As mentioned above, a statistic of a data set is sufficient when it contains the same information to determine a feature of interest as the tuples do. Hence, one can keep only summarising statistics and can neglect the summarised tuples. Clustering features are sufficient statistics to determine Gaussian clusters.

This subsection presents approaches that use sufficient statistics in general or clustering features in special to compute the result of an *EM* clustering algorithm. Yet, one can also use clustering features as index structure in the form of a clustering feature tree. Hence, this subsection also discusses approaches that use trees of clustering features.

Using clustering features for *EM* clustering is subject to results with low quality when the variance of a clustering feature is zero. If the variance of a clustering feature is zero, i.e. all tuples share the same attribute values, then several calculations necessary for *EM* such as the probability density at a specific point become undefined because the variance is included in the nominator of a fraction.

FREM is an *EM* like clustering algorithm which uses clustering features to increase its speed [58]. It solves the problem of undefined variances by adding a small constant to the variance of a clustering feature. The constant is small enough such that it does not influence the assignment of tuples to clusters. Yet, it is high enough to prevent computational instability due to zero-valued variance.

Approaches using sufficient statistics for *EM* clustering have in common that they construct sets of sufficient statistics in a first step before using them as replacement of tuples in a second step.

The approach of Bradley et al. [6] scans the data set of an analysis only once and tests each tuple if it can compress the tuple. The approach packs tuples which are very similar to each other in a subset of data and computes a clustering feature for each package. It maintains several lists of clustering features that vary in the rate of compression. It inserts tuples that are unlikely

to be tuples that significantly influence the result of an *EM* clustering into a list of clustering features with high compression rate. Contrary, it inserts tuples that are likely significant tuples into a list of uncompressed tuples.

Jin et al. compare several approaches using sufficient statistics or sampling for *EM* clustering in their paper [37]. The set of sufficient statistics used by Jin et al. is a triple consisting of count, centroid and average sum of squares of the tuples summarised by this triple. Obviously, this triple contains the same information as a clustering feature—yet, a clustering feature contains no averaged values but the total sums. A method of the approach of Jin et al. lays a multi-dimensional grid over the vector space in which the tuples of a data set are located and determine the above-mentioned triple of sufficient statistics for each non-empty cell of the grid. The other method of determining the sufficient statistics is to use BIRCH. Yet, many approaches use BIRCH to compute a tree of clustering features. Due to the amount of approaches, we will discuss this option separately in the next subsection.

Bradley et al. and Jin et al. gain independently the insight that compressing tuples inflicts results superior in terms of quality to results of clustering samples but is as fast as sampling [6][37]. Therefore, the approach of this dissertation focusses on compressing tuples using extended clustering features instead of taking samples. To be more specific, this dissertation includes an approach that extends clustering features to use them for multiple analyses with varying settings. It also includes techniques to derive specific representations of a data set to fit a set of analyses from a general representation of a data set. The specific representations of data are trees of clustering features which one might use to apply the approaches of Bradley et al. or Jin et al. on them. Thus, the benchmark results of both approaches apply in the same way for the approach of this dissertation.

Bin Zhang et al. present a clustering algorithm using a local search heuristic to re-compute centroids of clusters [79]. For improving runtime they split the data set into several small partitions which they represent by summarising statistics that contain the same information as clustering features.

Trees enable retrieving summarised statistics at different levels of granularity on demand. For this reason the approach of this dissertation stores summarising statistics in a tree.

Tian Zhang introduces an improved clustering algorithm called BIRCH [81] which consists of a hierarchical clustering that generates a tree of clustering features followed by partitioning clustering of leaf node entries in his dissertation [80] but implements only hierarchical clustering [1]. However, several authors have implemented his initial idea.

O'Callaghan et al.[55] use BIRCH to maintain a tree of summarising statistics to use it in a modified version of k-means. They extended their approach to k-medoids in [27]. Due to the above-mentioned properties of BIRCH, this approach is limited to analyses in a single vector space.

[1] We examined the source code of Tian Zhang's prototype which he used for testing which we obtained from Tian Zhang's web site.

The drawback of using BIRCH for partitioning clustering is that it is necessary to re-run the first phase of BIRCH when only a subset of data or attributes is relevant for a given cluster analysis. Yet, the first phase of BIRCH is the most time-consuming phase of all its phases because it is the only phase that accesses data from persistent storage—all other phases use several scans of data that is already loaded in the main memory.

Chiu et al. [9] present an expectation maximisation (EM) algorithm using summarised statistics organised as a cluster feature tree. A drawback of their method is that the dendrogram has to be constructed for each different set of selected attributes.

The algorithm presented in this dissertation is very similar to BIRCH but uses different summarising statistics to make it generally applicable. Hence, this set of summarising statistics is a called a general cluster feature. To be more specific, the approach of this dissertation supplies methods to adapt a tree of general cluster features to fit the needs of an analysis that needs only subsets of this tree. Moreover, these method allow the construction of new attributes and also give deterministic thresholds of quality measures.

3.5.4 Applying Partitioning Clustering to Data Streams

Clustering algorithms must be able to generate their result in a single scan to be applicable to data streams. As partitioning clustering algorithms typically scan the data multiple times, they need adaption to become applicable to streams.

Such an adaption also inflicts a significant improve in runtime because only one scan is needed after adaption. Thus, approaches adapting partitioning clustering algorithm to data streams are related work of this dissertation.

There has been much work recently on clustering data streams. Among this work, there are approaches modifying clustering algorithms like k-means to be applicable to streams. Similar to that, other approaches try to represent data streams with summarising statistics to use them for all kind of analyses.

Applying k-means to Streams

One of the most popular partitioning clustering algorithms is k-means which has been first introduced in two versions in 1965 by Forgy [17] and in 1967 by McQueen [46]. Yet, as k-means requires several scans of the data, it is inapplicable to data streams. Several approaches try to modify k-means to break this limitation.

Scanning the data only once and tolerating the so-induced error is a potential way to apply k-means on data streams.

Incremental kmeans [56] is a variant of k-means that scans the data only once. This approach utilises the observation that major improvements occur during the first iteration while further iterations only slightly modify the result of the first iteration. This approach has several drawbacks, namely the lack of re-doing analyses with different parameters, the induced error, and the way the algorithm is initialised.

First, re-doing analyses with different parameters is impossible but necessary to tell apart whether the difference between two analyses is caused by different parameters or different data—the latter would be an indicator for a trend while the first is an indicator for an ill-performed analysis. Yet, without storing the data stream's content, the tuples that are required for a re-scan are no longer present. However, storing the stream's content is omitted because it is very expensive.

Second, the error caused by scanning the data only once, is not negligible in general. Thus, Ordonez suggests applying *incremental kmeans* only to binary data streams. According to his paper [56] the resulting error is tolerable when the used attributes are binary. Yet, this condition is too restrictive for generally using *incremental kmeans*.

Finally, the way *incremental kmeans* is initialised might cause bad results because it depends on the order the data is scanned. *Incremental kmeans* buffers a part of the data stream which it uses to find a suitable initialisation. Once the algorithm is initialised with a set of k means, it scans the remaining stream and updates the means in regular intervals. Thus, the tuples that *incremental kmeans* reads first influence the algorithm more than tuples that the algorithm reads later on. If there are trends in the data such as that a cluster is changing over time then *incremental kmeans* might fail to find a suitable result because the old location of a cluster's mean influences the result too much.

Nasraoui et al. introduce in [51] an immune system learning model for clustering noisy streams. Yet, their approach is also incapable to re-do analyses because data is no longer present. Hence, it shares the same drawbacks with *incremental kmeans*.

As scanning once and tolerating error is only applicable to a very restricted number of applications, the concept of this dissertation takes no use of this idea.

Reducing the number of tuples by taking a sample is a common solution when a data set is too big for being clustered in a given time frame.

As sampling is very common, the test series in the experiments chapter benchmark against sampling. The test results show that under certain circumstances sampling can deliver better results than the approach shown in this dissertation does. However, there are several advantages of this dissertation's approach compared with sampling which are guaranteed error, re-usability[2], and runtime.

When using a sample of a data set for clustering, the result of this clustering might differ from the result using the total data set. We call the error that a percentage of tuples are assigned to clusters erroneously sampling error when other sources of error such as initialisation can be excluded.

Using Sufficient Statistics for Clustering Streams

Several approaches use aggregated representations of the data instead of the data self. A common method is maintaining a set of summarising statistics of

[2] One can re-use samples, too. Yet, re-using samples negatively affects the quality of results. We will discuss re-using samples in detail in Subsection 5.10.2.

the data. As long as such a representation can be updated incrementally, it is possible to use it as a surrogate of the data in a stream.

Algorithms using incrementally updated trees of clustering features are able to cluster data streams. In order to avoid duplicated presentation of algorithms, this subsection presents only those approaches that are specially designed for being applied on data streams.

The approach of Aggarwal et al.[1] maintains a set of summarising statistics representing a data stream. The authors demonstrate a temporal selection function that might be used to compare cluster models in different time intervals.

The approach of this dissertation generalises their approach by defining a selection function for any set of attributes—time is just another attribute for selection. In addition to that, the approach of this dissertation is able to project attributes and derive new attributes from existing attributes.

3.6 Related Classification Approaches

3.6.1 Bayes Classifier

There exist several approaches using a set of statistics as sufficient statistics for naïve Bayes classification. Yet, these statistics are frequencies of attribute values, commonly organised in a tree.

Moore and Lee cache sufficient statistics for classes of classification algorithms such as naïve Bayes and decision trees in a data structure they call *AD*tree [49], which is a *kd*-tree the inner nodes of which are annotated with frequencies of attribute values. An *AD*tree is able to reduce the need of space to store frequencies of attribute values but it is only applicable to non-continuous attributes. An *AD*tree uses loss-less compression of tuples. The experiments in Section 7.5 have shown that one can compress tuples in a tree of clustering features with potential losses without decrease in accuracy when the number of nodes exceeds some hundreds of nodes. Obviously, some hundred nodes are sufficient to store as many top-frequent pairs of attribute values as needed such that only rarely the frequency of a missing pair is needed. Additionally, the approach of this dissertation is capable to handle continuous attributes and offers a selection operation to re-use frequencies for analyses needing only a subset of the original data set. Caragea uses also only frequencies of pairs of attribute values as sufficient statistics [8].

3.6.2 Decision Trees

Jordan suggests in [39] to combine decision tree algorithms with approaches generating statistical models such as hidden Markov chains or *EM* clustering to improve the quality of resulting decision trees. Jordan's approach is similar to the approach mentioned in chapter 6 because it also uses a sequence of algorithms to improve the quality of classification. Yet, the herein-presented approach can also decrease the time needed to compute the combination of both

algorithms. Hence, one can use the approach we will introduce in chapter 6 to accelerate Jordan's approach because the general principles of both approaches do not contradict each other. Dobra and Gehrke present SECRET which which is also a combination of *EM* clustering algorithm and decision tree construction [13].

The approach of Ganti et al. [20] trains a decision tree from data in a stream and updates it regularly. For doing so, it divides a stream in blocks of fixed sizes and stores summarising statistics for each block. In regular intervals new blocks arrive and old blocks become deleted, i.e. if the main memory is full, their algorithm discards the oldest blocks.

The approach of this dissertation includes the same type of summarising statistics. Hence, one could use these statistics to produce the same result as Ganti et al.'s approach would do. In contrast to the approach of Ganti et al., the approach of this dissertation is able to re-analyse decision trees of past data because it is able to re-construct the database state of the past.

3.7 Related Approaches of Association Rule Mining

The intention of the approach in this dissertation is to perform tasks or subsets of them now that might be used later to reduce the time an analyst needs to perform an analysis. Hereby, the type of succeeding analysis is irrelevant—which is the discriminating factor of this approach with related approaches.

This section presents approaches to improve the time an analysts needs to perform an association rule analysis. Some of these algorithms focus on re-use of previous results such as [50] but none of them uses a previous result for an analysis of another type of analysis than association rule mining.

Algorithms mining association rules such as Apriori and FP-growth perform a complete search for association rules, i.e. they find all association rules that satisfy minimum support and minimum confidence. Hence, there is no quality improvement of the found result possible.

However, minimum support and minimum confidence depend on the analyst and might be small. If an analyst has chosen very small values for minimum support or minimum confidence, there can be more found association rules the analyst can handle [42]. Additionally, in cases where there are many rules the performance of algorithms is also low, as indicated in section 2.4.1.

Consequentially, approaches improving association rule mining either try to reduce the number of found rules to find only the most interesting rules or try to improve the performance of algorithms.

3.7.1 Updating Association Rules According to Changing Constraints

Mining association rules with constraints is a common use case of association rule mining. Constraints, conditions that transactions must fulfill to be relevant

for an association rule analysis, give the analyst the opportunity to focus an analysis on specific aspects of the analysed data [54]. There exist several types of constraints such as row-level constraints. A row-level constraint is a predicate that is either true or false for a tuple when applied on a data set. Hereby, the logical value of the predicate depends only on the values of the currently considered tuple. For instance, if an analyst performs a market basket analysis of only those transactions that customers with a specific feature have committed, he applies a row-level constraint to the set of transactions because it is possible to evaluate this constraint for a tuple without considering another tuple.

When a company analyses market baskets, the data source is always a table storing sales data. However, the relevant subset of that data source might differ from analysis to analysis. In other words, there might be several different constraints concerning the same data set.

For reducing the average time necessary to compute the result of an association rule analysis there exist several approaches to re-use the results of an association rule analysis for determining the result of another association rule analysis with different constraints.

The approach of Nag et al. uses a cache of frequent itemsets to speed up association rule mining with different constraints [50]. Whenever a constraint is changing, Nag et al. use Apriori to re-compute an association rule analysis. Hereby, the cached itemsets can reduce time needed for candidate generation. Additionally, in some cases Apriori can save scans as the cache might already contain all relevant itemsets. However, the occurrence of such events is stochastic. They test several replacement strategies of the cache, i.e. strategies that decide which itemsets shall remain in the cache when the number of found itemsets exceeds the capacity of the cache.

According to Nag et al.'s tests the *benefit replacement* is the most beneficial strategy to replace itemsets of the cache storing itemsets. It uses a B^+-tree to index itemsets according their support value. Hence, a simple range query can do the accessing of itemsets satisfying a specific minimum support.

The approach of this dissertation also utilises caching of frequent combinations of attribute values to improve the average response time of the *KDD* system. However, the approach of this dissertation is more general than Nag et al.'s approach because it is not limited to itemsets, only. Itemsets are combinations of attribute values of the same attribute in a transaction while the approach of this dissertation uses attribute value combinations of arbitrary attributes, as demonstrated in section 6.5.

Hipp and Güntzer show in [33] that it is possible to construct the result of an association rule analysis with row-level constraints using only the result of an unconstrained association rule analysis. According to their experience row-level constraints commonly consist only of conditions of items such as analysing only products of a specific product group in a market basket analyses. That way there exists an unconstrained itemset corresponding to each constrained itemset which means that constrained itemset and unconstrained itemset reference to the same type of event. For instance, the itemset "customer buys shoes and socks" under the condition that this customer also buys jeans references to the

same event as itemset "customer buys shoes, socks, and jeans" does. In general, the constrained itemset *condition* ∪ *consequence* under an itemset *constraint*, which represents a constraint, corresponds to the unconstrained itemset *condition* ∪ *consequence* ∪ *constraint*. In other words, as constraint, condition, and consequence are itemsets, the itemset that fulfills condition and consequence under the given constraint, is the itemset that includes all items of constraint, condition, and consequence.

If the itemset representing the constraint is frequent then performing a single unconstrained association rule analysis is sufficient to compute the results of constrained association rules analyses with row-level constraints consisting only of items.

If an analyst wants to use features that are no items as part of a constraint, then the analyst can circumvent this problem by introducing those features as items into the data set storing the transactions.

The approach of Hipp and Güntzer is able to reduce the time an analyst needs for an association rule analysis because a *KDD* system can perform the time-consuming task of determining of frequent itemsets once before the analysts interacts with the system: The analyst needs not to wait for the system performing time-consuming computations but can concentrate on choosing constraints and evaluating patterns.

However, determining the unconstrained frequent itemsets is more likely subject to combinatorial explosion because the same frequent itemsets typically have less support when they are unconstrained than when they are constrained. If the constrained itemset *condition* ∪ *consequence* shall be frequent under the constraint *constraint*, the fraction $\frac{s(condition \cup consequence \cup constraint)}{s(constraint)}$ must satisfy minimum support. If the support of the constraint is less than 1, then the according limit that determines whether an unconstrained itemset is frequent or not is lower than the minimum support of an association rule analysis with constraints. Hence, if one wants to make a constrained analysis with a specific minimum support, he or she must have done the unconstrained analysis with an accordingly lower minimum support because it is possible to increase the minimum support afterwards but impossible to decrease it.

Summarising, the approach of Hipp and Güntzer needs a tradeoff of pros and cons, i.e. a tradeoff between a cost saving due to pre-computed results and potential combinatorial explosion due to too low minimum support and too many different items. However, it is always possible to try if an unconstrained analysis with an appropriately small minimum support is subject to combinatorial explosion. If not, one can use the results for constrained analyses in the future. Otherwise, one has to consider other approaches.

The approach of Hipp and Güntzer represents an idea for constrained association rules that is very similar to the approach CHAD presented in chapter 5 which is a clustering algorithm. Due to the similarity of both approaches, we illustrate the approach of Hipp and Güntzer using the running example.

To exemplify this approach, consider the market basket analysis we have discussed in Section 2.4 once more—yet, now with focus on re-using constraints.

cid	timestamp	prid
4711	2005-01-05 16:15	1
4711	2005-01-05 16:15	2
4711	2005-01-05 16:15	3
4711	2005-01-05 16:15	-1
4711	2005-04-01 09:23	1
4711	2005-04-01 09:23	3
4711	2005-04-01 09:23	-1
0815	2005-04-01 09:27	1
0815	2005-04-01 09:27	4
0815	2005-04-01 09:27	-1
1704	2005-05-05 13:13	1
1704	2005-05-05 13:13	3
1704	2005-05-05 13:13	4
1704	2005-05-05 13:13	-2
1704	2005-06-05 10:30	3
1704	2005-06-05 10:30	4
1704	2005-06-05 10:30	-2

transactions
1, 2, 3, corporate
1, 3, corporate
1, 4, corporate
1, 3, 4, individual
3, 4, individual

Table 3.1: Clip of the tuples of the table that an analyst has pre-processed for a market basket analysis with constraints

Assume that an analyst of the publishing company of the running example wants to perform a set market basket analyses of customers having a specific type or ordering a specific group of products. The attribute *customer type* has two values *corporate* and *individual* which both are no items. Adding these values as additional items means to insert a new tuple that indicates the sale of a virtual product *corporate* to each transaction that is initiated by a corporate customer. A simple SQL insert statement can do this:

```
INSERT INTO marketbasketdata (cid, timestamp, prid)
  (SELECT DISTINCT (marketbasketdata.cid, timestamp), -1
   FROM marketbasketdata, customer
   WHERE customer.cid = marketbasketdata.cid
   AND customertype = 'corporate');
```

After inserting all values as items we receive a modified table of transactions as depicted in table 3.1. Determining the frequent itemsets of the table with additionally inserted items happens in conventional way as demonstrated in Section 2.4.

If we want to check if the rule product 1 $\rightarrow$ product 3 is valid according minimum support 40 % and minimum confidence 70 % under the constraint that the customer must be a corporate customer, we have to determine constrained support and confidence first. The constrained support of a rule is the unconstrained support of that rule divided by the support of the constraint [33]. The constrained itemset $\{1, 3\}$ under constraint $\{corporate\}$ corresponds with

the unconstrained itemset $\{1, 3, corporate\}$. Hence, the constrained support is $\frac{s(\{1,3,corporate\})}{s(\{corporate\})} = \frac{2}{3}$. Analogously, we determine the confidence of that rule as $\frac{s(\{1,3,corporate\})}{s(\{1,corporate\})} = \frac{2}{3}$. Therefore, the rule product 1 $\rightarrow$ product 3 fulfills minimum support but not minimum confidence. Hence, the rule is no valid association rule.

3.7.2 Replacing the Minimum Support Condition

Using the minimum support for restricting the number of frequent itemsets is two-edged: On the one hand, a high support can significantly reduce the number of frequent itemsets. However, there might be many interesting rules that lack sufficient frequency. On the other hand, if minimum support is small, many itemsets become frequent regardless they are interesting or not. Zijian Zheng et al. tested the performance of association rule algorithms on various real world data sets and found exponential growth of number of association rules when lowering minimum support, confidence, or lift [82].

Hence, there exist approaches that try to circumvent the above-mentioned problem by relaxing the strict minimum support condition.

The approach of Liu et al. uses different levels of minimum support where rare but interesting itemsets have a lower minimum support value [45]. The height of minimum support which an itemset must satisfy depends on the interestingness of its items. In other words, the less the frequency of a combination of items can be explained by chance the lower is the more interesting is that combination and the lower is the minimum support it must satisfy.

Wu et al. use statistical modeling to find association rules [77]. Therefore, they significantly differ from other approaches aiming to find association rules such as Apriori [2]. Each combination of items the frequency of which in a given data set is unable to be explained by a log-liner model of that items is an interesting itemset in that data set.

3.7.3 Approaches avoiding limits of Apriori

Sung et al. use sampling to forecast association rules in a data set that has not yet been analysed for association rules [70]. DuMouchel and Pregibon present a similar approach [14]. Yet, DuMouchel and Pregibon use Bayes statistics. Therefore, these approaches significantly reduce the time needed for scanning the data set for frequent itemsets. Hence, it is faster than Apriori algorithm. Yet, as fp-growth needs only two scans to determine all frequent itemsets of a data set without sampling error, the approach of Sung et al. offers only a slight improvement in speed when compared with fp-growth. However, the approach of Sung et al. remains beneficial because it enables analysts to compare association rules of a data set with those rules that one should observe due to experience made when analysing similar data sets.

Webb suggests to search association rules directly instead of using the two-folded process of finding frequent itemsets and association rules [74]. His ap-

proach applies support, confidence, and interesting measures when scanning the database. This process avoids combinatorial explosion as the interesting measures significantly limit the number of association rules. Yet, re-scanning is necessary each time the analyst changes any parameter.

3.7.4 Reducing the Need of Pre-Processing

Apriori and FP-growth use items which are stored in a single attribute, i.e. they interpret the attribute values of a single attribute as items. Perng et al.'s approach is able to find association rules between items which might be attribute values of a set of attributes [60]. Hence, their approach extends association rule analyses to data sets without pre-processing them to fit a pre-defined form—in an association rule analysis with Apriori or FP-growth, an attribute has to function as identifier of attributes while another attribute identifies items. The techniques proposed in this dissertation are able to offer pre-computed intermediate results for association rule analyses following the approach of Perng et al. as one can see in chapter 6.

Chapter 4

Concept of Anticipatory Data Mining for Improving the *KDD* Process

Contents

4.1 Things to Improve

Improving the *KDD* process means that the modifications of the *KDD* process result in a new *KDD* process which either is quicker or has results of higher quality in the average of its process instances. As improving runtime is possible by tolerating results having low quality and vice versa, we demand that a modification improving either time or quality only worsens the other dimension of improvement in an insignificant way, which means that a significant runtime improvement should not reduce quality in the optimal case—or only marginally.

Before improving the *KDD* process the drawbacks of the *KDD* process as is must be known. As mentioned in the introductory chapter, viewing the instances of the *KDD* process as isolated instances is the main drawback of the *KDD* process. However, this drawback causes several subproblems which are described together with the idea of their solution in the latter of this section:

doing the same things multiple times Viewing the *KDD* process as a process with single isolated instances is bad because different instances of that process require similar tasks with the same data. In an instance of such a process, the minimum number of scans needed is one. Lowering this bound is impossible.

If the isolated view of instances is given up, the minimum number of scans is still one—but the new limit is one scan for a set of instances.

neglecting experience of previous *KDD* instances Almost every instance of the *KDD* process increases the analyst's knowledge of the relations in the analysed data. The analyst chooses data mining techniques for further examination of data more efficiently with increasing knowledge.

However, most data mining algorithms operate as if nothing is known about the data to be analysed. However, results of previous *KDD* process instances can improve the quality of further instances. For instance, if the type of distribution of an attribute and the distribution's parameters are known, classifying algorithms can use this piece of information to train a classifier that is more accurate than the classifier that the algorithm would produce without the additional information concerning distribution of that attribute.

Previous work has presented several approaches to improve the results of data mining algorithms with additional knowledge.

This dissertation examines the cost of determining and storing additional information about the data being analysed and their potential effect on future *KDD* instances. In other words, it provides the answers to the questions "What kind of (intermediate) result should be stored to be used in future applications?" and "Is there an efficient determination of this type of information?". If a specific type of additional information is cheap to determine and potentially very useful, it should be determined when scanning a set of data—even it is not needed in the current *KDD* instance.

neglecting typical sequences of data mining techniques There is a small set of different types of data mining techniques such as classification, clustering, and association rule analysis—to name only the most-commonly used ones. Yet, complex analyses use a combination of several techniques to determine a suitable result. For instance, web usage mining uses association rule analysis of access log files to determine rules of navigation of users within a web site as its major data mining technique. Yet, classification can be used to filter page accesses of web crawlers of search engines that are indexing web sites. Accesses of web crawlers do not represent navigation rules of users but would influence the resulting association rules. Hence, they must be eliminated before beginning an association rule analysis.

This dissertation shows how to enhance popular complex analyses by determining additional information in the early phases of the *KDD* instance of such an analysis and using this information in later phases.

By giving up the isolated view on instances of the *KDD* process there is much room for improvement, as the above-mentioned items indicate.

If it is known that there will be other data mining algorithms doing some kind of analysis on the data, a currently running task can do more than computing its own result. For instance, it can prepare intermediate results for future analyses. As there are no additional disk accesses needed this is a good option to save time for succeeding algorithms with low cost for the anteceding algorithm.

The tests in the experiments' chapter show that the cost for computing additional results are low and the benefit of pre-computed items is high.

Thus, it is a good idea to compute intermediate results as early as possible—even in anticipation of a instance of the *KDD* process. In other words, it can be beneficial to compute intermediate results of a currently unknown process instance.

The next section focusses on the general idea of pre-computing parts of potential instances of the *KDD* process.

4.2 Splitting the *KDD* process in Parameter-Independent and Parameter-Dependent Parts

As mentioned in the last section, one should bring tasks forward that can be brought forward to save scanning the data multiple times. For the same reason, one should also bring forward the computation of all items such as intermediate results of potential future analyses and auxiliary data. Auxiliary data are unessential for computing an analysis but are helpful items that improve either performance or quality of an analysis. Therefore, we define intermediate results and auxiliary data as follows.

Definition 4.1 *An intermediate result is an item if there is at least one type of potential future analysis that needs the presence of that item.*

Definition 4.2 *An auxiliary date is an item if there is at least one type of potential future analysis that benefits of the presence of that item.*

Obviously, an item can be intermediate result as well as auxiliary data—the succeeding analyses determine whether an item is an intermediate result or an auxiliary date.

Yet, discrimination makes sense because intermediate results and auxiliary data have different effects on the *KDD* process. While pre-computing intermediate results saves time in a succeeding instance of the specific *KDD* process, computing auxiliary data can improve the quality of a succeeding instance of the specific *KDD* process.

Therefore, Subsection 4.2.1 discusses the requirements for splitting the *KDD* process in two distinct processes to save time. Analogously, Subsection 4.2.2 discusses analyses which produce results that might be beneficial for succeeding analyses in an instance of the specific *KDD* process. In other words, it examines analyses that compute auxiliary data.

4.2.1 Splitting Saves Time

If items that can be brought forward are independent of any specific parameter of an instance of the *KDD* process, it is possible to source out these items in a separate process. The separate process processes all parameter-independent items. A second process processes the remaining parameter-dependent items.

Splitting the *KDD* process saves time because multiple instances of the parameter-dependent process share the same instance of the parameter-independent process.

It is common practise to use data in a data warehouse for analysis because some pre-processing tasks such as data cleaning are already done when using data of data warehouses. The intention of using data warehouses is the same as splitting the *KDD* process in parameter-independent and parameter-dependent parts. Maintaining the data warehouse is a parameter-independent task while

data mining is a parameter-dependent task. However, splitting into parameter-independent and parameter-dependent parts considers more tasks than preparing data in a separate process, only.

In particular, splitting the *KDD* process considers the tasks of the *KDD* process as sequences of operations on data. An operation is a very low-level item of computation such as performing a specific select-statement on the data.

By splitting tasks in operations, there are more potential operations to source out than viewing tasks as atomic items.

Operations must be parameter-independent to source them out. For instance, the sequential computations of a data mining algorithm are a sequence of operations on data. Several operations of a mining algorithm are parameter-independent. For instance, any run of the Apriori algorithm must count the frequencies of attribute values in a table. Counting the frequency of attribute values is an operation that can be part of the parameter-independent process. Section 6.2 discusses how to pre-count the frequency of attribute values, while Section 7.5.2 discusses how classification analyses can benefit of pre-counted frequencies.

Figure 4.1 shows the phases of the *KDD* process split into operations. As the operations of the data mining phase depend on the type of analysis the analysts is intending to do, there are various operations of the data mining phase. Hence, Figure 4.1 only shows the operations of association rule analysis and naive Bayes classification as examples of operations of data mining techniques.

As an operation is only a part of a task, the result of an operation is only an intermediate result of the result of that task.

As the experiments of this dissertation show, the amount of tasks that can be pre-computed increases by splitting them in parameter-dependent and parameter-independent parts.

4.2.2 Additional Anticipatory Analyses Improve Quality

Prior knowledge that is gained in a previous analysis can improve the quality of the results of another analysis.

Both, intermediate results and auxiliary data, are either tuples fulfilling some special condition or some statistics of the data. For instance, tuples that are modus, medoid, or outliers of the data set are tuples fulfilling a special condition—modus and medoid represent typical items of the data set, while outliers represent atypical items. Furthermore, all kind of statistics such as mean and deviation parameter are potential characteristics of the data. Therefore, we define auxiliary statistics and auxiliary tuples as follows.

Definition 4.3 *An auxiliary statistic is a statistic of a data set where there is at least one type of potential analysis of that data set that might use this statistic as auxiliary date.*

Definition 4.4 *An auxiliary tuple is a tuple of a data set where there is at least one type of potential analysis of that data set that might use this tuple as auxiliary date.*

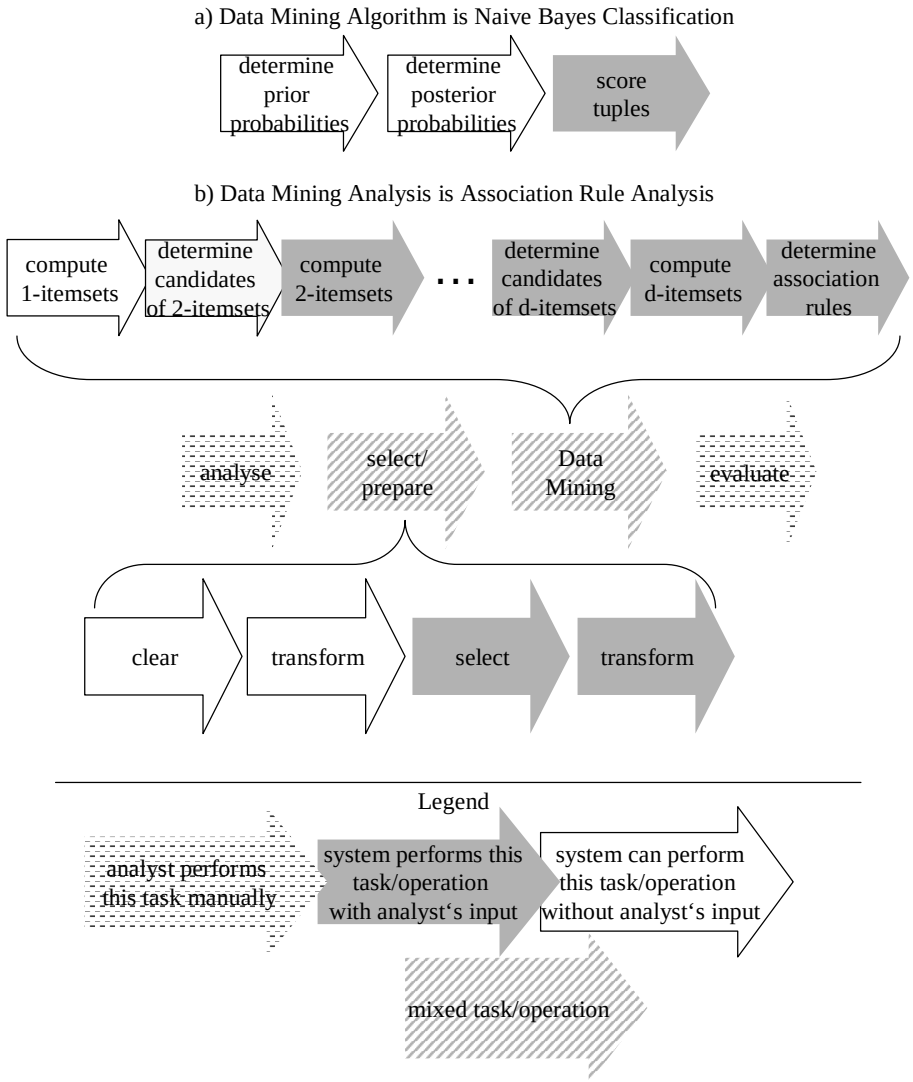


Figure 4.1: Splitting the phases of the*KDD* process into a set of operations illustrated by Naive Bayes and Apriori

Definition 4.4 implies that there must be at least one type of analysis where choosing a set of pre-selected tuples is more beneficial than selecting tuples at random—i.e., the expectation of using pre-selected tuples must exceed the expectation of using a random sample of tuples.

Statistics that indicate the correlation of attributes or the distribution of attribute values are potentially beneficial for instances of the specific *KDD* process. For instance, succeeding classification algorithms can bias classifiers to better fit the prior probabilities. Additionally, *EM*-clustering algorithm can improve the quality of its clusters when the covariance matrix is known a-priori.

Tuples also differ in their relevance for future analyses. The presence of outliers in the data set can cause classification algorithms and clustering algorithms to produce results with minor quality. Classification algorithms are prone to over-fit if there are outliers in the training set. Analogously, outliers influence tend to influence the result of a clustering algorithm that uses centroids above average. Hence, tuples that are non-outliers are better candidates than outliers for being part of a training set. Additionally, using only non-outliers for clustering with centroids tends to deliver better results than using outliers and non-outliers. Contrary, the analyst might be interested in outliers because they represent atypical behaviour. For instance, fraud detection is an application where one is more interested in patterns of atypical behaviour than patterns of typical behaviour.

Therefore, selecting a set of tuples that represent typical or atypical tuples in a data set is a good choice to improve the result of analyses.

Yet again, determining auxiliary tuples as well as auxiliary statistics must be efficient such that it is possible to pre-compute them in an anteceding pre-processing or pre-mining step, as is discussed in the following.

4.3 General Pre-Process

The *general pre-process* includes all tasks and operations of the original *KDD* process that are independent of any parameter.

Additionally, the *general pre-process* includes all tasks and operations of the original *KDD* process that have parameters which have only a small number of distinct values. Due to the small number of distinct values of a parameter, it is possible to compute the result for each distinct value of a parameter.

As the pre-processing phase and the data mining phase of the original *KDD* process contain parameter-independent tasks or operations, respectively, the general pre-process consists of the two phases *pre-processing* and *pre-mining*.

The *pre-processing* phase of the general pre-process includes all tasks of the pre-processing phase of the traditional *KDD* process that are parameter-independent. Hence, its tasks are a subset of the tasks of the traditional pre-processing phase.

However, the *pre-mining* phase includes runs of data mining algorithms that are made in anticipation of future instances of the specific *KDD* process that will make use of the results of these runs.

Thus, the remainder of this section is organised as follows: Subsection 4.3.1 discusses the parameter-independency of the tasks of the pre-processing phase which is required to become part of the pre-processing phase of the general pre-process. Subsection 4.3.2 first shows how to efficiently pre-compute intermediate results and auxiliary statistics, as first introduced in section 4.2. Second, it describes efficient ways to select auxiliary tuples.

4.3.1 Pre-Processing Phase

The pre-processing phase of the general pre-process includes those tasks of the pre-processing phase of the traditional *KDD* process which are independent of any parameter. Data cleaning and data integration are parameter-independent tasks. Therefore, they are also tasks of the pre-processing phase of the general pre-process.

Data cleaning and data integration happens in the same way in the general pre-process as it happens in the traditional *KDD* process. Hence, this section only discusses issues of pre-processing that need some special treatment or issues of pre-processing that arise when combining pre-processing with pre-mining. For details concerning data cleaning and data integration for *KDD* the interested reader is referred to Pyle [61, chapter 8].

Cleaning data consists of subtasks such as correcting erroneous tuples and inserting missing values. Many data mining algorithms such as clustering algorithms are unable to handle tuples having missing values. Hence, they have to be inserted before mining them. Yet, a missing value can occur due to several reasons. On the one hand, a correct value can exist but someone has failed to insert this value into the database. For instance, a customer might want to be as private as possible and omits to tell one's gender. Although there is a correct value for attribute *gender* of this customer but the company does not know it. On the other hand, there are tuples with missing values where the missing value is the correct value of an attribute. Such a phenomenon can occur when there is an optional foreign key constraint in the schema of a relation. If an attribute references to the key attribute of another relation then a missing value in a tuple of the referencing table denotes that there is no corresponding tuple in the referenced table. Assume that the company of the running example stores customers that attracted new customers. It realises this by adding the identifier of the attracting customer to the attracted customers. Customers having a missing value in that attribute came in touch with the company without being attracted by other customers.

If it is impossible or too expensive to determine the correct value of a missing value one can estimate the missing value using some kind of regression method such as linear regression, logistic regression, or auto-regression. Regression is a good option to fill missing values because it does not change the expectation value of the attribute which has missing values. Otherwise, a change in expectation value can cause a severe bias in succeeding runs of data mining algorithms.

Integrating data from different sources is useful to avoid network traffic when

accessing data for analyses. Additionally, integrating data avoids several network related problems such as temporarily inaccessible data sources. If a data source is temporally unavailable, one can integrate it when it becomes available again. As integration of data sources usually happens when there are no users working with the data, no user has to wait for temporally unavailable data sources. Contrary, if an unavailable data source which is not integrated is needed for an analysis, the analyst has to wait until it becomes available. Once a data source is integrated, it makes no difference whether the data source is available or not.

When integrating data from different sources one has the options to either pre-process the data locally on each machine that hosts a data source or pre-process the data globally on the machine that hosts the integrated data source. This option is not limited to pre-processing tasks. Especially, this is also an option for pre-mining tasks, i.e. one has also the option to choose to pre-mine data locally or globally. However, the pre-mining method which we will discuss later depends on the sequence of accessing tuples. Hence, the results of local and global pre-mining might differ.

Stefan Schaubschläger has shown in his master's thesis [63] that integrating data and pre-mining data is almost commutative for large data sets. In a test series he locally pre-mined tuples on a set of clients and integrated them on a single server. The result of pre-mining was a tree of summarising statistics. He compared this result with the result of another test series where he integrated data first on the server before pre-mining them on the server. The results were very similar but local pre-mining was significantly faster because several machines simultaneously pre-mine the data. To be more specific, only the summarising statistics at lower level of the tree of statistics differed from each other. Hence, local pre-mining is a good option to increase the performance of pre-mining.

4.3.2 Pre-Mining Phase

In order to split operations of data mining techniques into dependent and independent operations, we analyse the results of the operations of the data mining techniques we have discussed in chapter 2. If an intermediate result does not depend on any parameter then we can pre-compute it in the pre-mining phase. If an intermediate result depends on a parameter which has only a few distinct values we can pre-compute all potential intermediate results in the pre-mining phase.

This section discusses the ability of intermediate results of data mining algorithms to be pre-computed. Yet for some of these intermediate results, pre-computing is only possible if special constraints are given. As the focus of this section is only on the general ability of pre-computing intermediate results, we postpone the description of how to handle specific constraints to later sections which describe approaches to pre-compute intermediate results in full. Chapter 5 presents a novel approach for clustering that is able to use a tree of summarising statistics that once has been created and regularly kept up-to-date for

type of algorithm	(intermediate) result	depends on	pre-computable?
naïve Bayes classifier	frequencies of pairs of attribute values when attribute have few distinct values	chosen attributes	yes
	parameters of joint probability density function of pairs of attributes when attributes have many distinct values	chosen attributes	yes
decision tree classifier	split points of attributes	chosen attributes	yes
	decision tree	training set	no
Apriori	1-itemsets	chosen attributes	yes
	d-itemsets	minimum support	no
FP-growth	ordering of items	chosen attributes	yes
	FP-tree	minimum support	no

Table 4.1: Overview of intermediate results

multiple cluster analyses, while chapter 6 presents novel approaches of other data mining techniques, namely pre-computing 1-itemsets for Apriori and FP-growth, and using auxiliary data to increase the quality of decision tree building classification algorithms and naïve Bayes classification algorithms.

The section concludes with a discussion how to efficiently compute intermediate results and auxiliary data.

The order of this section follows the order in which section 2 has presented algorithms of different data mining techniques, i.e. clustering algorithms, classification algorithms, and algorithms for association rule analysis. Table 4.1 presents an overview of intermediate results for various types of data mining algorithms. Yet, clustering is missing in this table. Clustering will be discussed extensively in chapter 5.

Intermediate Results and Auxiliary Data for Clustering Algorithms

Bradley et al. have observed that tuples which are very similar are only rarely part of different clusters [59]. Hence, many approaches of related work we discussed in section 3.5 consider sets of very similar tuples instead of considering tuples individually to increase the performance of clustering algorithms.

When clustering a data set with a hierarchical clustering algorithm one receives a dendrogram in which the leaf nodes represent small groups of tuples which are very similar to each other. If one uses this sub-clusters to replace the original data set in another clustering algorithm, then these sub-clusters are intermediate results of the succeeding clustering algorithm.

The initial clustering of tuples is a pre-clustering of tuples that generates a set of intermediate results—which are sub-cluster in this specific case. Hence, initial clustering of data is a task of the pre-mining phase.

Cluster analyses might differ in the parameters of the used clustering algorithm and the analysed data set.

While existing work can pre-compute intermediate results for cluster analyses with different values of parameters, pre-computing intermediate results for cluster analyses that need different subsets of a data set is still a challenge. Chapter 5 presents a novel method that is capable to use a set of sub-clusters for analyses where the analysed data might be an arbitrary subset of a fact table that has been pre-clustered.

Intermediate Results and Auxiliary Data for Classification Algorithms

As mentioned in Section 2.3.2, the construction of a naïve Bayes classifier requires the frequency of each pair of attribute values, where one attribute is the class attribute and the other attribute is an attribute the analyst has selected. Hence, these frequencies are intermediate results for constructing naïve Bayes classifiers. However, the attribute that will function as class attribute in a future classification is typically unknown during the pre-mining phase. The same argumentation holds for those attributes which the analyst selects as relevant attributes to classify tuples.

Yet, the class attribute and all attributes the analyst considers as relevant attributes must be attributes of the table that might be used for analyses in the future. Hence, the schema of that table limits the number of attributes which are potentially relevant. Moreover, the class attribute typically has only few distinct values. Having too many distinct classes makes no sense for most applications. Additionally, the training set needs to be very large if the number of classes is high. Thus, the class attribute is an attribute that has only a few distinct values. Hence, attribute values of the class attribute must be frequent.

Section 2.3.2 has mentioned that the number of distinct values of an attribute must be small to receive highly accurate naïve Bayes classifiers. If not, one could use probability density functions instead of the according frequencies to improve the classifier's accuracy. Thus, we distinguish in our argumentation of the ability to pre-compute intermediate results for naïve Bayes classifiers into analyses using attributes having few distinct values and attributes having many distinct values.

If an attribute has few distinct values then most values of this attribute should be frequent.

When pre-computing the frequencies of pairs of frequent attribute values, the set of so-computed frequencies should also include the frequencies that a potential application of naïve Bayes classification needs as values of class attribute and relevant attributes are typically frequent.

An approach in Section 6.5 demonstrates how to derive a naïve Bayes classifier using only pre-computed frequencies. An experiment to this approach in section 7.5.2 shows that the accuracy one receives using pre-computed frequencies exceeds the accuracy one receives when using the conventional way of computing a naïve Bayes classifier using a training set.

Intermediate Results and Auxiliary Data for Association Rule Analyses

Apriori algorithm iteratively computes candidate itemsets and tests whether they are frequent or not. The 1-itemsets candidates depend on no parameter. Hence, one can determine them in anticipation of an association rule analysis. In analogy to the argumentation when discussing naïve Bayes classification, association rule analyses might differ in the attributes that identify transactions and items, respectively. Here again, the schema limits the number of potentially relevant attributes. Thus, one can pre-compute the frequencies of 1-itemsets for all attributes that might be relevant.

FP-growth algorithm needs the frequencies of items to receive a strict ordering of items in the FP-tree. The frequencies of 1-itemsets and the frequencies of items denote the same objects. Hence, the argumentation of Apriori also applies to FP-growth.

Summarising, many analyses require the frequencies of specific attribute values or combinations of attribute values. Especially, analyses using categorical attributes can benefit of pre-computed frequencies of attribute values and attribute value combinations.

Contrary, algorithms that operate in multi-dimensional vector spaces such as many clustering algorithms can profit of pre-computed statistics of sub-clusters.

Additionally, one can use pre-computed statistics for many different purposes and for many different types of analyses. For instance, clustering algorithms can save time using statistics instead of tuples. One can use pre-computed statistics to derive a probability density function of which a classification algorithm can profit of.

The discussion about the different types of data mining algorithms indicates that many pre-computable intermediate results depend on the attributes an analyst selects in an analysis. As in a fact table the number of attributes is significantly smaller than the number of tuples it is possible to pre-compute intermediate results for all attributes which an analyst might select.

Efficient Computation of Auxiliary Statistics

Many intermediate results of algorithm which the preceding subsections have discussed have the type auxiliary statistic, as defined in definition 4.3. Note that an auxiliary statistic can be an intermediate result as well as an auxiliary date. The type of analysis determines the role of an auxiliary statistic.

This subsection introduces an efficient way to pre-compute auxiliary statistics for being intermediate results or auxiliary data of future analyses.

Table 4.2 depicts a fact table with five attributes. Assume that attributes A, B, and C are numerical attributes. Further assume that attributes D and E are categorical.

As discussed in previous subsections of this section, statistics of interest include mean, deviation, correlation, and range for each attribute. In order to compute these statistics, it is sufficient to store count, linear sum, sum of

squares, and extreme values of each attribute—with the single exception of correlation. We will discuss this issue later when presenting the algorithm CHAD in chapter 5 because there are several alternatives to handle correlation.

CHAD is a clustering algorithm which produces a dendrogram of clusters in its first phase. Each entry of a node of this dendrogram stores count, linear sum, sum of squares, and extreme values for a small subset of tuples. Thus, each entry stores auxiliary statistics of a small and very similar junk of data.

Splitting a fact table into several small parts having very similar data is necessary to select relevant data in the specific *KDD* process. Thereby, a selection method applies a selection predicate to each junk of data to receive the statistics of those data that fulfill the predicate. In most cases, determining those statistics reduces to summing up the according statistics of the junks that fulfill the selection predicate. Section 5.4 presents the selection of statistics in full.

When pre-clustering the fact table the clustering algorithm has to compare distances of different attributes. For this purpose it needs a distance function which makes attributes comparable. The section introducing clustering algorithms has suggested to use normalised attributes when comparing distances of different attributes. Therefore, we normalise all relevant attributes. *Z*-normalisation is a very common technique of normalisation.

Due to its commonness we choose to z-normalise attributes. To do so, we need to know mean and standard deviation of each attribute.

Mean and standard deviation might change when inserting new tuples. Consequentially, the statistics we have determined are no longer z-normalised when this happens.

Yet, a change of the mean of an attribute has no effect on the relative distances between tuples. A change in mean affects only a linear transformation in the vector space which is spanned by the relevant attributes.

However, a change in deviation of a single attribute can affect the similarity of tuples—yet, only decreasing deviation can cause a problem. If the deviation of an attribute decreases then pairs of tuples which have been very similar before the deviation has decreased might become less similar. If a pair of tuples becomes so dissimilar that they cannot be members of the same junk, then we face the problem that we have tuples in the same junk that should be in different junks. This can happen because these tuples have previously been so similar that pre-clustering has inserted them into the same junk. If in contrast to that the deviation of an attribute increases then tuples become more similar. We might face tuples which now can be part of the same junk although they previously were too dissimilar. As it is possible to merge junks, increasing deviation is not a problem.

Fortunately, standard deviation tends to rise slightly with increasing number of tuples. Therefore, the above mentioned problem of decreasing deviation is a rarely occurring problem.

Frequencies of attribute values of categorical attributes are auxiliary statistics needed for several types of analyses such as classification and association rule analysis. Counting the frequencies of attribute values needs a single scan of the data. If counting of frequencies happens in combination with pre-clustering

	attributes				
node id	A	B	C	D	E
1	a_1	b_1	c_1	d_1	e_1
	a_2	b_2	c_2	d_2	e_2
	a_3	b_3	c_3	d_3	e_3
	a_4	b_4	c_4	d_4	e_4
	a_5	b_5	c_5	d_5	e_5
	a_6	b_6	c_6	d_6	e_6
2	a_7	b_7	c_7	d_7	e_7
	a_8	b_8	c_8	d_8	e_8
	a_9	b_9	c_9	d_9	e_9
	a_{10}	b_{10}	c_{10}	d_{10}	e_{10}
3	a_{11}	b_{11}	c_{11}	d_{11}	e_{11}
	a_{12}	b_{12}	c_{12}	d_{12}	e_{12}
	a_{13}	b_{13}	c_{13}	d_{13}	e_{13}
	a_{14}	b_{14}	c_{14}	d_{14}	e_{14}
	a_{15}	b_{15}	c_{15}	d_{15}	e_{15}
	a_{16}	b_{16}	c_{16}	d_{16}	e_{16}
	a_{17}	b_{17}	c_{17}	d_{17}	e_{17}
4	a_{18}	b_{18}	c_{18}	d_{18}	e_{18}
	a_{19}	b_{19}	c_{19}	d_{19}	e_{19}

Pre-clustering splits tuples into several small subsets in which tuples are most similar to each other

pre-compute statistics per subset for relevant non-categorical attributes

determine frequencies for relevant categorical attributes

Table 4.2: Pre-mining a fact table

of data, no additional scan is needed. Section 6.5 demonstrates an approach that stores the frequencies of distinct values of categorical attributes into a buffer with fixed size in anticipation of future classifications. Yet, one can use this approach to pre-compute frequencies needed for association rule analyses. If the buffer is too small to store the frequencies of all distinct values, the algorithm used in this approach removes infrequent attribute values for having enough space to store the most frequent attribute values. This simplification is tolerable because classifications and association rules analyses work only well with frequent values. The experiments of this dissertation have shown that the accuracy of classifiers using pre-counted frequencies is very high.

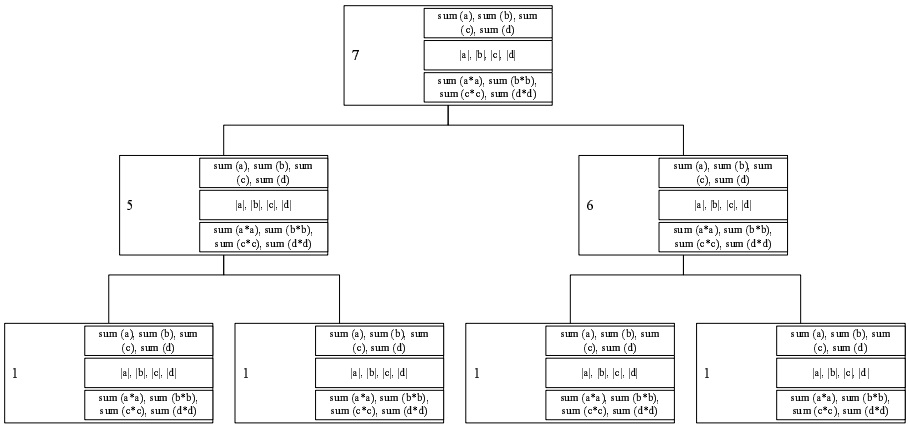


Figure 4.2: Pre-clustering of data returns a dendrogram of sub-clusters with auxiliary statistics

Efficient Selection of Auxiliary Tuples

Due to the number of potential auxiliary tuples storing all potential tuples is inappropriate. Hence, storing a representative selection of potential auxiliary tuple solves this problem.

When the data set is partitioned in areas of different density, it is inefficient to store the same fraction of potential auxiliary tuples for each area. Very dense areas are populated with tuples that are very similar to each other. Thus, a smaller portion of tuples is needed to represent the variety of tuples in these areas. Contrary, less dense areas need more auxiliary tuples to represent the tuples of these areas.

The probability density function over the tuples of the data set is able to keep the information about dense and sparse areas. That kind of information is necessary to choose auxiliary tuples of dense areas more likely in instances of the specific *KDD* process. Otherwise, the so-chosen sample would no longer be representative.

As the probability density function might be an arbitrary function in general, it has to be approximated. As any probability function can be approximated by a set of Gaussian distributions [64, Section 3.2 kernel estimators], one can partition the data set with a partitioning clustering algorithm to receive a Gaussian distribution for each cluster[1]. The less the average distances within a cluster the better is the approximation with the probability density function. EM clustering algorithm [12] is an algorithm that one can use to searches for the optimal Gaussian mixture model [66][65].

Figure 4.3 illustrates the concept of auxiliary tuples. It visualises two dimensions of an excerpt of one of the data sets used for the experiments in chapter 7.

[1]In Subsection 5.4.4 we will discuss assumptions and statistical approximations and their justification in detail.

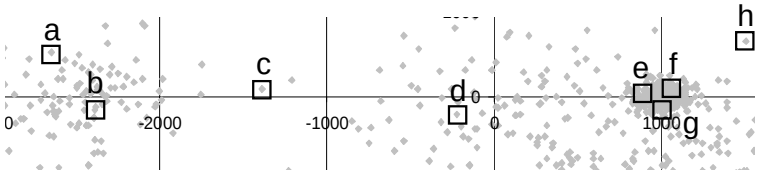


Figure 4.3: Auxiliary tuples a-h in an excerpt of the data set of Figure A.7

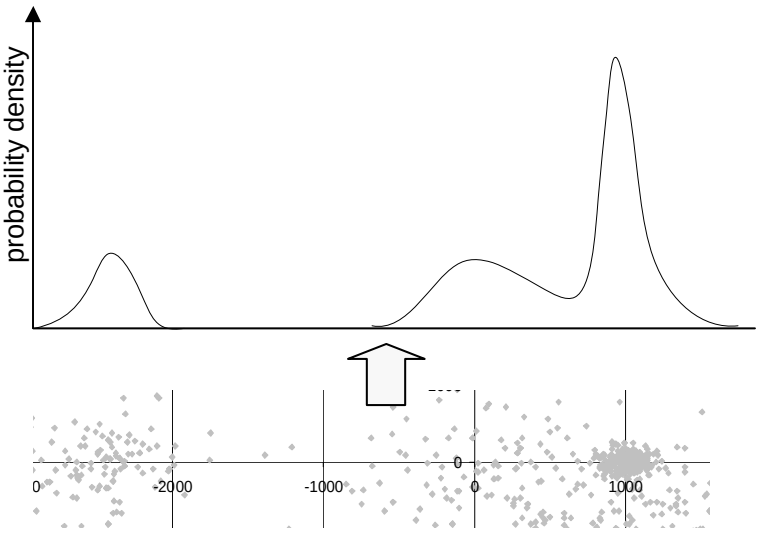


Figure 4.4: Probability density of one dimension of of figure 4.3

The boxes marked with a to h emphasise the tuples in their centres. We want to use these tuples to discuss auxiliary tuples.

Tuples b, d, and the cluster of tuples e, f, and g are located in dense regions of the data set. In contrast to tuple a these tuples are in the centre of a dense region. Thus, they are typical representatives of that data set. As the tests in chapter 7 show selecting these kind of tuples as training set is beneficial for classification. Thus, they are auxiliary tuples.

The dense area around tuples e, f, and g contains many tuples that might be auxiliary tuples. Yet, selecting all of them into the set of auxiliary tuples would mean to select the majority of auxiliary tuples that are all very similar.

Tuples c and h are located in sparse areas of the data set. Thus, they are potential outliers.

Special Handling for the Specific *KDD* Process

In order to use auxiliary tuples or auxiliary statistics in analyses of future instances of the specific *KDD* process, these statistics and tuples must either be

applicable in any analysis or there must exist an appropriate statistic or tuple for each potential analysis. In other words, one either computes a statistic that can be used in all potential analyses or one computes all values a statistic might assume for all analyses.

Potential analyses which use the same data set might differ in the specific subset of the data set the analyst is interested in. Moreover, the analyst can perform operations on the data which transform it such as aggregating tuples.

Aggregating data is a potential operation of an analyst on a data set which needs special handling during the general pre-process. For instance, an analyst of the company of the running example might analyse purchases per customer. For doing so, he/she selects data from the *sales* table with an SQL-statement and groups tuples according the identifier of customers.

Yet, if the attribute which the analyst uses for grouping has many distinct values the re-computation of statistics to fit the group-by-condition is error-prone.

As the number of attributes which the analyst might use to group tuples is limited, the option of computing statistics for all potential attributes the analyst might choose for grouping remains.

4.3.3 General Pre-Process of the Running Example

The section describing the general pre-process concludes with this subsection which demonstrates how to implement the concepts presented so far in the running example.

For this purpose, Table 4.3 depicts an excerpt of the fact table *sales*. It shows the data of six customers which have ordered literature. In this excerpt only customer 4711 has made more than one purchase.

The company wants to pre-process the fact table *sales* to improve speed and quality of *KDD* analyses in the future. Thereby, it pre-computes auxiliary statistics and pre-selects auxiliary tuples. Therefore, the succeeding subsections discuss pre-computing auxiliary statistic and pre-selecting auxiliary tuples, respectively.

Pre-Computing Auxiliary Statistics

In contrast to auxiliary tuples, not all attributes are relevant for being pre-computed in form of auxiliary statistics. Therefore, we want to discuss how suitable specific attributes of the fact table are for being part of an analysis to pre-compute statistics which base on those attributes.

Attribute *units* denotes how many units of a specific product a customer has bought in a specific sales transaction. The value of this attribute is needed to determine many properties of sales transactions such as the total price of a market basket, or the number of goods purchased in a transaction. Thus, it is important to store statistics of this attribute for various future analyses including cluster analyses and classifications.

Attribute *priceperunit* is needed in combination with attribute *units* for some analyses which examine revenue of sales transactions. Hence, pre-computing statistics of this attribute is necessary, too.

As attribute *units* is a product-specific attribute, we must not add units of different products if we want to be able to analyse purchasing behaviour concerning specific products or correlation of sales of products. Moreover, we have to consider the sold units of each product individually, i.e. as a separate attribute. The same argumentation holds for attribute *priceperunit*. If there are too many distinct products one can limit the number of attributes by using attributes for units and price per unit for each product group, only.

Attribute *timestamp* is needed for analyses which examine temporal changes of purchasing behaviour of customers. In special, this attribute is needed when an analyst wants to select data of a specific time frame. Hence, pre-computing this attribute is necessary to be able to select other attributes appropriately.

Attribute *prid* references to the product a customer has purchased in a specific sales transaction. The product identifying attribute is the most important attribute when performing a market basket analysis. Due to its many distinct values, attribute *prid* is no suitable attribute for other types of analyses beside market basket analysis. Too many distinct values make classification of categorical prone to over-fitting. Too many distinct products are also a problem for cluster analyses because there needs to be a pre-defined distance between each pair of products. Product groups are much better suited to be used in cluster analyses or classifications because their number of distinct values is much smaller.

Hence, we pre-compute the frequencies of all distinct values of attribute *prid* to speed up the computation of association rule analyses.

In order to improve other types of analyses in which the analyst intends to examine products we add the product group to the attributes of the fact table and pre-compute auxiliary statistics for this newly-introduced attribute. As a product group can have a super-ordinate product group there can exist several product groups to a given product. For being deterministic we assign the product group to a tuple which has the lowest level in the taxonomy of product groups as depicted in figure 4.5. If, for instance, product 8874 is a belletristic novel then it is also a book and a product. Yet, *belletristic* is the most specific product group that applies to product 8874.

The customer identifier (*cid*) is important to group sales transactions by customer. Yet, it is uninteresting as attribute of analyses because typically there are not enough data of a single customer for data mining. Hence, we omit computing frequencies and statistics of this attribute.

For pre-clustering the fact table we need to z-normalise all relevant attributes. Z-normalisation requires numerical scale of attributes. Yet, only attributes *units* and *priceperunit* are numerical attributes. The product identifier *prid*, for instance, is a categorical attribute although all its values are numbers. Yet, we can erroneously consider categorical attributes with numbers as numerical attributes and pre-cluster them. If we do so, we have to keep in mind that all queries using such an attribute select only a single value of this attribute,

units	priceperunit	timestamp	prid	shopid	cid
⋮	⋮	⋮	⋮	⋮	⋮
1	29.95	2004-12-06 14:30:10	6194	1	4711
1	15.95	2004-12-06 14:30:10	8874	1	4711
1	14.55	2004-12-06 14:30:10	1460	1	4711
1	49.95	2005-02-11 09:15:26	8874	1	4711
1	61.80	2005-02-11 09:15:26	9997	1	4711
1	49.95	2005-02-11 10:30:00	3526	1	0815
1	49.95	2005-02-11 11:15:34	2722	1	0515
1	37.50	2005-02-11 11:57:12	8945	1	1704
1	15.85	2005-02-11 11:57:12	820	1	1704
1	49.95	2005-02-11 13:30:25	9585	1	7007
1	69.95	2005-02-11 13:30:35	8649	1	1188
1	89.95	2005-02-11 15:30:45	2412	1	2468
1	59.95	2005-02-11 15:30:45	506	1	2468
4	49.95	2005-02-11 15:30:45	1975	1	2468
3	49.95	2005-02-11 15:30:45	2606	1	2468
1	69.95	2005-02-11 15:30:45	1980	1	2468
1	88.95	2005-02-11 15:30:45	1307	1	2468
1	49.95	2005-02-11 15:30:45	1909	1	2468
⋮	⋮	⋮	⋮	⋮	⋮

Table 4.3: Excerpt of tuples of fact table *sales*

i.e. only operator "=" is valid. Range queries require ordinal attributes. That way, we pre-process the attributes *prid* and *shopid*.

We can transform attribute *timestamp* to be a numerical attribute when we convert it into the time which has elapsed since a given start date. When, later on, tuples of sales are created, these tuples will have a steadily increasing value of the transformed attribute *timestamp*. Hence, standard deviation of this attribute continuously increases. We suggest to determine the standard deviation for an initial data set and use this deviation for a longer period of time. To face increasing deviation we suggest to update standard deviation in regular intervals. This process includes re-computing auxiliary statistics and determining the new value of standard deviation. Within an interval we omit any update of standard deviation.

For being able to handle selection predicates that contain a *group-by* clause, we need to compute each statistic for all potential *group-by* clauses having many distinct values. Contrary, if the the analyst wants to aggregate by an attribute which has only few distinct values, one can realise this grouping by selecting each distinct value. Consider attribute *shopid* which has few distinct values compared to the total number of sales. We can compute the total sales of each shop. All we must do is to perform several selection operations—one for each shop. The same proceeding is possible for attribute *productgroup* we additionally introduced. Therefore, the remaining attributes the analyst can use for grouping are attributes *timestamp* and *cid*. Or in other words, an analyst can analyse customers, transactions, or parts of a sales transaction. If an analyst uses no *group-by* clause, he/she analyses parts of a transaction. If he/she uses a *group-by* clause including attributes *timestamp* and *cid*, he/she analyses transactions. Finally, if he/she uses a *group-by* clause including only attribute *cid* then he/she analyses the total sales of a customer. As there are only three *group-by* clauses with large sets of distinct values we pre-compute the statistics for each of these settings.

Tables 4.4, 4.5, and 4.6 depict the fact table after it has been pre-processed for pre-clustering for the three different settings mentioned above. All attributes of these tables are z-normalised. For instance, value -1 of *sum sales online journal* corresponds with the un-normalised value 0. Hence, the mean must be positive. As each attribute might have different deviation and mean, the same un-normalised value can correspond with a different value of a normalised attribute, i.e. a different value per differently normalised attribute. Hence, the value 0 has three different normalised values in table 4.5, namely -0.9, -1, and -1.1. As the tables depict only an excerpt of the fact table, we are unable to compute the real values of mean and deviation. Thus, the tables contain random but realistic values. Primary keys have been removed because they are irrelevant attributes for data mining. To be able to find the corresponding tuples in the unprocessed fact table, the right-most column contains the missing primary key. Finally, we receive three dendrograms which we can use in the specific *KDD* process.

Tables 4.5 and 4.6 have more attributes than the original fact table. To be more specific, the newly introduced attributes are horizontal aggregates of the

units	priceperunit	timestamp	product group	shopid	customer
⋮	⋮	⋮	⋮	⋮	
-0.01	-0.1	-1.5	-0.5	0	4711
-0.01	-1.15	-1.5	-0.5	0	
-0.01	-1.25	-1.5	-0.5	0	
-0.01	0.2	1.703	-0.5	0	
-0.01	0.75	1.703	-0.5	0	
-0.01	0.2	1.703	-0.5	0	0815
-0.01	0.2	1.703	-0.5	0	0515
-0.01	0.05	1.704	-0.5	0	1704
-0.01	-1.2	1.704	-0.5	0	
-0.01	0.2	1.705	-0.5	0	7007
-0.01	0.8	1.705	-0.5	0	1188
-0.01	1.0	1.708	1.5	0	2468
-0.01	0.6	1.708	1	0	
0.9	0.2	1.708	1	0	
0.7	0.2	1.708	1.2	0	
-0.01	0.8	1.708	1.2	0	
-0.01	0.95	1.708	1.5	0	
-0.01	0.2	1.708	2	0	
⋮	⋮	⋮	⋮	⋮	

Table 4.4: Excerpt of tuples of pre-processed fact table *sales*

attribute *units* and *product group*. Using horizontal aggregation is very common when preparing a data set for *KDD* [57]. Here, the tables show only the top-most level of product groups due to restricted space. Yet, typically one would use the bottom level of the taxonomy of product groups to define an attribute per product group. By doing so, one receive a data set which has some dozens of attributes—which is still a small number of attributes.

Some analyses need auxiliary tuples. Hence, the following subsection presents the pre-selection of auxiliary tuples for the analyses mentioned when we introduced the running example.

Pre-Selecting Auxiliary Tuples

Auxiliary tuples represent a set of tuples sharing a specific condition such as that they are outliers or typical representatives of a data set. In contrast to auxiliary statistics, all attributes are relevant for auxiliary tuples.

The algorithm CHAD as presented in chapter 5 can determine auxiliary tuples as by-product of its first phase. CHAD can determine auxiliary tuples as follows: First, we define a buffer for typical representatives of the fact table we want to pre-process and a second buffer for atypical representatives—i.e. outliers.

sum units book	sum units article	⋯	sum sales article	sum sales online journal	timestamp	customer
⋮	⋮	⋱	⋮	⋮	⋮	
0.3	-0.9	⋯	-1.1	-1	-1.5	4711
0.25	-0.9	⋯	-1.1	-1	1.703	
0.01	-0.9	⋯	-1.1	-1	1.703	0815
0.01	-0.9	⋯	-1.1	-1	1.703	0515
0.25	-0.9	⋯	-1.1	-1	1.704	1704
0.01	-0.9	⋯	-1.1	-1	1.705	7007
0.01	-0.9	⋯	-1.1	-1	1.705	1188
-2.5	3.1	⋯	3.5	5.3	1.708	2468
⋮	⋮	⋱	⋮	⋮	⋮	

Table 4.5: Excerpt of tuples of pre-processed fact table *sales* grouped by customer and time

sum units book	sum units article	⋯	sum sales article	sum sales online journal	customer
⋮	⋮	⋱	⋮	⋮	⋮
0.4	-0.9	⋯	-1.1	-1	4711
0.02	-0.9	⋯	-1.1	-1	0815
0.02	-0.9	⋯	-1.1	-1	0515
0.28	-0.9	⋯	-1.1	-1	1704
0.02	-0.9	⋯	-1.1	-1	7007
0.02	-0.9	⋯	-1.1	-1	1188
-2.2	3.1	⋯	3.5	5.3	2468
⋮	⋮	⋱	⋮	⋮	⋮

Table 4.6: Excerpt of tuples of pre-processed fact table *sales* grouped by customers

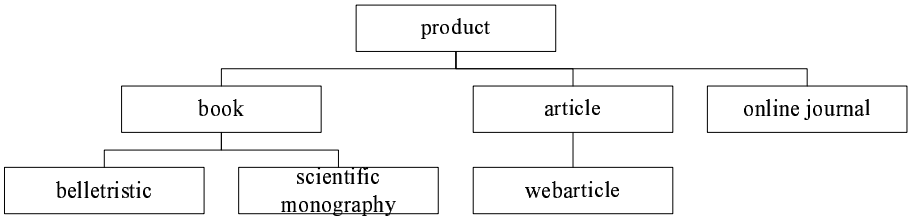


Figure 4.5: Taxonomy of product groups

Each time CHAD creates a new leaf node we temporarily store the first tuple of that leaf node in a temporary table. Depending on the size of the buffer, we also store the next tuples which CHAD tries to insert into the same leaf node until the number of tuples in that node exceeds the average number of auxiliary tuples per node.

We can compute the number of auxiliary tuples per node as quotient of buffer size and maximum number of leaf nodes. By doing so, we receive a table which contains a set of tuples where each node donates one or up to a fixed number of tuples to this table.

Yet, there is no classification which tuple represents typical or atypical tuples. Therefore, we test how dense tuples are in the neighbourhood of the tuples in the buffer. If they are very dense, we consider tuples as typical representatives. Otherwise, we consider them as outliers.

Some algorithms such as k-means are sensitive to outliers which means that the existence of outliers decreases the quality of results [16, p. 54]. Hence, outlier removal can improve the quality of outlier-sensitive clustering algorithms such k-means. Yet, we store outliers in a separate buffer because an analyst might need them for an analysis such as a fraud detection. If we do so, an analyst might use outliers if needed but the analyst can remove them if that would increase an analysis' quality.

For determining the density of the neighbourhood of a leaf node, we determine the average distance between the centroids of leaf nodes. As CHAD puts the nearest nodes into the same branch of a cluster tree, most of the nodes in the neighbourhood of a leaf node are on the same branch or a very near branch. If the average distance within the neighbourhood of a leaf node is higher than the average distance of all leaf nodes we consider that leaf node as outlier. Consequentially, we move the temporarily stored tuples of this leaf node into the buffer we reserved for outliers. If, later on, additional tuples are inserted into the tree in order to update the tree according to changes of the data set, then tuples that are outliers are inserted into the buffer. If the buffer is full, CHAD randomly removes outliers from the buffer such that each outlier has the same chance of being in the buffer. Therefore, the buffer stores a random sample of outliers.

With auxiliary statistics being continuously pre-computed and auxiliary tuples being pre-selected, we can use both of them to compute the results of

analyses in an instance of the specific *KDD* process, as demonstrated in the following section.

4.4 Specific *KDD* Process

The specific *KDD* process is very similar to the original *KDD* process. To be more specific, both processes share the same result and their phases are identical—although the number of tasks in the specific *KDD* process is reduced compared to the traditional *KDD* process.

However, the specific *KDD* process primarily uses intermediate results and auxiliary data of the general pre-process to compute the results of *KDD* analyses.

When changing from the traditional *KDD* process to the specific *KDD* process, the main step to do is to adapt operations and algorithm in a way such that they are able to process intermediate results and auxiliary data.

Therefore, this section discusses necessary modifications of data mining algorithms and operations such as selection and transformation of data. It first presents necessary changes of the pre-processing phase before it presents necessary adaptations of data mining algorithms. Note that evaluating results is out of the scope of this section because there are no changes necessary in that phase.

4.4.1 Specific Pre-processing

Pre-processing in the traditional *KDD* process includes all tasks that process data before applying data mining algorithms on them. These tasks include cleaning, integrating, selecting, and transforming data.

Some pre-processing tasks of the traditional *KDD* process such as cleaning and integrating are independent of any parameter. Hence, the general pre-process can completely process them. Consequentially, there is no longer a need to clean or integrate data during specific pre-processing.

Yet, other pre-processing tasks such as selecting and transforming data depend on parameters. Yet, the general pre-process could pre-compute intermediate results and auxiliary data which both are independent of any parameter.

Data mining algorithms of the specific *KDD* process require correctly selected and transformed intermediate results in the same way as data mining algorithms of the traditional *KDD* process need correctly selected and transformed tuples. Thus, the specific pre-processing phase of the specific *KDD* process includes the tasks *selecting* and *transforming*.

This section first introduces the general principle of selecting data using intermediate results and auxiliary data. In analogy to that, it 4.4.1 discusses the general principle of transforming data to fit the requirements of a specific *KDD* analysis.

Selecting Data Using Pre-Processed Intermediate Results and Auxiliary Data

Selecting data is necessary in many instances of the *KDD* process because only a subset of available data is relevant in the scope of a given analysis. A data warehouse contains many tables including few but large fact tables and a lot of dimension tables. An analyst might choose any subset of any set of joint tables. Only the expressiveness of the used query language and the analyst's creativity might limit the analyst in specifying queries.

The pre-mining phase has already processed intermediate results for all kind of potential analyses. To be more specific, the pre-mining phase computed these intermediate results using all data from a fact table. Yet, if only a fraction of that table is relevant in an analysis, these intermediate results are inadequate for that analysis.

As the pre-computed intermediate results are inappropriate to be used in a specific analysis except that this analysis needs all data from a fact table, there is a need for a method to process intermediate results in a way such that one receives those intermediate results one would have received when one had selected data first and would have computed intermediate results using the so-selected data later. In other words, selecting data first and computing intermediate results on the one hand and computing intermediate results on the other hand should be commutative.

However, in general selecting data and computing intermediate results are not commutative. Yet, we can define a selection method which produces intermediate results that are very similar to those intermediate results one would receive when computing intermediate results using previously selected data. There is only one requirement the data set must meet: attributes have to be non-categorical. Yet, the clustering algorithms that use intermediate results are also limited to non-categorical attributes. Hence, using non-categorical attributes is no additional constraint. We will see later, that the more tuples fulfill a selection predicate, the more similar are selected pre-computed intermediate results and intermediate results of selected tuples. In contrast to that, point queries are subject to low quality of the selection method. Yet, typical queries in the context of data warehouses are queries that select large intervals of data such as all data of sales in a range of quarters. Thus, the requirements for selecting pre-computed intermediate results with high quality are met in the context of data warehouses.

To be more specific, the selection method for intermediate results must face the following sub-problems:

Selection of auxiliary statistics If an intermediate result or an auxiliary date needed for an analysis has the type *auxiliary statistic* then the selection method needs to re-compute that statistic in a way that the resulting statistic approximately assumes the same value as one would receive when one had selected tuples first and had computed the value of the statistic using these tuples. Note that an auxiliary statistic as defined in definition 4.3 can be an intermediate result in one analysis while it can be an

auxiliary date in another analysis.

Selection of auxiliary tuples If an intermediate result or an auxiliary date needed for an analysis has the type *auxiliary tuple* then the selection method can test whether an auxiliary tuple satisfies a selection predicate in the same way as an SQL-processor would do when querying a relational database. Yet, this is only true as long as the selection predicate contains no aggregate functions.

If the selection predicate includes aggregate functions, the result of many common functions can be approximated with some auxiliary statistics, as demonstrated below.

Mixed selection of pre-mined tables and untreated tables The concept *anticipatory data mining* only includes pre-computing intermediate results and auxiliary data for very large tables such as fact tables of a data warehouse because due to the sheer size of these tables scanning them needs much time. In contrast to that, scanning small tables the tuples of which fit entirely into the main memory of the analysing computer system is so fast that a potential improvement due to anticipatory data mining does not justify the overhead needed to pre-compute intermediate results and auxiliary statistics.

If an analysis needs data of more than one table, then it is possible that there exist pre-computed intermediate results of some of these tables while there are other tables where there are no intermediate results available.

Yet, the analyst should not notice which table has pre-processed intermediate results and which not.

Selecting auxiliary statistics can be very complex depending on the selection predicate. A complete algebra that replaces the relational algebra for auxiliary statistics would be necessary for selecting auxiliary statistics using any kind of selection predicate.

Yet due to lack of space and time, we introduce a simplified relational algebra for auxiliary statistics that includes only the constructs which analysts most commonly use.

In a project with NCR Teradata we analysed the behaviour of analysts while they analyse data. We found out that analysts analyse a data set aspect by aspect. In each case the subset an analyst has selected for analysis could have been generated by an OlAP operation. This is interesting as no OLAP system was used to access data. Moreover, analysts were free to specify any query.

Therefore, the author of this dissertation argues to limit the degrees of freedom to specify queries on pre-computed intermediate results and auxiliary data to the following types of conditions:

slice, dice Slice and dice operation limit the range of attribute values fulfilling a selection predicate to a specific subset of the range of an attribute or a set of attributes, respectively. When considering the fact table as a

multidimensional cube, slice and dice cuts sub-cubes out of the original cube. In terms of relational algebra, one can express a slice and dice operation as a conjunction of a set of conditions having the form *attribute operator value* where *operator* might assume one of the values $\{<, \leq, =, \geq, >\}$.

drill-down, roll-up Drill-down and roll-up operations change the level of abstraction of a multi-dimensional cube. Drill-down operation refines the level of abstraction and roll-up coarsens it, respectively. Using SQL, one can realise drill-down and roll-up respectively by putting an attribute which represents the level of abstraction in a specific dimension into the *group by* clause. Subsection 4.3.2 has discussed how to handle *group-by* when the number of distinct values of the grouping attribute is high. If that number is low, one can implement a *group-by* of pre-computed statistics as demonstrated in section 5.4

These types of statements are sufficient to formulate range queries used for analysing data. However, if a query fails to satisfy the above-listed form, one cannot benefit of pre-computed auxiliary statistics. One could simplify a complex statement of a query to receive a query which satisfies the form specified above. If this is impossible, one has to scan the data once more.

The clustering algorithm presented in chapter 5 contains three phases of pre-clustering, selecting and transforming the pre-clustered data, and final clustering. Pre-clustering is a pre-mining technique, selecting and transforming are techniques of the specific pre-processing phase, and the final clustering is a technique of the specific data mining phase, as will be discussed in Section 4.4.2.

The intermediate results of a cluster analysis using intermediate results consist only of auxiliary statistics. Hence, the selection method of the clustering algorithm presented in chapter 5 describes the process of selecting auxiliary statistics in detail. As that section will present selecting auxiliary statistics in full, this section presents only its basic idea.

The basic idea of selecting auxiliary statistics is to break the data set into several small junks in which tuples are very similar to each other. If we have split the data set into several small junks we can test each junk if it fulfills a given selection predicate completely, partially, or not at all. When a junk fulfills a predicate partially, we can estimate the most likely value of the statistic.

The selected statistics can assume wrong values due to bad estimation of those junks that only partially fulfill a selection predicate. Yet, if the number of junks which completely fulfill the selection predicate is large, then the error is minor—even in worst case. Additionally, errors during estimation of one junk and another junk can compensate each other which means that the average error is low.

A short example shall illustrate the idea mentioned above: Assume that we have to compute the average value of a specific attribute of a subset of data. Then, the sum of values of that subset and the number of tuples in that subset are sufficient to compute the average value. By testing each junk of

tuples if it satisfies the selection predicate, we receive those junks that fulfill the selection predicate completely or partially. We add the sums of all junks that completely fulfill the predicate to receive an intermediate sum. For each junk that fulfills the predicate only partially, we estimate the most likely sum and add it to the previously computed intermediate sum. If we proceed the same way to determine the selected count of tuples we can derive the average value as quotient of our estimates for sum and count.

Transforming Data Using Pre-Processed Intermediate Results and Auxiliary Data

Transforming data becomes necessary when a given analysis needs attributes having a specific scale or a cluster analysis requires normalisation of tuples.

Making a continuously scaled attribute discrete is a common transformation of attributes because several algorithms such as decision tree algorithms need discretely scaled attributes. Even those decision tree algorithms that can handle continuously scaled attributes implicitly make attributes discrete when they search for appropriate split points.

Performing a cluster analysis with only a single attribute selected is a proper method to find potential split points of that attribute, as shown by the experiments in section 7.5.

Therefore, clustering is a solution to receive discrete attributes. Hereby, the affiliation to a specific cluster denotes the value of the discrete attribute. Following this approach, one would introduce an attribute denoting the cluster affiliation of tuples. One can determine the affiliation to clusters of tuples. Better, one can determine the affiliation to clusters of sub-clusters, i.e. the junks which the pre-mining phase has found. To do this, one must add an attribute to the sub-clusters. This means to add linear sum, sum of squares, and extreme values of this attribute to the set of pre-computed statistics of each sub-cluster. When assigning a sub-cluster to a cluster, the clustering algorithm assigns all tuples of that sub-cluster to the cluster, i.e. all tuples of a sub-cluster are part of the same cluster. Hence, one can compute the linear sum of the attribute which identifies cluster affiliation as product of the number of tuples in that sub-cluster and a fictive cluster number. Analogously, one can proceed with sum of squares and the extreme values of that attribute—the extreme values are identical with the fictive cluster number.

Several analyses require normalised attributes for comparing attributes—this is especially necessary when the ranges of attributes significantly differ in size. After the selection of pre-processed data the so-selected data are no longer normalised when the selection predicate selects only a subset of the original data set. Only in rare cases mean and deviation of all attributes remain the same. Hence, re-normalisation is necessary to receive normalised intermediate results and auxiliary data for a specific *KDD* analysis such as a cluster analysis.

After re-normalisation the mean of all selected tuples must equal zero; variance and standard deviation must assume the value 1. Re-normalising means to compute mean μ and deviation σ of all selected tuples once more which are

necessary to apply the z-normalisation $(x - \mu)/\sigma$. The z-normalisation is a simple linear transformation which one can apply to all statistics and tuples to receive the according normalised statistics and tuples. The argumentation of section 4.3.2 holds for re-normalising attributes as it does for normalising attributes: Decreasing deviation can negatively affect the result of an analysis while increasing deviation has no negative effect.

Although many cluster analyses need normalised attributes, there also exist analyses that need un-normalised attributes. When analysing geological data, one must not normalise latitude and longitude. Normalising coordinates of countries the extension of which is in one dimension larger than in the other one would mean to penalise the distances in one direction and favouring the other one. Hence, it is necessary to undo the normalisation of attributes if needed. Yet, the process of undoing normalisation is also a linear transformation. It happens as described above after selecting general cluster features and before specific data mining.

Although an analyst typically knows when normalisation makes no sense for a specific attribute, we use the combination of normalisation an un-normalisation to avoid human expert interaction during the pre-general process. In other words, the person that starts the general pre-process needs no knowledge about the pre-processed data.

4.4.2 Specific Data Mining

The specific data mining phase of the specific *KDD* process and the data mining phase of the traditional *KDD* process share the same tasks. In both of them, analysts apply sophisticated (data mining) algorithm on data sets to find patterns which the analysts evaluate for their interestingness. Yet, the algorithms used in the specific data mining phase must be able to handle pre-computed intermediate results. Further, they should be able to make beneficial use of auxiliary data to improve the quality of the patterns they find. Hence, it is necessary to adapt those algorithms which not yet satisfy this requirement.

This subsection sketches the basic principles of approaches that adapt data mining algorithm to be able to use intermediate results and auxiliary data instead of the original tuples which the pre-mining phase has used to pre-compute the intermediate results. The following two chapters of this dissertation, namely Chapter 5 and Chapter 6, will describe those approaches in detail.

As the data mining technique influences the way one has to adapt algorithms, we spread the discussion of necessary adaption over the next subsections—with a subsection for each data mining technique, as discussed in Chapter 2.

Specific Clustering

Many of the approaches we discussed in section 3.5 use summarised statistics to compute the result of a cluster analysis. When re-considering these approaches we observe that the statistics these approaches use are a subset of the auxiliary statistics which phase pre-mining has pre-computed and phase specific

pre-processing has processed, respectively. Hence, anticipatory data mining is an extension of these approaches because it makes those algorithms applicable to a broader set of applications. These approaches are limited to applications that use all tuples of a specific data set. Anticipatory data mining enables these algorithms to process also subsets of that data set. All one has to do is to replace the generation of statistics with the result of the phases pre-mining and specific pre-processing.

The algorithm CHAD which is short for *Clustering Hierarchically Aggregated Data* is an algorithm which implements the concept *anticipatory data mining* for cluster analyses. To be more specific, it contains three phases: The first phase pre-mines tuples. The second phase specifically pre-processes the statistics of the first phase. Finally, the third phase specifically clusters the pre-processed statistics. Currently, the proof-of-concept prototype implements k-means as specific clustering technique. Yet, the concept is not limited to k-means. We have already discussed several approaches that uses clustering features for EM clustering in Section 3.5. As CHAD can create clustering features, one can use the approaches mentioned there to use the result of CHAD's second phase for EM clustering. Additionally, Brecheisen et al. show in [7] how to use data bubbles for density-based clustering. Once more, as one can easily convert a clustering feature into a data bubble, one also can use the result of CHAD's second phase for density-based clustering. Chapter 5 presents the clustering algorithm CHAD in full.

Specific Classifying

Unlike clustering and association rule mining, the data mining technique classification uses only a fraction of a data set to train a classifier. Hence, finding highly accurate classifiers is more important than making classification algorithms scalable.

Carefully selecting the training set can improve the quality of classifiers. The set of auxiliary tuples include tuples which are either typical or atypical representatives of the data set they were selected from.

Experiments show that choosing the training set from auxiliary tuples which are typical representatives causes a higher accuracy than choosing tuples randomly from the data set. See section 7.5 for details of these experiments.

Furthermore, using a selection of pre-selected auxiliary tuples as training set for a classifier is beneficial because there is no need to access the persistent storage to randomly retrieve tuples as auxiliary tuples can be stored in a small cache which is fast to read.

If the used classification algorithm is naïve Bayes then one can use auxiliary statistics to compute a naïve Bayes classifier. Section 6.5 discusses an approach which uses only auxiliary statistics to compute a naïve Bayes classifier.

Specific Association Rule Mining

The pre-mining phase has pre-computed the candidates of 1-itemsets as it pre-counted the frequencies of attribute values and pairs of attribute values. Hence, algorithms mining association rules can start the computation of association rules the step after determining 1-itemsets.

Apriori algorithm first checks the frequency of candidates of 1-itemsets and determines frequent 1-itemsets. After this, it continues to iteratively generate and test itemsets with increasing number of items until there are no additional frequent itemsets. Hence, this involves several additional scans. Yet, pre-computing auxiliary statistics has saved exactly one scan of the fact table.

FP-growth also checks the frequency of the pre-computed 1-itemset candidates first. Then, it orders the frequent itemsets by their frequency. Finally, it builds the FP-tree as described in Section 2.4.4. Using pre-computed auxiliary statistics saves exactly one out of two scans of the fact table. Hence, it saves half of disc accesses.

4.4.3 Specific *KDD* Process of the Running Example

As mentioned in previous subsections of this chapter, the analyst should notice nothing but significant increase in speed or quality when using anticipatory data mining. Especially, he/she should not have to do additional tasks. Hence, this section describes the tasks an analyst of the company of the running example has to do when doing the analyses mentioned in Section 1.2—which are the same when using the traditional method or using anticipatory data mining. Yet, the system that processes the tasks the analyst specifies has to act differently in the background. Hence, we will also discuss how the mining system processes these tasks.

Chapter 5

Anticipatory Clustering Using CHAD

Contents

5.1 Architecture of CHAD

CHAD is short for *Clustering Hierarchically Aggregated Data*. Hence, it is a clustering algorithm that uses aggregated data for clustering.

CHAD implements the concept of anticipatory data mining for clustering, as shown in Figure 5.2. It consists of three phases which correspond with phases of anticipatory data mining: Phase 1 constructs a dendrogram of sub-clusters—each of which is represented by a set of statistics and extreme values. Phase 2 uses the statistics and extreme values of sub-clusters to select and transform the dendrogram according a selection predicate which the analyst has specified. Finally, phase 3 uses the selected dendrogram to cluster the data represented by the dendrogram. Thus, phase 1 of CHAD corresponds with the pre-mining phase of anticipatory data mining. Phase 2 corresponds with specific pre-processing. Last but not least, phase 3 of CHAD corresponds with the specific data mining phase of anticipatory data mining. To be more specific, phase 3 is a specific clustering algorithm.

Obviously, CHAD includes a phase for each phase of anticipatory data mining except the general pre-processing phase. Hence, CHAD requires that data has been previously cleaned and integrated.

Figure 5.1 depicts the general proceeding and the aim of CHAD. The left hand side of the figure shows from top to bottom how to find clusters in a data set in the traditional way: An analyst selects and transforms data before a clustering algorithm finds a set of clusters. Contrary, CHAD pre-clusters data first in a dendrogram which we will introduce as cfg-tree in section 5.2 before an analyst selects and transforms the pre-clustered data, as sketched on the right hand side of the figure.

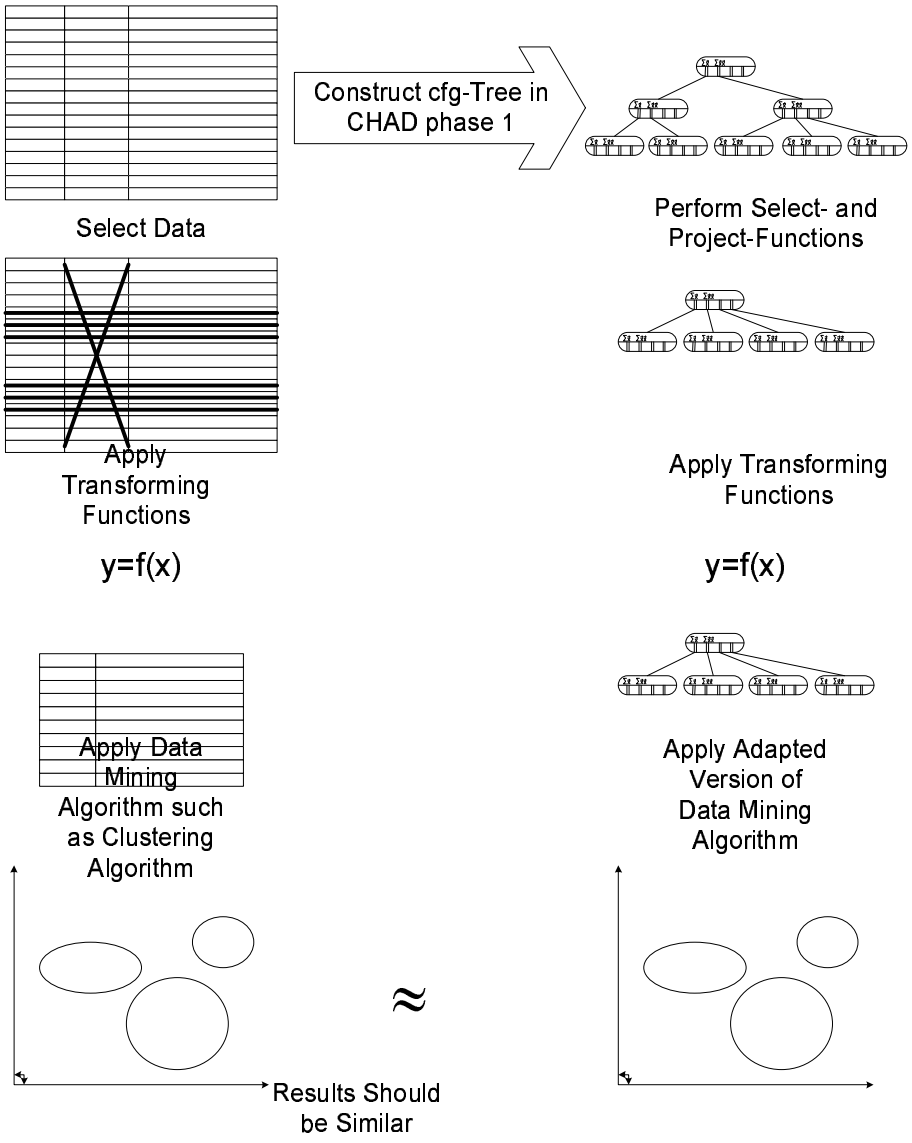


Figure 5.1: CHAD compared with traditional way of analysis

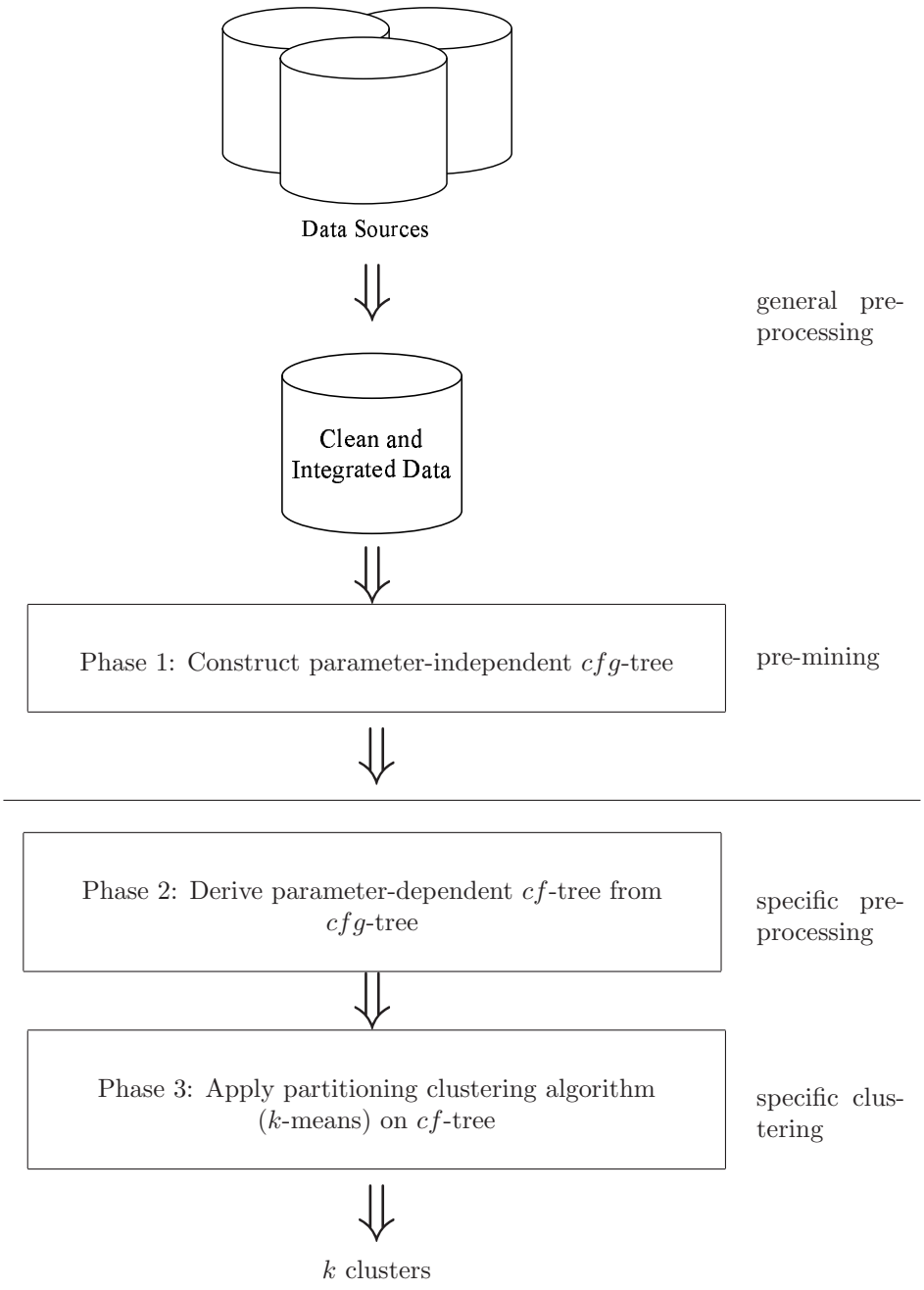


Figure 5.2: Phases of CHAD

The aim of CHAD is to find a similar result using aggregated data as a traditional clustering algorithm would find using non-aggregated data. Yet, using CHAD saves much time because it has only a single time-consuming scan which it can share between many runs. A traditional clustering algorithm needs to re-scan the data set at least once per run. Yet, traditional clustering algorithms requiring only a single scan are among the set of very efficient algorithms.

Due to aggregation and estimates CHAD makes, the quality of CHAD can suffer. Yet, the experiments show that the way a clustering algorithm is initialised influences the resulting quality by far stronger than any other effect can do. Yet, CHAD is much faster than traditional clustering.

Hence, one can invest a fraction of the time saved by CHAD to find a good initialisation and receive a better result in shorter time compared to the original clustering algorithm. In contrast to other generally applicable techniques such as sampling, CHAD is faster and more broadly applicable. Additionally, CHAD can determine an upper bound for the error it makes which is impossible when using sampling.

The first phase of CHAD is very similar to the first phase of BIRCH [81]. Yet, CHAD has some important extensions to make it applicable for anticipatory data mining in general and anticipatory clustering in specific. Hence, section 5.1.1 surveys commons and differences of both algorithms.

The first phase of CHAD scans a fact table once and constructs a dendrogram of sub-clusters. Hereby, it uses all attributes of the fact table that can be used for clustering as all attributes an analyst might select must be a subset of these.

The second phase of CHAD applies a selection predicate of an analyst to derive a new dendrogram of clusters. This dendrogram is a compact representation of the data set that an analyst would have gained by applying the selection predicate on the fact table.

The third phase of CHAD applies a partitioning clustering algorithm on the selected dendrogram. The experimental prototype of this dissertation implements the k-means clustering algorithm. Other authors such as Bradley et al. have shown that a subset of the statistics that are stored in the selected dendrogram is sufficient to compute the result of an EM clustering algorithm, too.

Sufficiency of statistics is important to use them in cluster analyses. Hence, Subsection 5.1.2 discusses it in detail.

5.1.1 Comparing BIRCH and CHAD

Tian Zhang introduced the hierarchical clustering algorithm BIRCH in his PhD thesis in 1997. It is a very efficient clustering algorithm because it introduces the concept of being main memory optimised. It has influenced many different approaches since then—including the approach presented in this chapter.

Thus, there are several concepts of BIRCH that one also can find in CHAD. BIRCH and CHAD share some very similar phases. However, there are also some significant differences. Hence, this subsection discusses commons and major differences of BIRCH and CHAD.

Phase 1 of both algorithms construct a tree of clustering features. Clustering features are sufficient statistics of a subset of data. We will discuss sufficiency of statistics in the next subsection because sufficiency depends on the type of application.

The aim of BIRCH is to optimally use the main memory of the machine which is running BIRCH. It uses all data which an analyst has specified to be relevant for a specific analysis. There is no focus on re-use the result for other analyses—analyses needing only a fraction of the original data set. Hence, among BIRCH's parameter are a memory restriction and page size of the machine running it—both in bytes.

Yet, the clustering feature of CHAD has additional elements to enable projection and selection of clustering features, which means to enable specific preprocessing of a pre-mined dendrogram. It is unnecessary to know the exact set of data the analyst is interested in because CHAD can apply selection later on when this piece of information becomes known. Hence, CHAD always processes a fact table as a whole. Due to their more general applicability the clustering features of CHAD are called *general cluster features*.

The construction of the tree is almost identical for both algorithms. Both algorithms use the insertion method which R^+-tree uses. Yet, R^+-tree uses a bounding rectangle to insert tuples. In contrast to that, BIRCH and CHAD determine the most proximate leaf node by determining the distance of the nodes centroid and the current tuple.

The focus of CHAD is making the result re-usable. Thus, it chooses very small values for the capacity of nodes. By doing so, it receives more inner nodes which it can use for selection. In our tests, a minimum capacity of five entries per node has shown to be a good choice that generates a good balance of inner nodes and leaf nodes which means that there are much more leaf nodes than inner nodes but there is still a reasonable number of inner nodes for efficient selection of a general cluster feature tree. Contrary, BIRCH chooses the capacity that a node optimally fits a page of the machine. Hence, BIRCH receives fewer inner nodes compared to CHAD. This difference in focus becomes also obvious when considering other phases.

The pruning phase of BIRCH increases the level of aggregation of the clustering feature tree of BIRCH to improve the performance of phase 3, the global clustering.

In contrast to BIRCH, the second phase of CHAD adapts the general cluster feature to the specific needs of an analysis that is about to be performed in phase 3. The operations performed in phase 2 include selection, projection, and transformation of the nodes of the general cluster feature tree. The aim is to compute approximately the same result as a relational algebra expression on the data set that has been aggregated in phase 1 would compute. The resulting cluster feature tree is typically smaller than the general cluster feature tree. However, this is only a side-effect of selection and projection.

The description of phase 3 is very short in [81]. Particularly, it gives no solution to handle the problem of a proper initialisation. Many methods of initialisation use samples of the data to find an initial solution. Yet, taking

samples from the data means to give up the savings of time needed for accessing tuples when the algorithm needs to access tuples again and again—not for the clustering algorithm but each time for its initialisation.

CHAD uses the effect that the centroids of sub-clusters on higher level of a dendrogram often coincide in the same location. It tries several initial solutions which it finds by randomly picking as k centroids of cluster features, where k is an analyst-chosen number of clusters. Additionally, the statistics stored in CHAD are sufficient to determine the quality of each clustering.

5.1.2 Sufficiency of Summarising Statistics

For efficiency reasons the dendrogram only stores statistics of the tuples of a sub-cluster which are sufficient to use them in potential cluster analyses as replacement of the tuples of the sub-cluster.

This subsection introduces the statistics used by CHAD. It shows that these pre-computed statistics are sufficient to use them for computing specific cluster analyses.

A statistic is a *sufficient statistic* when it comprises all relevant data of a sample to predict a random variable. Spoken in terms of databases, a statistic which summaries a set of tuples is sufficient when using the statistic to compute an unknown value returns always the same result as when using the set of tuples. For instance, linear sum and count of a set of tuples are sufficient statistics for the centroid of that set of tuples. Using all tuples to compute the centroid returns no other result but requires much more space. Hence, it is an inefficient option.

Sufficiency of a statistic depends on the item that shall be computed. In other words, the type of result of a clustering algorithm determines whether a statistic is sufficient or not. Linear sum and count of tuples are sufficient to determine centroids. Hence, these statistics are sufficient for k-means clustering. Yet, for determining Gaussian clusters one also needs statistics about deviation of attributes. Additionally, several quality measures of clustering also need information concerning the deviation of attributes. The sum of squares of tuples can be used in combination with linear sum and count of tuples to determine the deviation of tuples. As this set of statistics is suitable to compute many important characteristics of a set of tuples, several authors such as Bradley et al. [6] and Zhang et al. [81] denote this set as *clustering features* or *cluster features*.

For being able to implement the general pre-process for clustering, CHAD adds elements to a clustering feature. In order to denote that this extended cluster feature is the (intermediate) result of the general pre-process, we call it a *general cluster feature*—we will define it below.

As a general cluster feature is an extended version of a cluster feature as used in the literature, many statements hold for extended and original version. Yet, there are some limitations of the original version of a cluster feature. Especially, a cluster feature is valid in a specific subset of a data set, only—we will discuss this issue later on.

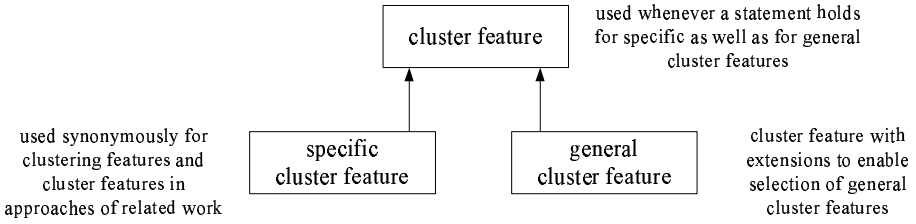


Figure 5.3: Taxonomy of cluster features

Therefore, we will use *cluster feature* as the generic term for general cluster feature and original version of cluster feature whenever a statement is true for both versions of cluster features, as depicted in Figure 5.3. Contrary, if a statement is only valid for the original version of a cluster feature, we will use the term *specific cluster feature*, which we define as follows.

Definition 5.1 *A specific cluster feature* $cf = (N, \vec{LS}, SS)$ *is a triple containing number, linear sum and sum of squares of all tuples represented by this clustering feature. A specific cluster feature represents a set of tuples* C*.*

Yet, sufficiency of specific cluster features is given as long as the vector space of an analysis is identical with the vector space of the data set of which the tuples of the specific cluster features originate from. When projecting attributes one cannot tell which part of the sum of squares of a specific cluster feature is the sum of squares of projected attributes or the sum of squares of non-projected attributes. Additionally, when selecting a subset of tuples of a data set using a selection predicate then one can estimate but not tell exactly whether tuples represented by the specific cluster feature fulfill the selection predicate or not. Hence, one cannot re-use the results of algorithms which use specific cluster features such as BIRCH when the selected data set contains less attributes as the original data set or a subset of tuples of it.

For distinguishing the different data sets and the vector spaces they span we further denote all data of the fact table as D and the set of attributes of this data set as A. Contrary, we reference the data set of an analysis and the set of attributes it consists of as D_a and A_a, respectively. Using these terms, we can re-formulate the above mentioned discussion as follows: A clustering feature of a data set $C \subseteq D_a$ which has the attributes A_a is only sufficient in an analysis when the data set coincides with the data set of that analysis $D_{a'}$ with attributes $A_{a'}$, i.e. $D_a = D_{a'} \wedge A_a = A_{a'}$.

Due to insufficient re-usability of a specific cluster feature, we extend the definition of a specific cluster feature and define a general cluster feature as follows:

Definition 5.2 *A general cluster feature* $cfg = (N, \vec{LS}, \vec{SS}, \vec{BL}, \vec{TR})$ *is a quintuple representing a set of tuples* $C \subset D$ *where* $N, \vec{LS}$ *and* $\vec{SS}$ *denote the number of tuples, the linear sum and the sum of squares for each dimension. Vectors* $\vec{BL}$

and $\vec{TR}$ describe a bounding rectangle of the tuples represented by this general cluster feature, where vector $\vec{BL}$ (bottom left) stores the minima of all dimensions and vector $\vec{TR}$ (top right) stores the maxima of all dimensions.

One can use the additional information of a general cluster feature to derive a specific cluster feature for an analysis which uses a data set D_a with attributes A_a. Data set D_a and attributes A_a must be subsets of the data set D with attributes A that has been used to determine the general cluster features.

As each data set D_a used in an analysis must be a subset of all available data D, the first phase of CHAD scans a fact table using all attributes of the fact table.

To be more specific, it uses all attributes except unique attributes and non-numerical attributes. If non-numerical attributes should be included one can pre-process that attribute by replacing it with an appropriate numerical attribute. Doing this, we erroneously consider a categorical attribute as numerical attribute. We can do so without negative effect if we omit range queries of this attribute, as we discussed already in Section 4.4.1.

The second phase of CHAD applies a selection and projection method on general cluster features to produce specific cluster features which are suitable to be used as sufficient statistics in a specific cluster analysis with a wide range of algorithms[1] that requires data set D_a.

Thus, a general cluster feature consists of sufficient statistics for any cluster analysis with a wide range of algorithms using data set $D_a \subset D$. In contrast to that, a specific cluster feature consists of sufficient statistics for analyses using exclusively data set D_a. Hence, general cluster features are widely applicable.

The next section will survey how CHAD efficiently constructs a dendrogram of general cluster features in a single scan of data.

5.2 CHAD Phase 1: Constructing a Parameter-Independent Tree of Summarising Statistics

As mentioned in Section 5.1.1 the first phases of BIRCH and CHAD are very similar. The first phase of BIRCH very efficiently computes a clustering feature tree. The different data structures they use are the major difference of the first phases of both algorithms. As BIRCH has been presented previously this section presents the first phase only in short.

Moreover, this section discusses the handling of specific items concerning efficient computation such as how to handle continuous insertion of tuples and distributed computation.

In the same way as BIRCH, CHAD uses a B^+-tree to store inner nodes. It also has a total capacity of nodes and a threshold for the maximum distance

[1]The experiments of this dissertation show sufficiency for k-means, only. However, as we discussed in this section and the section before, there exist a lot of approaches using specific cluster features as sufficient statistics for other partitioning clustering algorithms and density-based clustering algorithms.

a tuple might have to the centroid of a general cluster feature to be inserted into that general cluster feature. If a tuple is too far away of its nearest general cluster feature, CHAD inserts a general cluster feature as a new entry of a leaf node—increasing the tree that way. The initial value of this threshold is zero. Yet, if the size of the tree exceeds the capacity, the algorithm increases the threshold using linear regression.

The proceeding described above is identical for BIRCH and CHAD. Yet, they differ in some detail. For instance, CHAD uses a different way to re-assign entries of a node when the node is full and the algorithm needs to split it in two nodes. BIRCH takes each pair of entries and tests which pair is the most distant one. One of the entries of this pair stays in the currently split node while the other entry becomes the first entry of a new node. BIRCH moves the remaining entries from the old node to the new node if they are closer to the first entry of the new node than the other entry of the most distant pair of entries. CHAD uses the bounding rectangle of a general cluster feature to determine the two entries which are closest to the lower left and top right edges of the general cluster feature—the remaining proceeding is identical with BIRCH's proceeding. CHAD's proceeding is linear in the number of entries while BIRCH's proceeding is quadratic. The rationale of using the vectors $\vec{bl}$ and $\vec{tr}$ for splitting entries is that these points have the maximum possible distance within tuples of a cluster feature. There exist other pairs of corners of the bounding rectangle that also have the maximum distance. Yet, the vectors $\vec{bl}$ and $\vec{tr}$ already reside in the main memory. Therefore, we assume that the entries that are nearest to these two vectors have also a high distance. It might not be the pair of entries that has the highest distance but in our tests it was high enough for splitting entries well.

The remainder of this section focusses on using the first phase of CHAD as a pre-mining function for future analyses. This includes the handling of several problems which are continuous insertion of tuples, and distributed processing.

5.2.1 Continuously Inserting Tuples

Fact tables of data warehouses are subject to continuous change. In regular intervals ETL processes insert new tuples into the warehouse and modify or eliminate existing tuples. Yet, inserting new tuples is the most commonly found operation.

To analyse the current state of the data warehouse, it is necessary to keep a general cluster feature tree up-to-date.

Due to the additivity of cluster features one can insert a new tuple $\vec{x} = (x_1, \ldots, x_d)^T$ by adding a cluster feature representing the tuple, i.e. cluster feature $cf = (1, \vec{x}, (x_1^2, \ldots, x_d^2)^T)$, to its most similar cluster feature in the tree. Adding means to add cluster features element by element. Analogously, subtraction of cluster feature cf removes the tuple $\vec{x}$ again. A sequence of removal and insertion of tuples has the same effect as a modification of a tuple.

The re-computation of general cluster features is the same as the re-computation of specific cluster features. Yet, the bounding rectangle needs

special handling. When a tuple is added to an existing general cluster feature, one can determine the new bounding rectangle by checking if the new tuple is outside of the existing bounding rectangle. If it is outside then the value of the tuple in an attribute which is outside the bounding rectangle replaces either the minimum or the maximum value of this attribute.

Re-computing the bounding box of a general cluster feature after removing a tuple is simple as long as the values of this tuple does not assume any of the extreme values of the general cluster feature. If no value of the tuple which is about being removed is on the bounding rectangle then the tuple must be completely inside the bounding rectangle. Hence, there must be other tuples which define the bounding rectangle. Only tuples defining the bounding rectangle are critical when being removed because the information needed to find the new extreme value is missing in the general cluster feature.

Hence, CHAD does not re-compute the bounding rectangle of general cluster features of leaf nodes after removing tuples. By doing so, there could exist a smaller bounding rectangle of the tuples represented by the general cluster feature. However, the non-updated bounding rectangle has correct upper and lower limits of these tuples which are important to know for giving upper and lower limits of quality of results as is demonstrated in Subsection 5.10.3.

Yet, CHAD can re-compute the bounding rectangle of a inner node when removing a child node. Hence, it updates the parent nodes of a node that is removed.

When CHAD splits a node, it also re-computes the bounding rectangle of the node that has lost entries to another node. For doing so, CHAD adds the remaining entries of this node and replaces the old general cluster feature of this node by the result of this addition.

5.2.2 Making CHAD Applicable in Distributed Environments

A lot of data sets are distributed because the divisions of companies which own them are distributed. Centralising data is always possible but not always appropriate because transferring tuples costs time and bandwidth.

Pre-mining tuples locally and sending only local dendrograms to a central server saves time and bandwidth. It saves time because several machines compute dendrograms in parallel. Yet, it saves bandwidth because a dendrogram is smaller than the tuples used to build the dendrogram. Stefan Schaubschläger showed in his master's thesis that local pre-mining computes a result that is very similar to the result one receives when pre-mining the same data set globally. To be more specific, the cluster features of inner nodes of the top-most level of both general cluster feature trees shared the same centroids.

5.3 CHAD Phase 2: Deriving a Parameter-Dependent Tree of Summarising Statistics

The second phase of CHAD uses the general cluster feature tree to derive a new tree that fits the requirements of a specific cluster analysis. The general cluster features of the new tree represent the data needed for an analysis. Thus, the second phase of CHAD derives a parameter-dependent tree of summarising statistics from a parameter-independent tree. Hence, it implements the specific pre-processing phase for cluster analyses.

When an analyst traditionally would apply a selection predicate on a data set D to receive a selected data set D_a for a given analysis, an algorithm implementing the specific pre-processing phase such as CHAD must apply the same selection predicate on a set of intermediate results and auxiliary statistics to receive the intermediate results for data set D_a. Here, the general cluster feature tree is a tree storing intermediate results and auxiliary data.

Thus, CHAD needs a set of methods to derive a tree with entries that satisfy the analyst-given selection predicate. To do this, we need an algebra for selecting general cluster features. In Section 4.4.1 we limited the functionality of this algebra to queries performing slice and dice operations on the data set because they are most common in practise. In relational algebra we can formulate them as $\pi_{projectionattributes}(\sigma_{selectionpredicate}(\text{tuple set } T))$. In other words, we need a selection method and a projection method for general cluster features. The selection method selects the general cluster features of relevant tuples while the projection method prunes irrelevant attributes.

If D_a can be expressed as a combination of projection and selection of D^{all} such as $D_a = \pi_{A_a}(\sigma_{selectionpredicate}(D))$, it is possible to use this appropriately defined methods accordingly on a general cluster feature tree.

Furthermore, some analyses demand transforming data including re-scaling of attributes or adding new attributes from existing ones.

Pre-processing general cluster features for specific analyses is a novel concept introduced by CHAD—distinguishing it from other existing approaches such as [81] and [6]. Hence, a major part of the description of CHAD is reserved for the second phase. Thus, the next sections discuss each aspect of the specific pre-processing phase of CHAD in detail. They are organised as follows:

- Section 5.4 introduces a selection method for general cluster features. The selection method creates a new general cluster feature. Yet, the new general cluster feature only represents those tuples that fulfill the selection predicate.

- Section 5.5 introduces a projection method for general cluster features to choose only relevant attributes for an analysis.

- Section 5.6 introduces a method for transforming general cluster features according to linear and non-linear functions.

- Finally, Section 5.7 shows how to add new attributes by combining selection and transformation.

5.4 Selecting General Cluster Features

The general cluster feature tree, which is the result of the first phase of CHAD, represents all tuples of a fact table in a database. As no parameter of a specific analysis influences the first phase, the resulting tree is analysis-independent. However, specific analyses might need only subsets of the table which is represented by the general cluster feature tree.

As a consequence, there exists a selection operation in the framework of CHAD that creates a new general cluster feature tree that represents only those tuples that fulfill a user-given selection predicate.

Subsection 5.4.1 introduces the selection operation and describes an algorithm to approximate the selection operation. Approximation is necessary because the general cluster feature tree contains not all information to guarantee error-free selection—however, potential errors can be detected. Additionally, it is possible to determine an upper bound for the error that was made during approximation.

The subsections succeeding subsection 5.4.1 discuss important issues of the computation within the selection operation.

Subsection 5.4.2 describes pre-processing of the selection predicate which simplifies further computation because it avoids several cases in case differentiation.

Subsection 5.4.3 shows the usage of the bounding rectangle of a general cluster feature for selecting the general cluster feature. All general cluster features selected due to their bounding rectangle are selected error-free. In addition to that, the bounding rectangle provides a guaranteed limit of the error made by the estimation processes which are shown in subsections 5.4.4, 5.4.5, and 5.4.6

Subsection 5.4.4 introduces the estimation of specific values of a newly selected general cluster feature while subsections 5.4.5 and 5.4.6 describe the most important features of estimation in detail.

Last but not least, Subsection 5.4.7 shows the update of the bounding rectangle of a selected general cluster feature according to the selection predicate.

5.4.1 Definition of Selection Operation

Definition 5.3 *Let $\mathcal{P}$ denote the set of predicates over a tuple $\vec{x}$ in disjunctive normal form and D a set of tuples. Then, the selection of a general cluster feature $cfg_* \in CFG$ representing sub-cluster $C \subset D$ using the selection predicate $p \in \mathcal{P}$ is a function $\varsigma : CFG \times \mathcal{P} \rightarrow CFG$ that returns a general cluster feature $cfg_{**} \in CFG$, such that*

1. *the number of tuples N_{**} represented by general cluster feature cfg_{**} equals the number of tuples in general cluster feature cfg_* that fulfill the selection*

predicate p

$$N_{**} = |\{\vec{x} \in C | p(\vec{x})\}|, \tag{5.1}$$

2. *the linear sum $\vec{LS}_{**}$ of the tuples represented by general cluster feature cf_{**} equals the vector sum of all tuples in general cluster feature cfg_* that fulfill the selection predicate p*

$$\vec{LS}_{**} = \sum_{\vec{x} \in C : p(\vec{x})} \vec{x}, \tag{5.2}$$

3. *the vector SS_{**} has for each attribute a an element $ss_{**}[a]$ that stores the sum of squares of all attribute values $\vec{x}[a]$ of the tuples in general cluster feature cfg_* that fulfill the selection predicate p*

$$ss_{**}[a] = \sum_{\vec{x} \in C : p(\vec{x})} (\vec{x}[a])^2. \tag{5.3}$$

4. *the vector BL_{**} has for each attribute a an element $bl_{**}[a]$ that stores the minimum of all attribute values $\vec{x}[a]$ of the tuples that fulfill the selection predicate p*

$$bl_{**}[a] = \min_{\vec{x} \in C : p(\vec{x})} \vec{x}[a]. \tag{5.4}$$

5. *the vector TR_{**} has for each attribute a an element $tr_{**}[a]$ that stores the maximum of all attribute values $\vec{x}[a]$ of the tuples that fulfill the selection predicate p*

$$tr_{**}[a] = \max_{\vec{x} \in C : p(\vec{x})} \vec{x}[a]. \tag{5.5}$$

The selection predicate $p \in \mathcal{P}$ consists of a logical statement over tuple $\vec{x}$ in disjunctive normal form.

Although the definition of selection of a general cluster feature is valid for any predicate in disjunctive normal form, the statistics in a general cluster feature are insufficient to determine the correct values of selected statistics of all potential predicates in disjunctive normal form. Thus, we narrow the set of valid selection predicates to selection predicates that obey the syntax which we introduce next.

A selection predicate p consists of several disjoint terms t_i.

$$p = \bigvee_{i=1}^{r} t_i, r \in \mathbb{N}$$

Each term t_i consists of a set of literals that are conjuncted with each other.

$$t_i = \bigwedge_{j=1}^{s} x[a] \theta c_j, \theta \in \{<, \leq, =, \geq, >\}, s \in \mathbb{N}, a \in A, c_j \in \mathbb{R}, j \in \{1, 2, \ldots, s\}$$

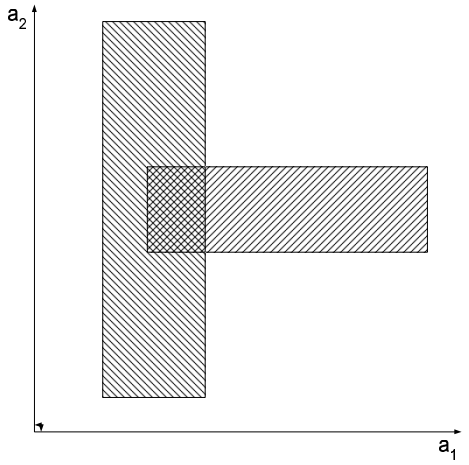


Figure 5.4: selection predicate in DNF describes a set of hypercubes in d-dimensional space

The literals in this statement have the form $\vec{x}[a]\theta c_j$, where $\vec{x}[a]$ denotes a tuple's value of attribute a and θ is one of the operators $\{<, \leq, =, \geq, >\}$. Finally, c_j is a user-defined constant in a set of s constants.

CHAD includes a selection algorithm to realise the selection of a general cluster feature as described in definition 5.3. Yet, the selection of general cluster features is defined on the tuples represented by general cluster features. Yet, the tuples are no longer accessible when performing selection. Thus, the new values of a general cluster feature after selection must be derived from the old values of the general cluster feature. This proceeding can be erroneous.

CHAD's selection algorithm tries to avoid erroneous selections if possible or to minimise the total error if an errorfree selection of a general cluster feature is impossible.

In detail, the selection algorithm generates a new general cluster feature from a general cluster feature tree in the following sequence of steps:

1. Use the selection predicate to determine a set of disjoint hypercubes H that contain all points that fulfill the predicate. Each hypercube is represented by its corners.

2. Create a new initially empty tree.

3. Traverse the general cluster feature tree and test the bounding rectangle of each visited element if it intersects with any of the hypercubes H. Depending on the result of this test do one of the following steps:

   (a) If the bounding box is completely outside of all of the hypercubes skip the node and all its descendants.

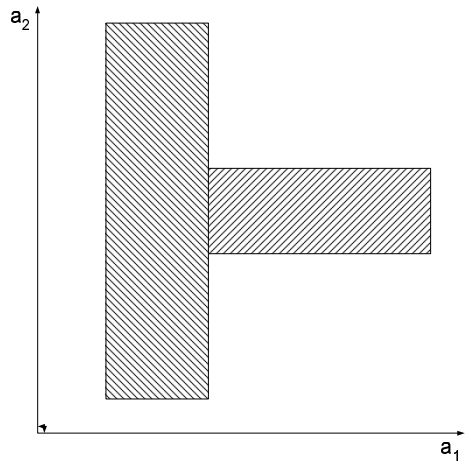


Figure 5.5: pruned version of selection criteria of figure 5.4

(b) If the bounding box is completely inside of any of the bounding cubes insert all leaf nodes of that branch into the new tree.

(c) If the bounding box of an inner node and at least one cube partially overlap then traverse the branch of this node.

(d) If the bounding box of a leaf node and at least one cube partially overlap then estimate the summarising statistics of the tuples that fulfill the selection predicate.

5.4.2 Pruning the Selection Predicate

Figure 5.4 shows the visualisation of a selection predicate in a 2-dimensional vector space. The selection predicate consists of two disjunct statements. As shown in figure 5.4, the rectangles, that represent the set of tuples that fulfill one of these statements, overlap.

Overlapping areas have to be taken into account or better avoided when estimating the proportion of tuples of a general cluster feature that fulfill the selection predicate. It is faster to make the boxes that represent disjunct statements non-overlapping than to handle overlapping parts for each node that is to be selected.

Thus, CHAD shatters the boxes of the selection predicate in a set of non-overlapping boxes—or hypercubes in a d-dimensional vector space. The result of the selection predicate in figure 5.4 after being shattered looks like in figure 5.5.

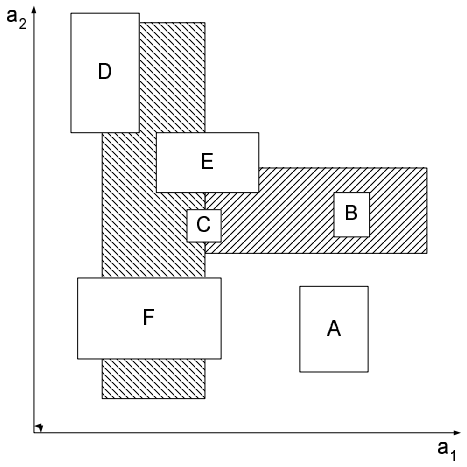


Figure 5.6: selection criteria and general cluster feature tree entries

5.4.3 Using the Bounding Rectangle of a General Cluster Feature for Selection

When selecting a node, CHAD first tests whether the bounding rectangle of an entry is sufficient for determining the result of selection. The bounding rectangle of an entry is a border that guarantees that there are no tuples of that entry outside the border. To be more specific, it guarantees that all tuples of an entry fulfill the selection predicate if the bounding rectangle is completely inside a hypercube of the selection predicate. If the bounding rectangle is completely overlapping-free of all hypercubes of the selection predicate, none of the entry's tuples fulfills the selection predicate.

Figure 5.6 shows the bounding rectangles of six entries and two hypercubes of the selection predicate. Rectangle A is completely outside of the hypercubes. Thus, CHAD omits this node because all of its tuples cannot fulfill the selection predicate. Rectangle B is completely inside the right hypercube. Thus, all of its tuples fulfill the selection predicate. Consequentially, CHAD inserts the entry unchanged into the new selected tree if it is an entry of a leaf node. If it is an entry of an inner node, CHAD takes all leaf node entries of that branch and insets them into the new selected tree.

Rectangles D, E, and F intersect one or both of the hypercubes of the selection predicate. Depending on the type of the entry of the rectangle—leaf node entry or inner node entry—, CHAD performs selection differently. If the intersecting rectangles belong to an inner node entry, CHAD takes the entries of the child node that is linked to the current inner node entry into account and tests for intersection once more. CHAD iterates this process until either a leaf node entry is selected or all entries' bounding boxes are completely inside or outside the hypercubes of the selection predicate.

Leaf node entries the bounding rectangles of which intersect the selection

predicates' hypercubes are selected using estimation. For this purpose, the bounding rectangles are insufficient. The detailed description of estimation is postponed to later paragraphs of this section.

Rectangle C is also completely inside the set of hypercubes that represent the selection predicate but not a single hypercube contains rectangle C completely. Using the proceeding as mentioned above means that estimation is used in a case where an errorless result could have been determined. Yet, for easiness of computation this special case is omitted which means that we accept the small error that occurs when the estimated values of both parts of rectangle C are lower than the correct values. If the estimation overestimates the real values, i.e. it would select more tuples than there are tuples in the general cluster feature, then the algorithm inserts the original general cluster feature into the new tree without changing its elements. Hence, there is no error made in a case of overestimation. In our tests of the selection method we could not observe an error that happened because of a entry with a rectangle like rectangle C because when we used a range query the bounding rectangles of most leaf node entries was either completely in or completely out of a hypercube, i.e. estimation was needed in rare cases, only. Furthermore, in rare cases, in which estimation was used, the phenomenon of a rectangle in a similar situation as rectangle C never occurred.

5.4.4 Estimating the New Attribute Values of Leaf Node Entries of Attributes That are Not Part of the Selection Predicate

If the bounding box of a leaf node entry is neither completely inside of one of the hypercubes representing the selection predicate p or completely outside of all of them, there is no longer a guaranty that the statistics of the selected general cluster feature are correct. For instance, all tuples represented by the general cluster feature of a leaf node entry may fulfill the selection predicate although the bounding rectangle intersects a hypercube of the selection predicate as is shown in Figure 5.7. The hypercube in Figure 5.7, that is depicted as a hatched rectangle, intersects the bounding rectangle, that is marked with dotted lines, but none of the tuples within the bounding rectangle is within the hypercube.

As the hypercubes that represent the selection predicate do not overlap, we can determine a general cluster feature cfg_{i**} for each term t_i of the selection predicate and add them to the new general cluster feature cfg_{**}, as shown in Formula 5.6 where $\bigoplus$ denotes the addition of general cluster features which we discussed in Section 5.2. We call a general cluster feature cfg_{i**} of a term t_i a component of the general cluster feature cfg_{**}. Components are only temporally stored during the selection process.

$$cfg_{**} = \bigoplus_{t_i} cfg_{i**} \tag{5.6}$$

This subsection and the following subsections describe the computation of the components of the general cluster feature.

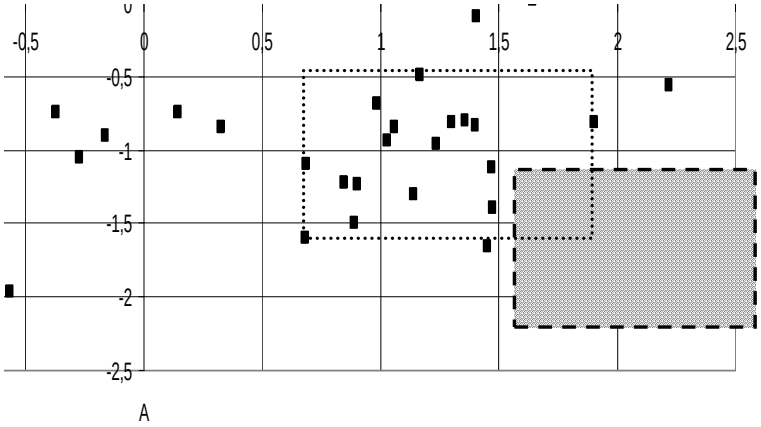


Figure 5.7: hypercube and bounding rectangle intersect although all tuples fulfill term

Estimation is used to determine the most-likely value of the new selected statistics of the general cluster feature that is about to be selected.

Due to the optimal distribution of tuples within a general cluster feature being unknown, choosing a different distribution that approximates the optimal distribution is necessary.

using normal distribution for approximation With mean and standard deviation of the tuples within the general cluster feature being known, we approximate the real distribution of each attribute with a normally distributed variable because normal distribution or a set of normally distributed variables can approximate many different distributions.

assuming conditional independency We assume conditional independency between each pair of attributes because it is unknown whether attributes are correlated positively or negatively. Conditional independency favours none of these extremes.

Approximating an arbitrary distribution with a set of normally distributed variables uses the well-known technique of kernel estimation in statistics [64]. Kernel estimation uses a kernel function, a function of a probability density function with parameters, and selects a set of variables which are distributed according to the kernel function—yet, each time with differently chosen parameters. The basic principle of kernel estimation is to find a small set of variables with well-chosen parameters that in their sum fit the arbitrary density function best.

Figure 5.8 depicts the probability density function of a single attribute of an arbitrary distribution which we will use to illustrate kernel estimation. The density function shows several concentration points on a macroscopic level but also shows areas of concentration at microscopic level. When clustering data

hierarchically, each cluster and sub-cluster represents a set of tuples that are concentrated globally or locally. Thus, the cluster feature of a leaf node entry represents a single local concentration of tuples.

Yet, if we decide to approximate an arbitrary density function on a given level with a set of Gaussian density functions, we cannot use a different set of Gaussian density functions of another level at the same time because this would mean to have two different assumptions of the distribution of tuples at the same time. We circumvent this problem by approximating the distribution of tuples in cluster features with Gaussian density functions at leaf level, only, because this level is the most fine-grained one.

The Gaussian density function is a good kernel function because it approximates many arbitrary functions quite well which is illustrated by some examples in Figure 5.9—the normal distribution with accordingly chosen parameters is depicted under each example of distribution. Except for the worst case example, each other example is well approximated by a normally distributed variable. Yet, these approximations are good approximations in an area around the mean. As normal distribution has unlimited range while uniform distribution has a constant positive probability density within a specific interval and zero density without this interval, normal distribution can approximate uniform distribution only in a limited interval around the mean of the normal distribution.

Due to the concentration of tuples around the mean, most tuples are within an interval around the mean and only few tuples outside this interval. Hence, there are only few wrongfully selected tuples when using Gaussian approximation for selection.

Moreover, when applying approximation on a set of general cluster features, the error made by overestimating the number of tuples fulfilling a selection predicate in a set of cluster features can remedy the error made by underestimating the number of tuples fulfilling a selection predicate in another set of cluster features. Hence, two different types of errors can compensate each other. If, for instance, there is a statement $\vec{x}[a] < c$ in the selection predicate and tuples are uniformly distributed in attribute a, then cluster features having $\vec{ls}[a]/n > c$ tend to underestimate the number of tuples fulfilling the selection predicate, while cluster features having $\vec{ls}[a]/n < c$ tend to overestimate that number.

The hierarchical clustering algorithm used in the first phase of CHAD seeks a data set to find concentrations of tuples within the data set on different levels. Yet, there is no guaranty that each sub-cluster exactly finds a concentration of tuples. If it fails to do so, it might happen that the tuples of a cluster feature are distributed like shown in example d) of Figure 5.9. This example, denoted as worst case, can hardly approximated by a single Gaussian function. It would be better to split the cluster feature into a pair of cluster features. Yet, again, the error made by overestimating number of tuples can compensate the error made by underestimating. Therefore, even in worst case, error might be compensated.

Storing the statistics that are necessary to determine conditional dependency of attributes such as the covariance matrix would be possible but is omitted because it would increase the size of a general cluster feature in an intolerable

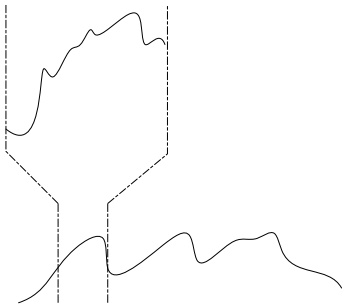


Figure 5.8: Probability density of a single attribute at macro-level (bottom) and at micro-level (top)

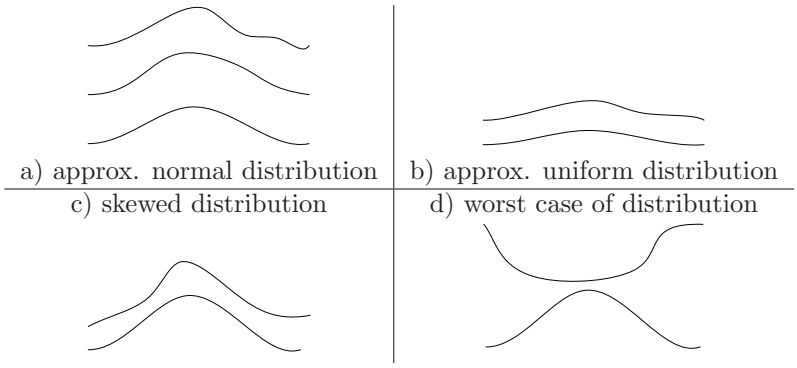


Figure 5.9: Approximating density functions with Gaussian distribution

way. Storing a covariance matrix requires $\mathcal{O}(d^2)$ storage space and would make the CHAD inapplicable to data sets with many attributes.

According to Glymour et al. "...assumptions are typically false but useful"[23, p. 13] which means that an assumption simplifies the original problem to a problem that is easier to solve while the unconsidered parts of the original problem are unable to influence the solution significantly.

The experience made with Naïve Bayes Classifier shows that assuming conditional independency can deliver good results even if attributes are correlated [36, p. 298].

Assuming conditional independency of attributes simplifies the computation of linear sum, sum of squares, bottom left vector, and top right vector if the attribute is not part of the selection predicate.

Due to conditional independency mean and deviation parameters of each attribute a that is not part of the selection predicate remain the same after selection such that the following conditions must hold. Determining the values of attributes that are part is described in Subsection 5.4.6.

As the mean, which is computed as quotient of linear sum and count of tuples, is not changing during selection, the quotients of linear sums and counts of tuples of the original general cluster feature cfg_* and a component cfg_{i**} must be equal.

$$\frac{\vec{ls}_{i**}[a]}{N_{i**}} = \frac{\vec{ls}_*[a]}{N_*} \tag{5.7}$$

Reformulating equation 5.7, we receive the formula to determine the linear sum $ls_{i**}[a]$ of a component.

$$\vec{ls}_{i**}[a] = \frac{N_{i**}}{N_*}\vec{ls}_*[a] \tag{5.8}$$

The fraction $\frac{N_{i**}}{N_*}$ is the proportion of tuples in the general cluster feature that fulfill the term t_i. By multiplying the same proportion with the old value of the sum of squares of attribute a, we receive the according new value.

$$\vec{ss}_{i**}[a] = \frac{N_{i**}}{N_*}\vec{ss}_*[a] \tag{5.9}$$

Due to the absence of tuples, correctly determining minimum and maximum of the selected portion of the tuples of a general cluster feature is no longer possible. Yet, the bounding rectangle of the original general cluster feature is also a bounding rectangle of the component. Thus, old and new values are identical.

$$\vec{BL}_{i**} = \vec{BL}_* \tag{5.10}$$

$$\vec{TR}_{i**} = \vec{TR}_* \tag{5.11}$$

5.4.5 Determining the Proportion of Tuples Fulfilling a Term of the Selection Predicate

As the proportion $\frac{N_{i**}}{N_*}$ of tuples fulfilling term t_i is so important for selecting general cluster features, let q_i denote this proportion $q_i = \frac{N_{i**}}{N_*}$. Determining the

unbounded on the left side	bounded	unbounded on the right side
$x[a] < c_j$	$c_i < x[a] < c_j$	$c_i < x[a]$
	$c_i \leq x[a] < c_j$	
$x[a] \leq c_j$	$c_i < x[a] \leq c_j$	$c_i \leq x[a]$
	$c_i \leq x[a] \leq c_j$	

Table 5.1: types of conditions in a term of a selection predicate

proportion q_i differs with the type of comparison operators $\theta = \{<, \leq, =, \geq, >\}$ used in the selection predicate.

Due to conditional independency, the total proportion of tuples fulfilling the condition denoted by a term is the product of the proportions $q_i[a]$ of tuples fulfilling the condition denoted by the literals limiting a single attribute a. Thus, we sort the literals in a term by attributes and determine the proportion of tuples fulfilling the condition of each attribute.

Assume that a term t_i of the selection predicate p consists of the attributes $A_b \subset A$ and a set of constants C.

$$q_i = \prod_{a \in A_b} q_i[a] \tag{5.12}$$

For limiting the number of potential cases, we group the literals of term t_i by the attributes they constrain and eliminate redundant statements. Thus, by complete enumeration we receive the eight different types of conditions as shown in table 5.1. For instance, if there is

$$x[airtemp] > 20 \wedge x[airtemp] \geq 15 \wedge x[airtemp] < 30$$

in a term of the selection predicate, we receive the pruned condition

$$20 < x[airtemp] < 30.$$

The condition $x[a] = c$ is expressed as $c \leq x[a] \leq c$. We postpone this case because it needs special handling.

As we discussed above, the hypercube that represents term t_i intersects the bounding rectangle of the general cluster feature in at least one dimension. Additionally there is no dimension in which the corners of the bounding rectangle is completely outside of the hypercube. Otherwise, the hypercube would be completely outside or inside the bounding rectangle which means that the general cluster feature would have already been handled as described in the previous subsection.

For all non-intersecting attributes[2] the proportion of tuples fulfilling the condition of that attribute is 1 because non-intersection in a dimension means

[2]Non-intersecting attributes are attributes that either are not part of the selection predicate, or the bounding box of the general cluster feature is completely inside the hypercube when considering this attribute, only.

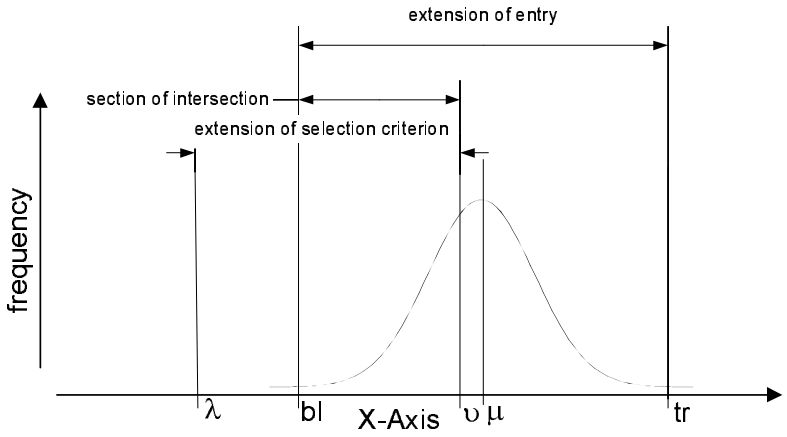


Figure 5.10: selection predicate includes condition $x[a] \leq \nu$

that all tuples fulfill the condition of that attribute.

$$q_i[a] = 1 \left\{ \begin{array}{l} \text{if } x[a]\theta c_j \text{ in term } t_i \wedge tr[a] < c_j \\ \text{if } c_i\theta_1 x[a]\theta_2 c_j \text{ in term } t_i \wedge c_i < bl[a] \leq tr[a] < c_j \\ \text{if } c_i\theta x[a] \text{ in term } t_i \wedge c_i < bl[a] \end{array} \right\} \text{with } \theta, \theta_1, \theta_2 \in \{<, \leq\} \tag{5.13}$$

Assume that the selection predicate includes a term having a condition in the form $x[a]\theta\nu$, where ν is a constant used as upper bound in a condition and θ is either $<$ or $\leq$, i.e. the condition is one expression of the left side of Table 5.1. Further assume there is an overlap of the interval $[bl[a], tr[a]]$ of the bounding rectangle and the interval $(-\infty, \nu]$ of the condition.

Figure 5.10 shows a special case of a term including the condition $x[a] \leq \nu$ because it contains the condition $\lambda \leq x[a] \leq \nu$ where $\lambda, \nu \in \mathbb{R}$ but the minimum $bl[a]$ of attribute a is greater than the lower bound λ of the selection condition. Thus, the condition $\lambda \leq x[a] \leq \nu$ consists of two conditions $\lambda \leq x[a] \wedge x[a] \leq \nu$, where the left condition is true for all tuples of the general cluster feature.

Therefore, the conditions $\lambda \leq x[a] \leq \nu$ with $bl > \lambda$ and $x[a] \leq \nu$ require the same statistical estimation.

Let $\mu[a]$ and $\sigma[a]$ denote the mean and standard deviation of the current leaf node in the dimension spanned by attribute a. Basing on this assumption, the z-normalised value of $x[a]$ can be determined—which is $\frac{x[a]-\mu[a]}{\sigma[a]}$—to be used in the gaussian probability density function ϕ.

Using the assumption of normal distribution of tuples within the cluster represented by the general cluster feature we determine the proportion of tuples fulfilling the selection condition $x[a]\theta\nu$ by determining the integral over the probability density function of the normal distribution, i.e. the area between the Gaussian curve, the x-axis, and the vertical line at $x[a] = \nu$ in Figure 5.6.

The cumulative Gaussian function Φ at the z-normalised value $(\nu - \mu[a])/\sigma[a]$ of ν returns the result of the integral $\int_{-\infty}^{\nu} \phi(\frac{x[a]-\mu[a]}{\sigma[a]})dx[a]$, which is the esti-

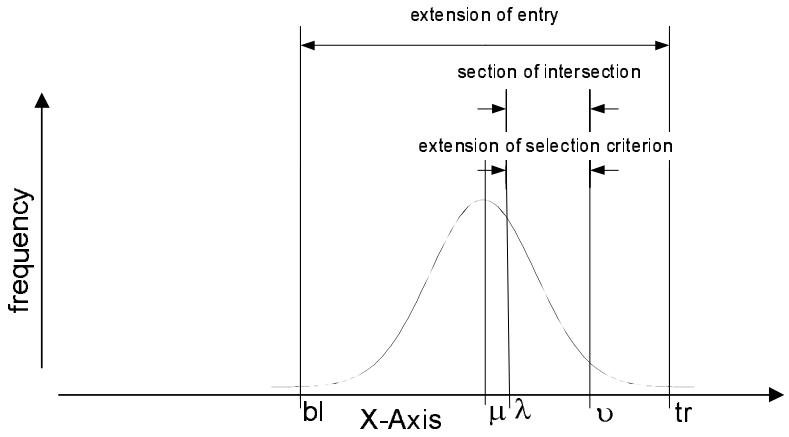


Figure 5.11: selection predicate includes condition $\lambda \leq x[a] \leq \nu$

mated proportion of the tuples of the general cluster feature fulfilling the condition.

$$q_i[a] = \int_{-\infty}^{\nu} \phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right) dx[a]$$

$$q_i[a] = \Phi\left(\frac{\nu-\mu[a]}{\sigma[a]}\right) \tag{5.14}$$

Estimation must consider two boundaries in cases where a term t_i includes a condition $\lambda \leq x[a] \leq \nu$ and there is no guaranteed boundary on one of the sides of the interval $[\lambda, \nu]$, i.e. the boundaries λ and ν of the condition are completely inside of the boundaries of the general cluster feature as shown in Figure 5.11.

Again, the proportion $q_i[a]$ is the result of the integral of the probability density function between the limits of the selection condition—yet, in this case the integral is bounded on both sides, i.e. by the lines $x[a] = \lambda$ and $x[a] = \nu$, respectively.

$$q_i[a] = \int_{\lambda}^{\nu} \phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right) dx[a] \tag{5.15}$$

Determining the result of the integral in equation 5.15 with the cumulative gaussian function requires splitting the integral in two integrals because the cumulative gaussian function only determines the probability that a random variable is less than the argument's value.

$$q_i[a] = \underbrace{\int_{-\infty}^{\nu} \phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right) dx[a]}_{\Phi\left(\frac{\nu-\mu[a]}{\sigma[a]}\right)} - \underbrace{\int_{-\infty}^{\lambda} \phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right) dx[a]}_{\Phi\left(\frac{\lambda-\mu[a]}{\sigma[a]}\right)} \tag{5.16}$$

$$q_i[a] = \Phi\left(\frac{\nu-\mu[a]}{\sigma[a]}\right) - \Phi\left(\frac{\lambda-\mu[a]}{\sigma[a]}\right) \tag{5.17}$$

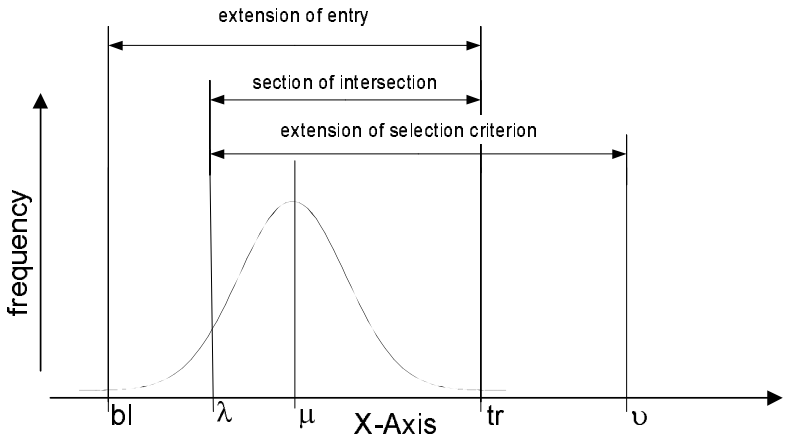


Figure 5.12: selection predicate includes condition $x[a] \geq \lambda$

A condition having the form $\lambda \theta x[a]$ with operator $\theta \in \{<, \leq\}$ is handled in analogous way to a condition having the form $x[a]\theta\lambda$. As the conditions $x[a] \leq \lambda$ and $x_{a_v} > \lambda$ divide all attribute values $x[a]$ in two disjoint and exhaustive subsets, the proportion of tuples fulfilling a selection condition like $x[a] > \lambda$ is the difference of the proportion fulfilling the selection condition $x[a] \leq \lambda$ to 1.

$$q_i[a] = 1 - \Phi\left(\frac{\lambda - \mu[a]}{\sigma[a]}\right) \tag{5.18}$$

Equality in conditions such as $x[a] = c$ is covered by using the bounding rectangle of the general cluster feature as described in the previous subsection. The following paragraphs will discuss this point.

The integral $\int_c^c \phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right) dx[a]$ is 0 unless the standard deviation is 0, too. For the standard deviation being 0, minimum and maximum of attribute a must coincide. Yet, if minimum and maximum are equal, testing for intersection with the bounding rectangle can either be true for all tuples or for no tuples. Thus, this case must have been handled before. In other words, either the general cluster feature has been dropped or it has been completely moved into the new tree.

One needs testing for equality when using categorical attributes because categorical attribute do not permit range queries. Moreover, it makes no sense to sum up values. Thus, linear sum and sum of squares of this attribute make no sense except all values in a general cluster feature of this attribute are identical. Yet, when they are identical, we can determine this value as quotient $ls[shopid]/N$. Therefore, CHAD calculates sums of values of categorical attributes which it represents as numbers knowing that it cannot use these sums when it has to add distinct values. Therefore, it is necessary to avoid adding distinct values into the same cluster feature. CHAD inserts two distinct values of an attribute a into two different cluster features if the distance of these two

values exceeds the current threshold of the general cluster feature tree which depends on the size of the cluster feature tree. The more nodes a cluster feature tree might have, the smaller is the threshold. In our tests, we experienced that if the number of nodes exceeds the number of distinct values then CHAD inserts different categorical values into different cluster features.

We want to illustrate the argumentation above with the running example. Assume an analyst wants to select sales transactions which were bought online—buying online means that the shop, in which the transaction happened, is the online shop of the publishing company, the identifier of which is stored in attribute*shopid*. As *shopid* is a categorical attribute, we can use a cluster feature only if it contains only the sum of N times the same value. To ensure this, we choose a size of the cluster feature tree that exceeds the number of shops in the running example. By doing so, the deviation of attribute *shopid* of a leaf node entry is zero. Yet, we do not need this deviation because minimum and maximum are also identical. Therefore, we can test for each node if its minimum and maximum equals the identifier of the online shop. As CHAD performs the test that uses the bounding rectangle before estimation, one never need to handle the special case in which deviation is zero.

The operators $<$ and $\leq$ can be handled in the same way. If the standard deviation of the general cluster feature cfg_* is not zero in the attribute a, the resulting components of a term with $x[a] < \nu$ and a term $x[a] \leq \nu$ have the same values because the probability of $x[a] = \nu$ is 0. Therefore, the integral over the probability density function is the same, regardless the upper bound of the integral is ν or an infinitesimal smaller value, as shown in Equation 5.19.

$$\lim_{u \overset{\leq}{\rightarrow} \nu} \int_{-\infty}^{u} \phi\left(\frac{x[a] - \mu[a]}{\sigma[a]}\right) dx[a] = \int_{-\infty}^{\nu} \phi\left(\frac{x[a] - \mu[a]}{\sigma[a]}\right) dx[a] \tag{5.19}$$

5.4.6 Estimating the Statistics of Attributes that are Part of a Term

Updating the statistics of an attribute a that is part of a term t_i of the selection predicate requires different handling because in contrast to attributes that are not part of term t_i mean and deviation of the tuples of the general cluster feature change.

Consequentially, the statistics linear sum and sum of squares of a component cfg_{i**} are determined using the expectation values of mean and sum of squares given the condition t_i $\mathcal{E}(x[a]|t_i)$ and $\mathcal{E}(x[a]^2|t_i)$, respectively.

$$ls_{i**}[a] = q_i[a] N_* \mathcal{E}(x[a]|t_i) \tag{5.20}$$

$$ss_{i**}[a] = q_i[a] N_* \mathcal{E}(x[a]^2|t_i) \tag{5.21}$$

The expectation value of mean $\mathcal{E}(x[a])$ is defined as the sum of the products of each potential value of statistic variable $x[a]$ and the according probability—for continuous variables the integral over the product of variable and the according

probability density function is used. Depending on the form of condition in term t_i, the expectation value is one of the integrals shown in formula 5.22.

$$\mathcal{E}(x[a]) = \begin{cases} \int_{-\infty}^{\nu} x[a]\phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right)dx[a] \text{ if } x[a] \lesseqgtr \nu \text{ in term } t_i \\ \int_{\lambda}^{\nu} x[a]\phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right)dx[a] \text{ if } \lambda \lesseqgtr x[a] \lesseqgtr \nu \text{ in term } t_i \\ \int_{\lambda}^{\infty} x[a]\phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right)dx[a] \text{ if } \lambda \lesseqgtr x[a] \text{ in term } t_i \end{cases} \tag{5.22}$$

The antiderivative of the integral for determining the expectation value of mean is the negative of the Gaussian density function, as can be shown by differentiation. For easier integration we substitute the term in the integral with $z := (x[a] - \mu[a])/\sigma[a]$ and receive the integral $\mu[a] + \sigma[a] \int z\phi(z)dz$.

$$\int z\phi(z) = \int z\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}z^2}dz = -\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}z^2} + C = -\phi(z) + C \tag{5.23}$$

By re-substituting variable z and inserting the antiderivative in formula 5.22 we receive formula 5.24.

$$\mathcal{E}(x[a]) = \begin{cases} \mu[a] - \sigma[a] * \phi\left(\frac{\nu-\mu[a]}{\sigma[a]}\right) \text{ if } x[a] \lesseqgtr \nu \text{ in term } t_i \\ \mu[a] + \sigma[a]\left(\phi\left(\frac{\lambda-\mu[a]}{\sigma[a]}\right) - \phi\left(\frac{\nu-\mu[a]}{\sigma[a]}\right)\right) \text{ if } \lambda \lesseqgtr x[a] \lesseqgtr \nu \text{ in term } t_i \\ \mu[a] + \sigma[a] * \phi\left(\frac{\lambda-\mu[a]}{\sigma[a]}\right) \text{ if } \lambda \lesseqgtr x[a] \text{ in term } t_i \end{cases} \tag{5.24}$$

Determining the expectation value of the sum of squares is similar. Yet, the integral $\int x[a]^2\phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right)dx[a]$, that determines the expectation value of the sum of squares, has no polynomial antiderivative. Again, we substitute variable $x[a]$ with the z-normalised variable $z = \frac{x[a]-\mu[a]}{\sigma[a]}$ for easier computation.

$$z^2 = \frac{(x[a] - \mu[a])^2}{\sigma[a]^2} \Leftrightarrow x[a]^2 = \sigma[a]^2z^2 + 2\mu[a]x[a] - \mu[a]^2$$

By inserting z^2 in the integral that determines the expectation value of the sum of squares, we receive an integral where most terms are known.

$$\int x[a]^2\phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right)dx[a]$$

$$= \int \sigma[a]^2z^2\phi(z)dz + \int 2\mu[a]x[a]\phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right)dx[a] - \int \mu[a]^2\phi(z)dz =$$

$$= \sigma[a]^2 \underbrace{\int z^2\phi(z)dz}_{\text{see equation 5.28}} + 2\mu[a] \underbrace{\int x[a]\phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right)dx[a]}_{\mathcal{E}(x[a])} - \mu[a]^2 \underbrace{\int \phi(z)dz}_{\Phi(z)} \tag{5.25}$$

The antiderivative of term $\int z^2\phi(z)dz$ is as follows:

$$\int z^2\phi(z)dz = \int z^2 \frac{1}{\sqrt{2\pi}}\, e^{-\frac{1}{2}z^2} dz = -\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} z + \underbrace{\frac{1}{2}\operatorname{erf}\left(\frac{z}{\sqrt{2}}\right)}_{\Phi(z),\ \text{cf. } 5.27} + C \quad (5.26)$$

The antiderivative of integral $\int z^2\phi(z)dz$ includes the density function of the Gaussian distribution in its first addend. The second addend of the antiderivative is the cumulative Gaussian function Φ—hereby, function *erf* is the so-called *error function*, which is defined as $\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^2 e^{-t^2} dt$ which is also the limit of the infinite series $\operatorname{erf}(z) = \lim_{n\to\infty} \frac{2}{\sqrt{\pi}} \sum_{i=0}^{n} \frac{(-1)^i z^{2i+1}}{i!(2i+1)}$.

$$\int \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} dz = \frac{1}{2}\operatorname{erf}\left(\frac{z}{\sqrt{2}}\right) + C = \Phi(z) + C \quad (5.27)$$

Therefore, we can re-write the antiderivative of integral $\int z^2\phi(z)dz$ in 5.26 using the transformation in 5.27 as follows:

$$\int z^2\phi(z)dz = \Phi(z) - \phi(z)z + C \quad (5.28)$$

With the antiderivative of integral $\int z^2\phi(z)dz$ we have all terms of the antiderivative of the integral that determines the expectation value of the sum of squares. We insert these terms into Equation 5.25 and sort the terms in terms using the cumulative Gaussian density function and terms using the Gaussian density function. For shortness of description, we omit re-substituting variable z.

$$\left(\sigma[a]^2 - \mu[a]^2\right)\Phi(z) - \left(\sigma[a] + \sigma[a]^2 z\right)\phi(z) + C$$

$$\overset{z=\frac{x[a]-\mu[a]}{\sigma}}{\Leftrightarrow} \left(\sigma[a]^2 - \mu[a]^2\right)\Phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right) + \sigma[a]\left(\mu[a] - 1 - x[a]\right)\phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right) + C \quad (5.29)$$

With the complete antiderivative of integral $\int x[a]^2\phi\left(\frac{x[a]-\mu[a]}{\sigma[a]}\right)$ we can express the expectation of the sum of squares $\mathcal{E}\left(x[a]^2\right)$ in terms of mean $\mu[a]$, standard deviation $\sigma[a]$, and the limits λ and ν only. The expectation is the value of the antiderivative at the upper bound minus the value of the lower bound. If one side is unbounded, the limits of the antiderivative in positive or negative infinity replace the value of the antiderivative. As limit $\lim_{\nu\to\infty}\Phi(\nu)$ is 1 and the limits $\lim_{\lambda\to-\infty}\Phi(\lambda)$, $\lim_{\nu\to\infty}\nu\phi(\nu)$, and $\lim_{\lambda\to-\infty}\nu\phi(\nu)$ are all 0,

the expectation of sum of squares is as follows:

$$\mathcal{E}\left(x[a]^2\right) = \begin{cases} \left(\sigma[a]^2 - \mu[a]^2\right)\Phi\left(\frac{\nu-\mu[a]}{\sigma[a]}\right) \\ +\sigma[a]\left(\mu[a]-1-\nu\right)\phi\left(\frac{\nu-\mu[a]}{\sigma[a]}\right) \text{ if } x[a] \leqq \nu \text{ in term } t_i \\ \\ \left(\sigma[a]^2 - \mu[a]^2\right)\left(\Phi\left(\frac{\nu-\mu[a]}{\sigma[a]}\right) - \Phi\left(\frac{\lambda-\mu[a]}{\sigma[a]}\right)\right) + \\ +\sigma[a]\left(\mu[a]-1-\nu\right)\phi\left(\frac{\nu-\mu[a]}{\sigma[a]}\right) - \\ -\sigma[a]\left(\mu[a]-1-\lambda\right)\phi\left(\frac{\lambda-\mu[a]}{\sigma[a]}\right) \text{ if } \lambda \leqq x[a] \leqq \nu \text{ in term } t_i \\ \\ \left(\sigma[a]^2 - \mu[a]^2\right)\left(1 - \Phi\left(\frac{\lambda-\mu[a]}{\sigma[a]}\right)\right) \\ -\sigma[a]\left(\mu[a]-1-\lambda\right)\phi\left(\frac{\lambda-\mu[a]}{\sigma[a]}\right) \text{ if } \lambda \leqq x[a] \text{ in term } t_i \end{cases} \tag{5.30}$$

With expectations $\mathcal{E}\left(x[a]\right)$ and $\mathcal{E}\left(x[a]^2\right)$ being known, determining the new values of the statistics linear sum and sum of squares of the component cfg_{i**} selected general cluster feature cfg_{**} is possible.

5.4.7 Updating the Borders of the Bounding Rectangle

Although the bounding rectangle of the general cluster feature cfg_* is also a bounding rectangle of the selected cluster feature cfg_{**}, it is a good idea to update the bounding rectangle to fit the limits of the selection predicate—if these are more strict than the limits of the bounding rectangle. Think of a general cluster feature that is selected twice. If we shrink the bounding box according to the selection predicate of the first selection, the chance that the smaller bounding rectangle fulfills the second selection predicate is higher. Consequentially in the average there are less estimates required.

Each term of the selection predicate defines a hypercube in which all tuples fulfilling the predicate must be. As there might be several terms in a selection predicate, the geometric object that is defined by the selection predicate is a compound of hypercubes. It is possible to determine a bounding rectangle that includes this compound of cubes.

We receive the bounding rectangle of the selected general cluster feature cfg_{**} by intersecting the bounding rectangle of the selection predicate and the bounding rectangle of the general cluster feature cfg_* that is about to be selected.

To be more specific, updating the vectors $\vec{bl}_{**}$ and $\vec{tr}_{**}$ of the bounding rectangle of the selected general cluster feature cfg_{**} is done as follows:

1. Determine the minimum $\underline{m}[a]$ and the maximum $\overline{m}[a]$ of the constants in the terms of the selection predicate that are the lower limit $\underline{m}[a]$ or the upper limit $\overline{m}[a]$ of an attribute a of the general cluster feature.

   If there are two terms in the selection predicate, where one term has a lower limit of an attribute and another term that has no lower limit in the

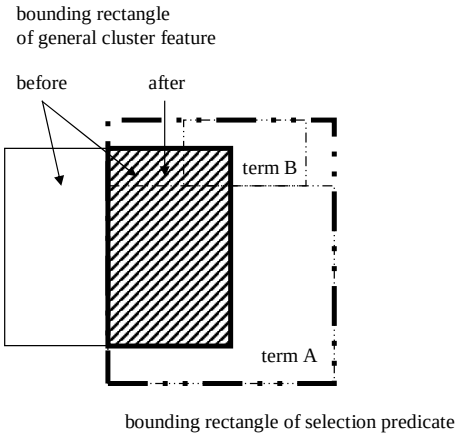


Figure 5.13: updating a bounding rectangle of a general cluster feature with the bounding rectangle of the selection predicate

same attribute, then set the minimum $\underline{m}[a]$ to the minimum value $bl[a]_*$ of the bounding rectangle in the dimension that corresponds with attribute a. This is necessary to receive a valid bounding rectangle. Otherwise, it can happen that the updated bounding rectangle is constrained too much.

Proceed in the same way with maximum $\overline{m}[a]$ when there is a term without an upper bound in that attribute.

2. If the minimum $\underline{m}[a]$ is greater than the minimum value $\vec{bl}[a]_*$ of the bounding rectangle in the dimension that corresponds with attribute a, then the minimum value $\vec{bl}[a]_{**}$ of the new general cluster feature is the minimum of the lower limits of that attribute, i.e. $\vec{bl}[a]_{**} = \underline{m}[a]$. Otherwise, old and new value are identical.

3. If the maximum $\overline{m}[a]$ is less than the maximum value $\vec{tr}[a]_*$ of the bounding rectangle in the dimension that corresponds with attribute a, then the maximum value $\vec{tr}[a]_{**}$ of the new general cluster feature is the maximum of the upper limits of that attribute, i.e. $\vec{tr}[a]_{**} = \overline{m}[a]$. Otherwise, old and new value are identical.

Note that it is unnecessary to determine the bounding rectangle of individual components of a general cluster feature.

5.5 Projecting a General Cluster Feature Tree

Projection of a general cluster feature simplifies the general cluster feature and leaves only those items needed for a given analysis to make the clustering of the third phase faster. Storing the sum of squares in a general cluster feature attribute by attribute is necessary to tell which attribute contributes how much

to the sum of squares. Yet, storing the sum of squares attribute by attribute requires additional time to re-compute the total sum of squares each time the algorithm of the third phase of CHAD needs it. Thus, it is useful to use cluster features instead of general cluster features. Additionally, cluster features need less space than general cluster features. Hence, more cluster features than general cluster features fit a page in the main memory—making the algorithm a bit faster and saving resources.

Thus, we define a projection method π which prunes a general cluster feature to receive a cluster feature that contains only the information needed for a given analysis, as shown in Definition 5.4.

Definition 5.4 *The projection of a general cluster feature* $cfg = (N, \vec{ls}, \vec{ss}, \vec{bl}, \vec{tr})$ *with a set of attributes* $A_a \subset A$ *is a function* $\pi : CFG \times A \rightarrow CF$ *that returns a cluster feature* $cf = (N', \vec{ls}', ss')$ *valid in a given analysis in which attributes* $A_a \subset A^{av}$ *are the relevant attributes, where:*

$$N' = N \tag{5.31}$$

$$\forall a \in A_a : \vec{ls}'[a] = \vec{ls}[a] \tag{5.32}$$

$$ss' = \sum_{a \in A_a} \vec{ss}[a] \tag{5.33}$$

Projection of attributes removes those attributes that are not in the set of selected attributes of a given cluster analysis. If not all attributes are projected, the number of attributes of the projected cluster feature is smaller than the number of attributes of the general cluster feature.

Yet, projection does not alter the number of tuples. Thus, a projected cluster feature and the general cluster feature that was projected to generate it contain the same number of tuples. Hence, the count of tuples N of a general cluster feature and its projected cluster feature are identical. For the same reason, the vector that stores the linear sum of tuples of the projected cluster feature has the same elements as the vector that stores the linear sum of tuples of the the general cluster feature. Yet, the vector of the general cluster features has additional components which store the values of those attributes which have been excluded from projection.

Two methods of projection are possible. First, the entire tree can be projected. Second, a single general cluster feature can be projected when it is needed. In [26] we project tuples on demand. But as selection is better used once for the entire general cluster feature tree, in this approach we favour projecting each node after it has been successfully selected.

5.6 Transforming General Cluster Features

Transforming data is often necessary to make attributes comparable. Clustering algorithm require distance functions to express dissimilarity between two tuples. The best distance function is application-dependant.

For instance, normalisation is useful to compare attributes that have no common distance function. The air temperature in the El Niño data set varies between 18 and 30 degrees Celsius while the geographical longitude has a range of ±180 degrees. This example requires transformation for two reasons. First, as longitude has a broader range it would dominate clustering if both attribute remain un-normalised. Second, the longitude values 180 and −180 denote the same location.

In contrast to that, attributes that have a common distance function should not be normalised. If we omit the problem with longitude values in the vicinity of 180 degrees west and east, respectively, longitude and latitude have a common distance function. It is arbitrary whether we move some degrees in North-South or East-West direction. If we normalise longitude and latitude we favour the attribute latitude because longitude has a a broader range as latitude, i.e. 360 degrees in contrast to 180 degrees. That means that one degree difference in North-South direction is equal to two degrees in East-West direction.

Yet, as the first phase of CHAD does not know the meaning of an attribute, it cannot automatically determine whether an attribute should be normalised or not. We omit polling the user for further details about pre-processed data because this would contradict the basic principle of anticipatory data mining which is to pre-process all kinds of intermediate result in the first phase without user interaction. In other words, the person that initiates pre-processing should not have to be an expert.

Thus, the standard method of processing attributes is to z-normalise each attribute in phase 1 and to undo normalisation in phase 2—if normalisation was erroneously applied in phase 1. Z-normalisation is a transformation that replaces each attribute value with a new value such that the expectation value of the new values is 0 and their standard deviation is 1. CHAD determines mean $\mu[a]$ and deviation $\sigma[a]$ of each attribute and determines the z-normalised value as

$$z := \frac{x[a] - \mu[a]}{\sigma[a]}. \tag{5.34}$$

The general cluster feature of the root node of the general cluster feature tree has all statistics that are necessary to determine mean and deviation of each attribute.

As for some attributes normalisation is a bad choice, there must be means to remedy normalisation. For this reason CHAD has transformation operations to transform general cluster features to receive the general cluster features one would have received by transforming the data and determining the statistics of the general cluster features using the so-transformed data.

Undoing z-normalisation is the most-common application of transformation but transformation is also necessary for deriving new attributes from existing ones, see section 5.7 for details of attribute derivation.

Another side-effect of transformation is that re-scaling attributes is easy. Determining mean and deviation of an attribute would require a scan of the table. As both values must be known before constructing a general cluster feature

tree, this would require two scans in total. Yet, that would make incrementally updating the general cluster feature tree impossible. Thus, CHAD uses a different strategy to build the cluster feature tree and determine mean and deviation at the same time: CHAD scans a small part of the table and determines mean and deviation of this part. With this initial mean and deviation CHAD normalises tuples and builds a tree of general cluster features. At regular intervals, it re-determines mean and deviation. If old mean and new mean or old and new deviation vary significantly, CHAD transforms the tree according to the new values of mean and deviation and continues inserting tuples—now with updated mean and deviation.

The remainder of this section is organised as follows: Subsection 5.6.1 describes linear transformation in detail, as they are an essential part of CHAD. Subsection 5.6.2 shows the more general approach to handle arbitrary functions.

5.6.1 Linear Transformation

CHAD needs a linear transformation function to re-scale attributes and to derive new attributes.

Definition 5.5 *Given a data set D with d dimensions, a set of general cluster features CFG representing D, a multiplicity vector $\vec{m} \in \mathbb{R}^d$, and a constant vector $\vec{c} \in \mathbb{R}$. Then, the linear transformation of a general cluster feature $cfg \in CFG$ is a function $(CFG \times \mathbb{R}^d \times \mathbb{R}^d) \rightarrow CFG$ with a multiplicity vector $\vec{m} \in R^d$ and a constant transformation vector $\vec{c} \in R^d$ as parameters that returns a transformed general cluster feature $\widehat{cfg} \in CFG$ the statistics of which are those statistics one would receive by transforming each tuple $\vec{x}$ that is represented by the general cluster feature cfg according the linear transformation*

$$\hat{\vec{x}} = \mathbf{M}\vec{x} + \vec{c}, \tag{5.35}$$

where the matrix $\mathbf{M}$ is the solution of the following equation:

$$\mathbf{M}(1, \ldots, 1)^T = \vec{m}. \tag{5.36}$$

The remainder of this subsection describes how to use the definition of the linear transformation function to determine the new values of a linearly transformed general cluster feature $\widehat{cfg} = \left(\hat{N}, \hat{\vec{ls}}, \hat{ss}, \hat{\vec{bl}}, \hat{\vec{tr}}\right)$.

Matrix $\mathbf{M}$ in the definition is the result of Equation 5.36. It is a matrix where all values except the values on the diagonal line from the top left value are zero. Each value on the diagonal line is a coefficient of a single attribute, as shown below:

$$\mathbf{M} = \begin{pmatrix} m[a_1] & 0 & \cdots & 0 \\ 0 & m[a_2] & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & m[a_d] \end{pmatrix}$$

The number of tuples remains the same before and after transformation because transformation influences only values but not the count of tuples. Thus, the counts of tuples N and $\hat{N}$ are identical.

$$\hat{N} = N \tag{5.37}$$

The linear transformation transforms all tuples of a general cluster feature in the same way, which means that each attribute value $x[a]$ of each tuple $\vec{x}$ that is represented by the general cluster feature is transformed into a new attribute value $\hat{x}[a]$ as follows:

$$\hat{x}[a] = m[a]x[a] + c[a].$$

As bottom left vector $\vec{bl}$ and top right vector $\vec{tr}$ are points transforming them happens in the same way as transforming tuples. Thus, equations 5.38, and 5.39, respectively, describe how to compute the transformed values of the vectors of the bounding rectangle of a transformed general cluster feature.

$$\hat{bl}[a] = m[a]bl[a] + c[a] \tag{5.38}$$

$$\hat{tr}[a] = m[a]tr[a] + c[a] \tag{5.39}$$

The linear sum of a general cluster feature comprises an aggregate of several tuples. To be more specific, it is the sum of N tuples $\vec{x}$ or $\hat{\vec{x}}$, respectively, where $\vec{x}$ denotes a tuple of the original general cluster feature and $\hat{\vec{x}}$ a tuple of the transformed general cluster feature.

With simple algebraic transformation we receive the expression that determines the new transformed value of the linear sum of a transformed general cluster feature as follows:

$$\hat{ls}[a] = \sum_{\hat{\vec{x}}} \hat{x}[a] = m[a] \sum_{\vec{x}} x[a] + \sum_{\vec{x}} c[a] = m[a]ls[a] + Nc[a].$$

$$\hat{ls}[a] = m[a]ls[a] + Nc[a] \tag{5.40}$$

Analogously, we determine the transformed value of sum of squares $\hat{ss}[a]$ of attribute a of the transformed general cluster feature as follows.

$$\begin{aligned}
\hat{ss}[a] &= \sum_{\hat{\vec{x}}} \hat{x}[a]^2 = \\
&= \sum_{\vec{x}} \left(m[a]x[a] + c[a]\right)^2 = \\
&= \sum_{\vec{x}} \left(m[a]^2x[a]^2 + 2c[a]m[a]x[a] + c[a]^2\right) = \\
&= m[a]^2 \underbrace{\sum_{\vec{x}} x[a]^2}_{ss[a]} + 2c[a]m[a] \underbrace{\sum_{\vec{x}} x[a]}_{ls[a]} + \underbrace{\sum_{\vec{x}} c[a]^2}_{Nc[a]^2}.
\end{aligned}$$

Summarising, we receive the new transformed value $\hat{ss}[a]$ of sum of squares of an attribute a of a transformed general cluster feature as follows:

$$\hat{ss}[a] = m[a]^2ss[a] + 2c[a]m[a]ls[a] + Nc[a]^2 \tag{5.41}$$

5.6.2 Non-linear Transformation

This subsection introduces how to extend the linear transformation function when the needed transformation is non-linear. Non-linear transformation is needed when deriving new attributes to predict non-linear trends such as periodic trends or above-linear trends. We illustrate non-linear transformation with the following example: Assume there is a seasonal trend in the number of sold books which is known to be best approximated by the function $g : \hat{y} = y_0 \sin(\omega t)$ where $\hat{y}$ is the approximated number of sold books per day, y_0 is the average number of sold books per day, t the number of days since the first day of a year, and ω a constant that denotes the periodicity of the saisonal trend. If an analyst wants to classify whether a marketing event was a good or a bad event, then the analyst has to consider saisonal trends, too. To be more specific, the analyst must consider the seasonal trend to correct its influence on the classifier. Therefore, the analyst tests if current number of sold books y exceeds the estimated number of sold books $\hat{y}$. In other words, the analyst defines a new attribute by combing a non-linearly transformed attribute with another attribute.

When transforming tuples linearly, the difference vector between the point representing an original tuple and the point representing the transformed tuple of this tuple is constant for all tuples.

When transforming tuples non-linearly, the difference vector between original point and transformed point is no longer constant. Consequentially, it might be different for each tuple. Moreover, it is no longer possible to compute aggregated features such as sum or average of a transformed general cluster feature directly using the according values of the original general cluster feature. Hence, one would have to transform each tuple individually and compute the aggregate function of the transformed values. Yet, there are no tuples stored in a general cluster feature.

As the tuples of a general cluster feature are not stored in the general cluster feature, a function that transforms general cluster features non-linearly must approximate the values of a transformed general cluster feature using the information kept in the original general cluster feature.

Assume that function $f : \mathbb{R}^d \to \mathbb{R}^d$ is the non-linear function which transforms a point representing a tuple into another point representing the transformed tuple of the original tuple. We will further reference it as non-linear transformation function of tuples.

Definition 5.6 *The non-linear transformation h defined for a general cluster feature $cfg = (N, \vec{ls}, \vec{ss}, \vec{bl}, \vec{tr}) \in CFG$ with attributes A is a function $CFG \to CFG$ with a non-linear transformation function of tuples $f : \mathbb{R} \to \mathbb{R}$ that returns a transformed general cluster feature $\widehat{cfg} = (\hat{N}, \hat{\vec{ls}}, \hat{\vec{ss}}, \hat{\vec{bl}}, \hat{\vec{tr}}) \in CFG$ the statistics of which are the expectation values of those statistics one would receive by transforming each tuple $\vec{x}$ using function f. Namely, each component of a general cluster feature is computed as follows:*

$$\hat{N} = N. \tag{5.42}$$

$$\forall a \in A : \hat{ls}[a] = N\mathcal{E}\left(f\left(x[a]\right)\right). \tag{5.43}$$

$$\forall a \in A : \hat{ss}[a] = N\mathcal{E}\left(f\left(x[a]\right) f\left(x[a]\right)\right). \tag{5.44}$$

$$\forall a \in A : \hat{bl}[a] = \min_{x[a] \in [bl[a]; tr[a]]} f\left(x[a]\right). \tag{5.45}$$

$$\forall a \in A : \hat{tr}[a] = \max_{x[a] \in [bl[a]; tr[a]]} f\left(x_{a_v}\right). \tag{5.46}$$

The expectation value of a function over an attribute is the result of the integral

$$\mathcal{E}\left(f\left(x[a]\right)\right) = \int_{-\infty}^{\infty} f\left(x[a]\right) p\left(x[a]\right) dx[a],$$

if the attribute is continuously scaled, or the result of the sum

$$\mathcal{E}\left(f\left(x[a]\right)\right) = \sum_{x[a]} f\left(x[a]\right) P\left(x[a]\right),$$

if the attribute is discretely scaled, where p and P denote the probability density function of the attribute values $x[a]$ of attribute a and the probability of each value $x[a]$ of attribute a, respectively. Yet, the probability density function can also approximate the sum of probabilities. This approximation is useful when there are many distinct values of that attribute.

The expectation value of a variable determines the average of that variable. Hence, expectation value of the term $f\left(x[a]\right)$ is the average transformed value, not the expected linear sum. By multiplying this average value and the count of tuples in the general cluster feature, we receive the expected transformed linear sum. Analogously, we determine the expected transformed sum of squares.

In analogy to the approximation used for the selection operation, we approximate the distribution of tuples within a general cluster feature with a normal-distributed variable. In other words, we interpret the attribute value $x[a]$ as statistical variable which is distributed normally with mean $\mu[a]$ and standard deviation $\sigma[a]$, i.e.

$$x[a] \sim N\left(\mu[a], \sigma[a]\right).$$

In order to circumvent the problem of conflicting distributions, cf. the discussion on page 142, we use this approximation on leaf node level, only. By approximating the unknown probability density with normal distribution the previously unknown probability density function p has now a known pendant, namely the Gaussian probability density function ϕ. Thus, the computation of the expectation value of linear sum and sum of squares is computable when there exists an anti-derivative for the terms shown in the equations below:

$$\mathcal{E}\left(f(x[a])\right) = \int_{bl[a]}^{tr[a]} f(x[a])\phi\left(\frac{x[a] - \mu[a]}{\sigma[a]}\right) dx[a] \tag{5.47}$$

$$\mathcal{E}\left(\left(f(x[a])\right)^2\right) = \int_{bl[a]}^{tr[a]} \left(f(x[a])\right)^2 \phi\left(\frac{x[a] - \mu[a]}{\sigma[a]}\right) dx[a] \tag{5.48}$$

Re-consider our example at the beginning of this subsection. Function g transforms only attribute values of attribute t. Thus, all other attribute values $x[a]$ are transformed according to the implicit function $\hat{x}[a] = x[a]$, i.e. they remain unchanged. Assume there is a general cluster feature with number of tuples $N = 100$, linear sum $ls[t] = 2500$, sum of squares $ss[t] = 100000$, minimum value $bl[t] = 0$, and maximum value $tr[t] = 60$. Due to their irrelevance, the values of other attributes than attribute t are omitted here. Further assume, that the average number of sold books y_0 per day is 1000 while constant ω is $\frac{2\pi}{365}$ which means that one full season lasts exactly one year. We determine the mean $\mu[t]$ as quotient of linear sum and number of tuples $\mu[t] = ls[t]/N = 2500/100 = 25$ and the standard deviation $\sigma[t]$ using the transformation law of variances as follows $\sigma[t] = \sqrt{ss[t]/N - ls[t]^2/N^2} = \sqrt{100000/100 - 2500^2/100^2} = 5\sqrt{15} \approx 19$. With mean and deviation determined, we compute the transformed values of linear sum and sum of squares using their expectation values, i.e. $\hat{ls}[t] = N\mathcal{E}\left(g(x[t])\right) = N\int_{bl[t]}^{tr[t]} g(x[t])\phi\left(\frac{x[t]-\mu[t]}{\sigma[t]}\right)dx[t]$ and $\hat{ss}[t] = N\mathcal{E}\left(g(x[t])^2\right) = N\int_{bl[t]}^{tr[t]} g(x[t])^2\phi\left(\frac{x[t]-\mu[t]}{\sigma[t]}\right)dx[t]$, respectively. We used MathWorld's antiderivative finder to find the complex antiderivatives of the integrals of both expectation values. The antiderivative of the expectation value of $g(x[t])$ is

$$-\frac{y_0}{2}\sqrt{\frac{\pi}{2e}}\left((-i)\text{erf}\left(\frac{i+x[t]}{\sqrt{2}}\right)+\text{erfi}\left(\frac{1+ix[t]}{\sqrt{2}}\right)\right)+C.$$

Using this antiderivative, we receive the result of the integral within the limits $bl[t]$ and $tr[t]$ which we need to determine the expectation value tuples.

$$\int_{bl[t]}^{tr[t]} g(x[t])\phi\left(\frac{x[t]-\mu[t]}{\sigma[t]}\right)dx[t]$$

$$= -\frac{y_0}{2}\sqrt{\frac{\pi}{2e}}\left((-i)\quad\underbrace{\text{erf}\left(\frac{i+tr[t]}{\sqrt{2}}\right)}_{\approx(0.828175+0.356354*i)}\quad+\underbrace{\text{erfi}\left(\frac{1+tr[t]i}{\sqrt{2}}\right)}_{\approx0.651514+0.818215*i}\right)$$

$$+\frac{y_0}{2}\sqrt{\frac{\pi}{2e}}\left((-i)\underbrace{\text{erf}\left(\frac{i+bl[t]}{\sqrt{2}}\right)}_{\approx0.953438i}+\underbrace{\text{erfi}\left(\frac{1+bl[t]i}{\sqrt{2}}\right)}_{\approx0.953438}\right)$$

$$\approx y_0\sqrt{\frac{\pi}{8e}}((-1)(0.356354+0.651514)+(2*0.9534382))$$

$$\approx 0.34y_0$$

Thus, we receive the transformed linear sum as $\hat{ls}[t] = 0.34y_0N = 34'000$. As computing sum of squares happens analogously we omit to discuss it here.

Minimum $\hat{bl}[t]$ and maximum $\hat{tr}[t]$ of the transformed general cluster feature are minimum and maximum of the transformed attribute values within the range of the old minimum and maximum values $bl[t]$ and $tr[t]$. In our example, function g is monotonously increasing within $bl[t]$ and $tr[t]$. Hence, the transformed minimum is $g(bl[t])$ while the transformed maximum is $g(tr[t])$. When inserting the values of $bl[t]$ and $tr[t]$, we receive $\hat{bl}[t] = 0$ and $\hat{tr} = y_0 \sin \frac{120\pi}{365} \approx 0.86 y_0 = 860$.

5.7 Deriving New Attributes

This section describes how the task of deriving new attributes can be accomplished by applying a combination of selection and transformation.

Sometimes an analysis needs values of objects which are not stored in a database but which can be computed from the attribute values of the tuples representing those objects. For instance, there is no attribute in the fact table of the running example storing the total price of a sales transaction. Yet, one can easily compute the total price using the attributes of the fact table.

Let attribute $a \notin A^{av}$ denote a new attribute which we want to add to the set of existing attributes. Further, let attributes $a' \in A$ and $a'' \in A$ be any two existing attributes the statistics of which are already elements of the general cluster feature into which we want to add the new attribute a.

Assume that function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ determines the relation between an attribute value $x[a]$ of the new attribute a and the attribute values x_{a_v} and x_{a_w} of the existing attributes a' and a'', respectively, i.e.

$$x[a] = f(x[a'], x[a'']).$$

Defining a function that takes only two attributes as arguments is sufficient for defining new attributes which require several involved attributes for their computation because it is possible to split functions taking more than two arguments into several sub-functions taking only two. For instance, if we need a function $f : R^3 \rightarrow R$ taking three arguments for a given analysis we can use two functions $f_1 : R^2 \rightarrow R$ and $f_2 : R^2 \rightarrow R$ which we combine such that the condition $f_1(x[a_1], f_2(x[a_2], x[a_3])) \equiv f(x[a_1], x[a_2], x[a_3])$ holds.

A general cluster feature cfg that is about to receive an additional attribute a requires linear sum, sum of squares, minimum, and maximum of the values of the new attribute. As adding attributes leaves the count of tuples untouched, count N is the same before and after adding an attribute.

In analogy to transformation function we use the expectation value of linear sum to approximate the value of linear sum of the new attribute, i.e. the value of linear sum $ls[a]$ of the new attribute is approximated as

$$ls[a] = N\mathcal{E}\left(f\left(x[a'], x[a'']\right)\right). \tag{5.49}$$

Multiplying count and expectation value of linear sum is necessary as the expectation value represents only the average value, not the sum of values of the new attribute.

We proceed in the same way for sum of squares of the new attribute.

$$ss[a] = N\mathcal{E}\left(f\left(x[a'], x[a'']\right) f\left(x[a'], x[a'']\right)\right). \tag{5.50}$$

Again, multiplying count and expectation value of sum of squares is necessary because the expectation value represents only the average sum of squares of a single tuple.

Approximating the tuples distribution by normal distribution of attribute values in the dimensions spanned by attributes a' and a'', respectively, enables determining the value of the expectation values of linear sum and sum of squares using the Gaussian probability density function ϕ. As we additionally assume conditional independency of attribute values $x[a']$ and $x[a'']$, we can approximate the joint probability density function of attribute values $x[a']$ and $x[a'']$ with the product of the probability density function of each attribute value $x[a']$ and $x[a'']$, respectively. Using these assumptions, we can determine the term of the expectation value of linear sum of the new attribute by integrating the product of each pair of attribute values $(x[a'], x[a''])$ and the joint probability density function as follows:

$$\mathcal{E}\left(f\left(x[a'], x[a'']\right)\right) =$$
$$= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f\left(x[a'], x[a'']\right)\phi\left(\frac{x[a'] - \mu[a']}{\sigma[a']}\right)\phi\left(\frac{x[a''] - \mu[a'']}{\sigma[a'']}\right)dx[a']\right)dx[a''].$$
$$\tag{5.51}$$

If only a subset of tuples is relevant for the computation of a derived attribute then the selection function ς selects an auxiliary general cluster feature which includes the information about relevant tuples of a general cluster feature to compute the new derived attribute. Assume that a selection predicate determines which tuples are relevant for the new attribute. For a given general cluster feature cfg we use the general cluster feature $cfg' = \varsigma_{\text{selection predicate}}(cfg)$ as auxiliary general cluster feature to predict the values of the new attribute when computing these values for general cluster feature cfg. The steps needed to determine the new values for linear sum and sum of squares are still the same as when there would have been no selection. Yet, mean $\vec{\mu}'$ and standard deviation $\vec{\sigma}'$ of the auxiliary general cluster feature replace mean and deviation of the original general cluster feature cfg. For instance, we determine the linear sum of the new attribute a as follows:

$$ls[a] = N'\mathcal{E}\left(f\left(x[a'], x[a'']\right)\right), \tag{5.52}$$

$$\mathcal{E}\left(f\left(x[a'], x[a'']\right)\right) =$$
$$= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f\left(x[a'], x[a'']\right)\phi\left(\frac{x[a'] - \mu'[a']}{\sigma'[a']}\right)\phi\left(\frac{x[a''] - \mu'[a'']}{\sigma'[a'']}\right)dx[a'']\right)dx[a''].$$
$$\tag{5.53}$$

To illustrate the concept of defining new attributes let us re-consider the pre-processed fact table *sales* grouped by customer and time, i.e. Table 4.5. The schema of this pre-processed fact table includes the attributes *sum units books* and *sum units article*.

Assume that we have an analysis where we are interested to know whether the typical mix of product groups influences the likelihood that a customer will quit one's contract or not. Hence, we need to derive new attributes that represent the difference of product groups. The absolute numbers of items sold of a product group such as sold books or sold articles is irrelevant for this analysis. Thus, assume that we want to derive new attributes *diff* and *quot* that comprise the difference of bought books and bought articles on the one hand and the quotient of both attributes on the other hand. Additionally, we consider only those books as relevant which are scientific monographs. Thus, we receive two functions f_1 and f_2 with

$$\begin{aligned} x[\text{diff}] &= f_1(x[\text{sum units articles}], x[\text{sum units books}]) \\ &= x[\text{sum units articles}] - x[\text{sum units books}] \end{aligned}$$

and

$$\begin{aligned} x[\text{quot}] &= f_2(x[\text{sum units articles}], x[\text{sum units books}]) \\ &= \frac{x[\text{sum units articles}]}{x[\text{sum units books}]}, \end{aligned}$$

respectively.

In the running example the product group *scientific monographs* has product group number 3. Thus, we are interested in only those tuples in which the product group equals 3. Attribute *product group* is categorical although its values are numbers, only. Hence, testing for equality is the only valid operator of comparison.

When determining the new values for bottom left, top right, sum of squares, and linear sum of attributes *quot* and *diff* of a general cluster feature cfg, we first determine the auxiliary general cluster feature $cfg' = \varsigma_{productgroup=3}(cfg)$. Each time we need a statistic concerning the number of sold books, we use the appropriate statistic of the auxiliary general cluster feature. When we need a statistic of any other attribute, we use the appropriate statistic of the original general cluster feature cfg such as the number of tuples in the resulting general cluster feature.

The bottom left value $bl[\text{diff}]$ of the new attribute *diff* is the minimum of function f_1 in the ranges $x[\text{sum units articles}] \in [bl[\text{sum units articles}]; tr[\text{sum units articles}]]$ and $x[\text{sum units books}] \in [bl'[\text{sum units books}]; tr'[\text{sum units books}]]$. In our example the minimum of function f_1 is

$$bl[\text{diff}] = bl[\text{sum units articles}] - tr'[\text{sum units books}].$$

Analogously, we determine the top right tr_{diff} as

$$tr[\text{diff}] = tr[\text{sum units articles}] - bl'[\text{sum units books}].$$

The value of linear sum of general cluster feature cfg in the new attribute *diff* is approximated with the expectation value of function f_1, weighted by the

count of tuples N.

$$ls[\text{diff}] = N\mathcal{E}\left(x[\text{sum units articles}] - \varsigma_{\text{product group = 3}}\left(x[\text{sum units books}]\right)\right)$$

As previously shown in this section, by integrating over all arguments of function f_1 we receive the expectation value we are looking for. If we re-formulate this integral we receive two integrals, the result of which we already know. For keeping expressions short we introduce the variables x and y to substitute the attribute values of sold articles and sold monographs as

$$x := \left(x[\text{sum units articles}] - \mu[\text{sum units articles}]\right)/\sigma[\text{sum units articles}]$$

and

$$y := \frac{x[\text{sum units books}] - \mu\left(\varsigma_{\text{product group = 3}}(x[\text{sum units books}])\right)}{\sigma\left(\varsigma_{\text{product group = 3}}(x[\text{sum units books}])\right)}.$$

Thus, we can re-formulate the expectation value of linear sum of function f_1 as follows:

$$\mathcal{E}\left(x-y\right) = \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty}\left(x-y\right)\phi(x)\phi(y)dx\right)dy =$$

$$= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty}x\phi(x)\phi(y)dx\right)dy - \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty}y\phi(x)\phi(y)dy\right)dx =$$

$$= \int_{-\infty}^{\infty}\left(\phi(y)\underbrace{\int_{-\infty}^{\infty}x\phi(x)dx}_{=\mathcal{E}(x)}dy\right) - \int_{-\infty}^{\infty}\left(\phi(x)\underbrace{\int_{-\infty}^{\infty}y\phi(y)dy}_{=\mathcal{E}(y)}dx\right) =$$

$$= \mathcal{E}\left(x\right)\underbrace{\int_{-\infty}^{\infty}\phi(y)dy}_{=1} - \mathcal{E}\left(y\right)\underbrace{\int_{-\infty}^{\infty}\phi(x)dx}_{=1} =$$

$$= \mathcal{E}\left(x\right) - \mathcal{E}\left(y\right).$$

Thus, the expectation value of function f_1 is the difference of expectation values of attribute values of sold articles and sold monographs. By multiplying this the expectation value and the count of tuples of general cluster feature cfg we receive the linear sum $ls[\text{diff}]$ of attribute *diff*. By expressing linear sum of attribute *sum units articles* and the linear sum of the selected attribute values of attribute *sum units books* as weighted expectation values, we can show that linear sum of attribute *diff* is the difference of linear sums of

$$ls[\text{diff}] = N\mathcal{E}\left(x-y\right) = N\mathcal{E}\left(x\right) - N\mathcal{E}\left(y\right) =$$

$$= ls[\text{sum units articles}] - ls'[\text{sum units books}].$$

We omit discussing the computation of the remaining items of the general cluster feature because it happens in analogy to the above-described computation.

5.8 CHAD Phase 3: Using Parameter-Dependent Summarising Statistics for Partitioning Clustering

The third phase of CHAD computes the final result of a given cluster analysis. Hereby, it uses either specific cluster features or general cluster features which the second phase of CHAD has specifically pre-processed according to the needs of the given cluster analysis.

This section gives an overview of the third phase of CHAD. It presents the general principle of algorithms implementing the third phase of CHAD. It discusses algorithms of other approaches that are able to implement the third phase. Yet, it also discusses the implementation of the third phase of CHAD in the prototype of CHAD.

As this section gives only an introduction into the third phase of CHAD, the succeeding sections will discuss specific details of the prototype implementing it.

5.8.1 General Principle of CHAD's Third Phase

The third phase of CHAD computes the result of a partitioning clustering algorithm such as k-means using a dendrogram of cluster features as replacement of the tuples which are represented by these cluster features.

A partitioning clustering which uses a dendrogram instead of tuples for clustering works in a similar way as the same partitioning clustering algorithm that uses tuples does. The most significant difference is that a clustering algorithm using tuples moves tuples from one partition to another partition, while a clustering algorithm using a dendrogram moves sets of tuples represented by cluster features from one partition to another one.

Yet, using a dendrogram causes specific problems, namely *extension in space* and *level of aggregation*.

cluster features have an extension in space While a partitioning clustering algorithm can unambiguously determine the affiliation of a tuple to a cluster, it might fail to do so for cluster features. A tuple is represented by a point in a vector space. Hence, it has no extension in space—not so cluster features. A cluster feature represents a set of tuples and has an extension in space with the consequence that a part of the area covered by the cluster feature are closer to a specific cluster and another part of the area covered by this cluster feature are closer to another cluster.

selecting the right level of aggregation If tuples are represented by hierarchically organised cluster features, then each tuple is represented by several cluster features. At the most fine-grained level, a tuple is represented by an entry of a specific leaf node. This entry is a cluster feature. Yet, any ancestor node of this leaf node contains an entry which also represents the same tuple as the entry of the leaf node does.

> Thus, an clustering algorithm using a dendrogram has to choose which entry to select for clustering. If it chooses entries close to the root node of the dendrogram then computation becomes faster because there are less entries to consider because each entry represents more tuples than entries of levels below that level do, Yet, the extension of that entries is larger than the extension of lower-level entries. Thus, the afore mentioned problem becomes more significant. Consequentially, an algorithm using a dendrogram has to select entries at the right level of aggregation for balancing speed and correctness of tuples' cluster affiliation.

In addition to the problems mentioned above, an algorithm implementing the third phase of CHAD faces another problem concerning the algorithm's initialisation. The initialisation method should avoid accessing tuples as this would slow down the specific data mining. Hence, traditional methods of initialisation such as clustering a sample to find an initial solution are inefficient because accessing tuples is needed if the sample has not been pre-selected. Yet, we omit pre-selecting samples because algorithms using sufficient statistics have shown superior runtime and quality, confer Section 7.3.

Algorithm 3 shows the working of partitioning clustering algorithms using a dendrogram. It is very similar to the working of partitioning clustering algorithms using tuples. Algorithms using a dendrogram additionally test whether it is sufficient to assign a cluster feature to a given cluster as a whole or not. As this test might fail for a specific cluster feature there exists a function that replaces a too coarse grained cluster feature by a more fine-grained one.

The next subsection discusses existing partitioning clustering algorithms using either a dendrogram or cluster features. It shows how one can use these algorithms to implement the third phase of CHAD.

The next sections present how the implementation of CHAD handles the above-mentioned problems. To be more specific, section 5.9 introduces the so-called *bounding rectangle condition* which is a function that tests if a cluster feature has been correctly assigned to a cluster as a whole. Section 5.10 introduces a method to initialise the third phase of CHAD without accessing tuples.

Note that in a previous version of CHAD there exists another function to test if a cluster feature has been correctly assigned to a cluster as a whole, which was called *probability condition*. The probability condition uses approximation with a set of normal distributions to test the probability of misclassified tuples in a cluster feature. Yet, the probability condition was implemented in a first prototype of CHAD by Dominik Fürst, see [19] for details. Yet, his tests have shown that the probability condition only rarely prevents splitting. This dissertation includes tests using the probability condition. As the probability condition does not limit the number of splits, the clustering algorithm used all leaf nodes for clustering. Therefore, these tests show the same performance as the new version of CHAD does when the bounding rectangle condition is turned of. As we will discuss in Section 5.9, the bounding rectangle condition is error-free. Hence, results are identical when using the bounding rectangle condition or not.

Algorithm 3 Pseudo code of partitioning clustering algorithms using a dendrogram of cluster features

Require: set of tuples represented by a set of cluster features $CF' \subset CF$ organised hierarchically $haschild : CF' \rightarrow 2^{CF'}$, number of clusters k, initial solution Θ_ι

Ensure: set of k features $\Theta = \{\theta_1, \ldots, \theta_k\}$

1: $\Theta \leftarrow \Theta_\iota$
2: $CF'' \leftarrow$ `select_entries_of_initial_level` (CF')
3: $CF_P \leftarrow \{CF_1, \ldots, CF_d\}, \forall i \in \{1, \ldots d\} : CF_i = \{\}$ /* create d initially empty clusters */
4: **for all** $cf \in CF''$ **do**
5: $i \leftarrow$ `determine_index_most_similar_feature`$\left(\frac{cf.\vec{ls}}{cf.N}, \Theta\right)$ /* test similarity of $\Theta_i \in \Theta$ and the centroid of cf */
6: $CF_i \leftarrow CF_i \cup \{cf\}$ /* add cluster feature cf to i-th cluster */
7: **end for**
8: **for all** $\theta_j \in \Theta$ **do**
9: $\theta_j \leftarrow$ `update_solution`(CF_j)
10: **end for**
11: **repeat**
12: **for all** $CF_j \in CF_P$ **do**
13: **for all** $cf \in CF_j$ **do**
14: $i \leftarrow$ `determine_index_most_similar_feature` $\left(\frac{cf.\vec{ls}}{cf.N}, \Theta\right)$
15: perform a test that checks if tuples in cf are correctly assigned to θ_i /* Algorithms using a dendrogram of cluster features for partitioning clustering vary in the implementation of this test. Some algorithms test if the expected number of wrongfully assigned tuples exceeds a given threshold while other algorithms test if it can be guaranteed that there exist no wrongfully assigned tuples */
16: **if** test fails **then**
17: $CF_{temp} \leftarrow cf.haschildren$ /* replace cf by its children */
18: **if** $CF_{temp} \neq \{\}$ /* if there are children */ **then**
19: $CF'' \leftarrow CF''\backslash\{cf\}$ /* remove cf from cluster feature list */
20: $CF_j \leftarrow CF_j\backslash\{cf\}$ /* remove cf from its cluster */
21: $CF'' \leftarrow CF'' \cup CF_{temp}$ /* add children to list of cluster features */
22: $CF_j \leftarrow CF_j \cup CF_{temp}$ /* add children to cf's cluster */
23: **end if**
24: **end if**
25: **if** $i \neq j$ **then**
26: $CF_i \leftarrow CF_i \cup \{cf\}$
27: $CF_j \leftarrow CF_j\backslash\{cf\}$
/* if algorithm uses MacQueen-method then update solution here instead of line 31 */
28: **end if**
29: **end for**
30: **for all** $\theta_j \in \Theta$ **do**
31: $\theta_j \leftarrow$ `update_solution`(CF_j)
32: **end for**
33: **end for**
34: **until** stop criterion is met
35: **return** Θ

5.8.2 Using Existing Approaches to Implement CHAD's Third Phase

Several authors have previously published approaches which use cluster features for computing the results of cluster analyses. There exist approaches which perform an *EM* clustering using cluster features such as [6]. There also exist approaches that implement k-means using cluster features.

As CHAD can produce a cluster feature tree that represents a subset of a data set which satisfies a given selection predicate, all approaches using a cluster features tree can work with the result of CHAD's second phase.

There exist approaches which use lists of cluster features instead of a tree of cluster features. CHAD can produce a list of cluster features for these approaches. To do so, CHAD must select all entries of the cluster feature tree which CHAD has produced in its second phase and add these entries—which are cluster features—into an initially empty list of cluster features.

5.8.3 Implementing CHAD's Third Phase for k-means

This subsection discusses how to implement the generalised pseudo code of an algorithm of CHAD's third phase for k-means as depicted in Algorithm 3. It also serves as introduction for the succeeding sections which discuss special issues of this implementation in detail.

Initialising happens outside of Algorithm 3. Thus, another function must generate an appropriate initial solution Θ' which consists of a set of initial centroids. The experimental prototype of CHAD uses a genetic algorithm to find a set of k points which serve as initial centroids for k-means. This genetic algorithm uses the centroids of the entries of a set of nodes of the cluster feature tree as its initial population. It tests the fitness of individuals by running Algorithm 3 and determining the quality of that run's result. In other words, the genetic algorithm calls the Algorithm 3 to determine a new improved solution.

Section 5.10 introduces the details of the genetic algorithm which generates initial solutions for the third phase of CHAD without accessing tuples.

First, the experimental prototype of CHAD selects the level of the cluster feature tree that contains more than $2k$ entries, where k denotes the number of clusters to be found. The lower limit of $2k$ entries is the limit of choice because the initial clusters contain only a very small number of cluster features at the beginning that way—the smaller the number of cluster features is the faster is the algorithm. Yet, it is not too small that the chance to receive empty clusters is too high.

If the rare phenomenon occurs that a cluster is empty after initially assigning cluster features, the algorithm aborts the current run and signals the calling genetic algorithm that it failed to produce a result. The genetic algorithm penalises the initial solution that created no result by assigning a very low fitness to it.

Except for some extensions we will discuss below, the implementation of Algorithm 3 in the experimental prototype works in the same way as the k-

means variant of MacQueen [46], i.e. it iteratively re-assigns cluster features to clusters and re-computes the centroids of cluster each time after the number of cluster features affiliated with a cluster has changed.

Most tuples of a cluster feature must be nearest to the cluster that minimises the distance of centroid of the cluster and centroid of the cluster feature. Yet, there might be tuples nearer to the centroids of other clusters. As a tuples represented by the same cluster feature might be part of several clusters, CHAD tests if a cluster feature might contain tuples of other clusters but the cluster the centroid of the cluster feature is nearest. If this probability is greater than a given threshold then CHAD replaces the cluster feature by the cluster features of its child nodes—if there are child nodes of this cluster feature. CHAD performs this test each time it tests the affiliation of a cluster feature to clusters.

Section 5.9 presents details of the test whether replacing a cluster feature is necessary or not.

The implementation of Algorithm 3 in the experimental prototype stops when there have been no re-assignments of cluster features and no cluster features have been replaced by their children within the current iteration.

5.9 Testing the Existence of Wrongfully Assigned Tuples with the Bounding Rectangle

The third phase of CHAD needs to test if there might exist tuples in a cluster feature which are closer to another cluster's centroid than to the centroid of the cluster the cluster feature is assigned to. If CHAD cannot guarantee that there are none of these tuples then CHAD replaces the cluster feature with the entries of its child nodes.

We reference the condition that a cluster feature must satisfy that it needs not to be replaced as *bounding rectangle condition* because it uses the bounding rectangle of a cluster feature to test if there could be a tuple within the cluster feature that is wrongfully assigned to the cluster to which the cluster feature is assigned.

If the bounding rectangle condition holds for many cluster features of the cluster feature tree of a given analysis, then the third phase of CHAD works with a small set of cluster features, only. As a consequence of this, the algorithm terminates faster.

This section discusses the bounding rectangle condition in detail.

The bounding rectangle of a cluster feature represents a sub-space of a d-dimensional vector space that envelopes all tuples represented by this cluster feature. Or in other words, there exist no tuples of the cluster feature outside of the cluster feature's bounding rectangle.

The bounding rectangle condition tests if the bounding rectangle of a cluster feature is completely inside the sub-space that represents all tuples of a cluster. As the used clustering algorithm determines the geometric form of this sub-space, we focus our discussion on k-means clustering. Yet, one can apply the

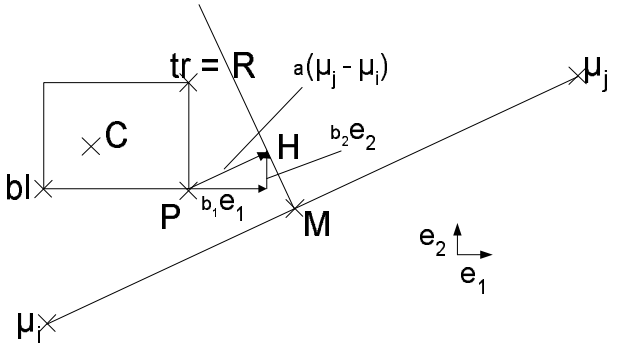


Figure 5.14: Testing if all points in a bounding rectangle are closer to a mean than to the mean of another cluster

same techniques we discuss here on analyses with other clustering algorithms in analogous way.

In a k-means cluster analysis the bisecting hyperplane between a pair of means separates all tuples that are closer to one of these means from tuples that are closer to the other mean. If we would use EM instead of k-means, the sub-space that represents all tuples of a cluster is the sub-space that contains all vectors where the probability density function of the cluster is higher than the probability density function of another cluster.

Thus, if we can show that all tuples of a cluster feature are on the same side of all bisecting hyperplane between each pair of means, we have also shown that all tuples of this cluster feature are closer to the same mean.

If a pair of means $(\vec{\mu}_i, \vec{\mu}_j)$ and a cluster feature are given, then the centroid $\vec{C}$ of the cluster feature is either nearer to one of both means or the centroid has equal distance to both centroids. Assume that C is nearer to mean $\vec{\mu}_i$ Consequentially, there exist tuples that are nearer to mean $\vec{\mu}_i$ than to the other mean$\vec{\mu}_j$. Hence, if there exists only a single tuple in the cluster feature that is nearer to the other mean $\vec{\mu}_j$ then there are tuples in the cluster features that are part of two different clusters—and not the same cluster, as it is required by the bounding rectangle condition.

For testing if a tuple that conflicts with the bounding rectangle condition might exist, we consider the perpendicular of any point in the bounding rectangle on the bisecting hyperplane. Note that a bounding rectangle of a cluster feature cf is a d-dimensional sub-space containing an infinite number of points—yet, only $cf.n$ points represent tuples. Therefore, if there is no point conflicting with the bounding rectangle condition, then none of the cluster feature's tuples can conflict with the bounding rectangle condition.

For testing on which side of the bisecting hyperplane a point $\vec{P}$ is located we drop a perpendicular from point $\vec{P}$ onto the bisecting hyperplane, as illustrated in Figure 5.14. Hereby, let vector $\vec{H}$ denote the intersection point of the

perpendicular and the bisecting hyperplane. The difference vector of mean $\vec{\mu}_i$ and mean $\vec{\mu}_j$, vector $\vec{\mu}_j - \vec{\mu}_i$, is also perpendicular to the bisecting hyperplane. Therefore, the difference vector $\vec{\mu}_j - \vec{\mu}_i$ is parallel to the perpendicular of point $\vec{P}$. Therefore, we can express the perpendicular from point $\vec{P}$ to the bisecting hyperplane as a linear combination

$$\vec{H} - \vec{P} = a(\vec{\mu}_j - \vec{\mu}_i).$$

As difference vector has an orientation, we can use the factor a to test on which side of the bisecting hyperplane point $\vec{P}$ is: If factor a is positive, point $\vec{P}$ is on the same side of the bisecting hyperplane as mean $\vec{\mu}_i$. If it is negative, point $\vec{P}$ is on the side of mean $\vec{\mu}_j$. Finally, if it is zero then point $\vec{P}$ is exactly on the bisecting hyperplane.

If there exists at least one point where the linear combination has a negative factor a, then the bounding rectangle condition is false.

We determine the point that minimises factor a because this point is easy to determine and if this point has a non-negative value of factor a, then all other points in the cluster feature have a non-negative value, too—which means that the bounding rectangle condition is true.

In order to determine the point that minimises factor a we start at any corner of the bounding rectangle and try to find a corner that is connected via an edge with the current corner that has a smaller value of factor a. If we find such a corner, we recursively try to find better corners until there is no better corner.

Each edge of a bounding rectangle is parallel to exactly one of the axis of the d-dimensional vector space that contains all tuples. Let vector $\vec{e}_m$ denote a vector parallel to the m-th axis with normalised length of 1. We will reference this vector as the m-th axis vector. Therefore, depending on the corner we are currently considering, we can move in the m-th dimension either in positive or negative direction of vector $\vec{e}_m$.

Moving from a corner in positive direction in the m-th dimension means to move from the minimum value to the maximum value in this dimension, i.e. a move from value $\vec{bl}[m]$ to value $\vec{tr}[m]$. Contrary, moving in negative direction means to move from maximum to minimum value.

If we also represent the difference vector $\vec{\mu}_j - \vec{\mu}_i$ as a linear combination of all axis vectors, i.e.

$$\vec{\mu}_j - \vec{\mu}_i = \sum_{m=1}^{d} b_m \vec{e}_m,$$

then a move in positive direction of axis vector $\vec{e}_m$ decreases the value of factor a or results in the same value if the coefficient b_m of the m-th dimension is also positive. In a Cartesian vector space, each coefficient b_m is the component of the difference vector in the m-th, dimension, i.e.

$$b_m = (\vec{\mu}_j - \vec{\mu}_i)[m]$$

Therefore, each coefficient b_m determines whether to make a positive or a negative move in the m-th dimension. Hence, the corner $\vec{R}$ that minimises factor a is

$$\vec{R} \in \mathbb{R}^d, \text{ with } \vec{R}[m] = \begin{cases} \vec{tr}[m] \text{ if } b_m \geq 0 \\ \vec{bl}[m] \text{ otherwise.} \end{cases}$$

Once we have determined the corner $\vec{R}$ that minimises factor a we can test if the distance of corner $\vec{R}$ is nearer to $\vec{\mu}_j$ than to $\vec{\mu}_i$. If the condition is false, the bounding rectangle condition is false, too. If the condition is true, we use the same test for other means. If all tests are positive, the bounding rectangle condition holds. Therefore, we summarise the argumentation above and define the bounding rectangle condition as follows.

Definition 5.7 *Given a cluster feature $cf = (n, \vec{ls}, ss, \vec{bl}, \vec{tr})$ and a set of means $M = \{\dots, \vec{\mu}_i, \dots, \vec{\mu}_j, \dots\}$, where μ_i is the mean that is nearest to the cluster feature's centroid $\vec{C} = cf.\vec{ls}/cf.n$. The bounding rectangle condition brc of a cluster feature is a predicate that is true, if all corners of the bounding rectangle are nearer to mean $\vec{\mu}_i$ than to each other mean. In other words, the following condition must hold:*

$$\forall(\vec{\mu}_j, \vec{\mu}_i) \in M \times M \text{ with } \vec{\mu}_i \neq \vec{\mu}_j$$

$$\text{where } \vec{R}_j \in \mathbb{R}^d, \text{ with } \vec{R}[m] = \begin{cases} \vec{tr}[m] \text{ if } (\vec{\mu}_j - \vec{\mu}_i)[m] \geq 0 \\ \vec{bl}[m] \text{ otherwise.} \end{cases} :$$

$$||\vec{R}_j - \vec{\mu}_i|| \leq ||\vec{R}_j - \vec{\mu}_j||$$

5.10 Initialising the Third Phase of CHAD

5.10.1 Third Phase of CHAD is Liable to Local Minima

CHAD's implementation of k-means is very similar in structure to MacQueen's variant of k-means. The major difference is that the implementation of CHAD's third phase moves entire sub-clusters instead of single points between the k main clusters. So, it might be vulnerable to the same shortcomings as k-means is. We will discuss these shortcoming in this subsection before we will discuss different methods to initialise k-means and the effectiveness of theses methods to master the shortcomings in succeeding subsections.

k-means is a hill-climbing optimisation method. Each hill-climbing method starts with an initial solution and searches the local environment of the current solution for a better solution. The method repeats this process until there is no better solution in the local environment of the current solution. Due to this local search approach it is possible that the local environment contains no better solution than the current one while there is a better solution in global scope.

Hence, a drawback of k-means is its liability to the initial clustering. A badly chosen initial clustering can cause k-means to return a local minimum. Due to the complexity of the clustering problem, the optimal solution cannot

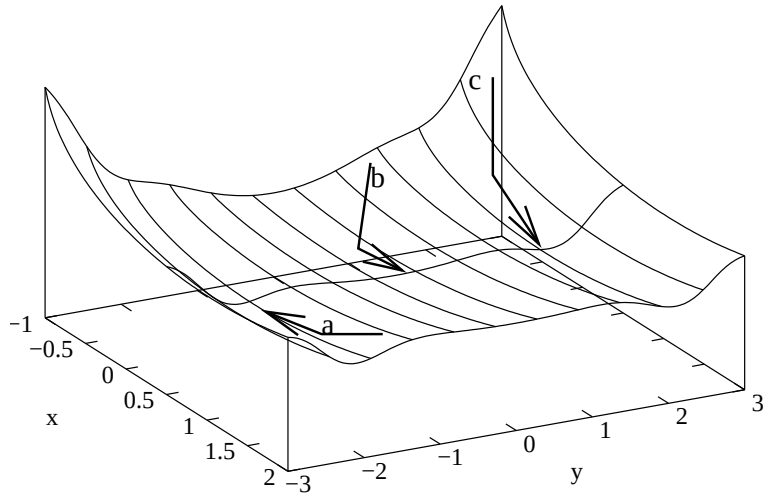


Figure 5.15: finding the optimal clustering for two one-dimensional clusters

be determined efficiently—this is also true for determining an initial clustering that causes k-means to result in the optimum. Hence, heuristical methods try to maximise the probability of finding an optimal solution.

The liability to local minima can be seen in Figure 5.15. This figure shows a fictive clustering problem with two clusters. The according vector space is one-dimensional. So, all potential solutions of this clustering problem can be visualised in a two-dimensional drawing, where a single solution consists of a pair of two variables—in Figure 5.15 denoted as variable x and variable y, respectively. Variable x denotes the mean of the first cluster, while variable y denotes the mean of the second one. The third dimension contains the value of a quality measure. In the case of k-means clustering, the quality measure is the sum of distances. In other words, the optimal clustering is the combination (x, y) which minimises this quality measure. In Figure 5.15 the left-most and the right-most valley are both minimal in this figure. Yet, there is another (local) minimum that have about the same value of total sum of differences.

Figure 5.15 also includes three initial solutions—marked as polylines with arrows—that all result in a final solution after two iterations of k-means. While initial solutions a) and c) result in one of the global minima, solution b) fails to find a globally optimal solution.

The influence of local minima on quality can be greater than the influence of all other influences on quality. Figure 5.16 depicts a test result of a test series we will discuss in detail in Section 7.4. It is a test series which uses sampling for efficient clustering of a data set. However, this figure is a good example of the effect of local minima on quality. Most initial solutions result in a total distance of approximately $1.05 \cdot 10^8$. If we would look at the exact values of total distance we would notice that the total distances of tests vary about 1% around $1.05 \cdot 10^8$

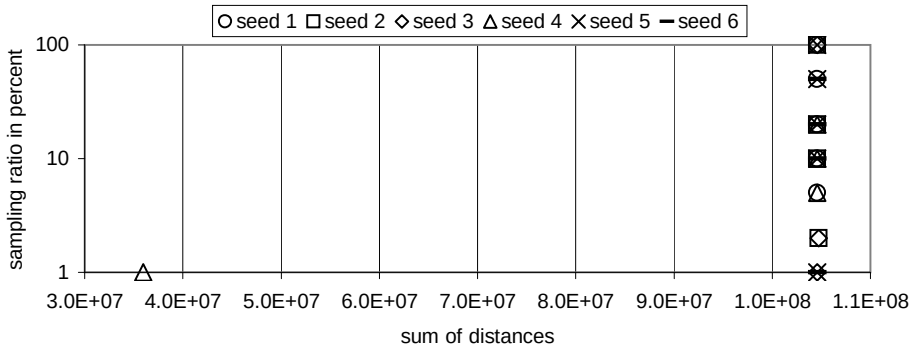


Figure 5.16: El Niño test series with three clusters

in both directions—see Section 7.4 for details. Yet, one solution is significantly better than the other solutions which means that there is one initial solution ending in a different local minimum than the other initial solutions do. The other initial solutions find a solution very close to the same local minimum but they differ due to sampling error. Yet, it is hard to notice the effect of slightly varying results because the effect caused by the different local minima outweighs other sources of error such as sampling error or computational inaccuracy.

Summarising, we state that the method used to initialise k-means, regardless, whether it is a variant using tuples or clustering features, must have a high chance to find an initial solution that ends in the global minimum of total distance. Hence, the following subsections of this section discuss different methods to initialise k-means.

It makes no difference if the used k-means variant uses tuples or clustering features because finding an initial solution can happen outside of k-means. Hence, it is possible to use a sample of tuples to find an initial solution for a k-means algorithm which then uses clustering features to compute a final solution.

However, if the tuples are no longer present to take a sample of them, one needs alternatives to sampling to initialise k-means. Alternative ways of initialisation are also needed when taking a sample would worsen the performance of k-means. Taking a sample means to scan at least the entire index of a fact table to ensure that each tuple has the same chance of being part of the sample.

Preparing a sample for initialising would be a potential solution. Yet, we will show that this is a bad solution because we are unable to judge the quality of results. Therefore, the next subsection discuss specific methods to initialise k-means. A subsection about the method to initialise k-means which the prototype of CHAD uses concludes this section.

5.10.2 Initialising k-means With a Sample

Taking a set of samples of the data to be clustered and performing k-means on each sample is a popular way to initialise k-means. Hence, this subsection discusses this method of initialisation.

Clustering a sample of a set of tuples produces a result which is similar to the result the clustering algorithm would have been produced when it had used the set of tuples. Yet, the results of both methods can differ due to sampling error. In other words, a clustering algorithm using a sample could produce a wrong result because the sample contains more tuples of a specific cluster than it does of another cluster. Consequentially, there is a bias in the result favouring the over-represented cluster.

Yet, clustering a data set which has a small number of tuples is very fast. Hence, clustering a sample which is a small subset of the original data set is faster than clustering the original data set.

Taking a sample also needs time. In order to receive an unbiased sample each tuple has to have the same chance of being selected. To achieve this, one must at least scan the entire index of a table before accessing the chosen tuples by point accesses. If the sample ratio is very high it is more efficient to access tuples in a bulk operation, which means that the entire table is scanned. Yet, if index or table, respectively, are very large then taking a sample needs much time. Moreover, it can take longer than the clustering of the sample as clustering uses data that fits the main memory, only. Hence, the clustering needs no disc accesses.

To improve the speed of taking a sample, pre-selecting a sample that can reside in the main memory could be beneficial because the sample would be present when needed without accessing the disc.

Therefore, we discuss different techniques of taking samples from a fact table, namely sampling tuples, sampling samples, and sampling centroids. Sampling tuples requires disc accesses. Hence, it is slow when the index of the sampled table is reasonably large. Sampling a pre-selected sample requires disc accesses only when selecting the initial sample which is used to take samples. Hence, it needs no additional disc accesses when taking the samples from this sample. Therefore, it is a fast method to take several samples. The final option, sampling centroids, uses the clustering features of a clustering feature tree for sampling. To be more specific, it uses each centroid of a clustering feature as potential element of the sample. As the clustering feature tree resides in main memory, this method is a very fast method.

However, the above-mentioned different techniques of taking a sample also differ in the expected sampling error. Therefore, the following subsections discuss each technique. We will show how each methods influences the expected quality of initial solutions.

Sampling Tuples

If there are several clusters or several dimensions, there exist only heuristics to find the optimal solution which means that k-means either manages to find the globally optimal solution or it finds a locally optimal solution.

Our tests show that the likelihood that a randomly selected initial solution terminates in the global optimum does not increase with increasing sample size when sample size exceeds few thousands of tuples, see Section 7.4 or confer [25]. For instance, Figure 5.16 depicts a test series in which the test with the smallest sample size finds a better solution than the tests having larger sample sizes.

By iteratively taking samples one can increase the probability of finding the global optimum. Assume that probability p denotes the likelihood that an initial solution terminates in the globally optimal solution. Further, assume that probability q denotes the likelihood that an initial solution terminates in a locally optimal solution, only. Obviously, both probabilities sum up to $p+q = 1$. Using these assumptions, the likelihood not to find the global optimum in d attempts with a sample for each attempt is q^d. The term q^d is a geometric series which approximates zero. In other words, with rising number of tries it becomes more and more unlikely to fail each time to find the global optimum.

Hence, it is a good choice to iteratively take small samples and use the sample which returns the best result when used as initial solution of k-means. Yet, taking a sample of tuples requires additional disk accesses which slow down the total performance of an algorithm. Therefore, we will discuss taking samples from a pre-selected sample in the next subsection because once the pre-selected sample has been retrieved there are no additional scans necessary.

Sampling a Sample

In this subsection we show that sampling a sample results in the same estimated distribution of tuples. Hence, the initial solution received by sampling a sample is suitable to find the optimal solution. Yet, we also show that finding this solution only depends on the pre-selected sample—not the samples of this sample.

When taking a sample having the size n from a data set with N tuples that consists of $N - M$ noisy tuples and M non-noisy tuples, the number of non-noisy tuples m in the so-taken sample is a random variable X that is distributed according to a hypergeometric distribution because taking a large sample is a similar random process as drawing random items from a ballot box without putting them back—which is the random process that leads to a hypergeometric distribution.

$$X \sim h(n; N; M)$$

The expectation μ of the number of non-noisy tuples X is given by the expectation of the hypergeometric distribution as

$$\mu = n\frac{M}{N}.$$

Hence, the expected proportion of non-noise in the sample equals the proportion of non-noise $\frac{M}{N}$ in the entire data set, i.e.

$$\frac{\mu}{n} = \frac{nM}{nN} = \frac{M}{N}.$$

In other words, sampling does not change the expected ratio of non-noise or noise, respectively. Too much noise would negatively influence the quality of results. Hence, a sample is prone to noise in the same way as the set of tuples is.

However, the variances are different when a sample is taken from a sample on the one hand or taken from the data set on the other hand. To show this, we take a small sample from the data set and take a sample of the same size from the large sample.

Let the variable m denote the actual number of non-noise in the large sample. Further, let variable r denote the sampling ratio of the large sample. Let variable d denote the common size of both small samples.

When taking a small sample from the entire data set the number of tuples that is non-noisy Y_1 is again distributed according to a hypergeometric distribution—yet, now with an expectation of

$$\mathcal{E}(Y_1) = d\frac{M}{N}$$

and a variance of

$$Var(Y_1) = d\frac{M}{N}\left(1 - \frac{M}{N}\right)\frac{N-d}{N-1}.$$

Contrary, when taking a sample of the large sample, expectation and variance of the number of tuples that are non-noisy Y_2 in the so-taken sample depend on the outcome of the first sample. In other words, the number of tuples of the large sample n and the number of non-noisy tuples in the large sample m replaces the according values of the data set N and M, respectively. Hence, random variable Y_2 has an expectation of

$$\mathcal{E}(Y_2) = d\frac{m}{n}$$

and a variance of

$$Var(Y_2) = d\frac{m}{n}\left(1 - \frac{m}{n}\right)\frac{n-d}{n-1}.$$

Let us compare the quotient of both variances. Further we assume that the outcome of number of non-noisy tuples in the large sample has the average number of non-noisy tuples, i.e. the ratio of non-noisy tuples is identical $\frac{M}{N} = \frac{m}{n}$ in the large sample and the data set.

Under the assumption of identical ration of non-noisy tuples, the quotient of variances shows that the variance of the sample of the total data set is higher:

$$\frac{d\frac{m}{n}\left(1 - \frac{m}{n}\right)\frac{n-d}{n-1}}{d\frac{M}{N}\left(1 - \frac{M}{N}\right)\frac{N-d}{N-1}}$$

$$\overset{\frac{M}{N}=\frac{m}{n}}{\Rightarrow} \frac{d\frac{M}{N}\left(1-\frac{M}{N}\right)\frac{n-d}{n-1}}{d\frac{M}{N}\left(1-\frac{M}{N}\right)\frac{N-d}{N-1}} = \frac{(n-d)(N-1)}{(N-d)(n-1)} < 1 \text{ if } d > 1 \wedge N > n.$$

Having higher variance means that it is more likely to have extreme samples when using the entire data set for sampling. Thus, we might receive samples with many noisy tuples as well as samples with almost only non-noisy tuples. If we take several samples there is a good chance to receive at least in one case a sample that has only a minimal number of noisy data—which is good for classification and clustering as shown in the experiments section.

Additionally, sampling a sample changes the chance that a specific tuple is selected. This means that there is an additional but unintended bias. As we will discuss in the experiments section, a bias can be beneficial when it reduces the number of noise in the used data set. However, an unintended bias can result in bad results—we will also discuss this in the experiments section when discussing experiments with a bias that increases noise.

Assume that one has taken a large sample once and takes several small samples from this sample later on. Then, the likelihood that a tuple is part of one of these final samples depends on the outcome of the initial sampling. Even if the chance that a tuple is selected when taking a small sample from the data set is low, the chance that this tuple will be selected at least once in series of several samples is comparably high. If the chance of being selected in a sample is p then the probability to be selected at least once in a series of k samples is $1-(1-p)^k$. The limit of this likelihood is 1. Therefore, there exists a number of samples k where the term $1-(1-p)^k$ is higher than the limit of the according probability that a tuple is never selected when taking a sample from a sample.

The necessity of a tuple determines whether it is sufficient not to select that tuple or not. If the tuple fulfills a special condition such as its attribute values assume the extreme values in a specific attribute then the tuple is necessary to be part of a sample when an analysis needs that extreme values.

Consequentially, using samples of a sample for initialising k-means is inferior in terms of quality to sampling the total data set because it is subject to unintended bias and it has lower deviation which means that it less extensively explores the solution space compared to sampling the total data set.

5.10.3 Determining the Quality of a Solution

After a solution has been determined it is necessary to know the quality of that solution—regardless, whether this solution is an initial solution or a final solution. Yet, if a sample has been used to find that solution, one cannot use this sample to determine the quality of the solution. Otherwise, it would be the same problem as is testing a hypothesis with the same data set that was used to derive it.

Hence, this section describes how to judge the quality of a solution

1. if a sample has been used to find the solution, or

2. if cluster features, i.e. clustering features or general cluster features, have been used to find the solution.

We will show that testing quality with general cluster features is superior to the other mentioned methods because general cluster features contain all necessary information to derive solutions and to test these solutions' quality. Moreover, they need no external information to judge the quality of solutions.

Determining the Quality of a Sample

As mentioned above, a sample is insufficient to judge the quality of solutions that have been generated using this sample. The sampling error that negatively influenced the found solution occurs in the same way when determining the quality of this solution. Consequentially, using the sample to determine the sample's solution's quality could not detect sampling error and would suggest good quality where there is no good quality. It is the same effect we have discussed as *over-fitting* in the introductory section about classification.

Therefore, a second sample is needed to judge the quality of a solution which has been computed from a given sample. The sample used for testing quality and the sample used for finding a solution must be taken independently from each other to prevent that the error of one sample depends from the error of the other sample.

Moreover, the above-mentioned condition of independency of samples fails to hold for samples taken from the same sample. By taking two samples from a sample one might determine a solution and test the quality of this solution. Yet, one can only determine whether this solution is a good solution with respect to the sample that has been used to take both samples. However, the result of this test cannot be used to predict the quality of the solution with respect to the entire data set because it is unknown whether the the initial sample is a good sample or not.

The lack of verifying quality of a sample without taking another sample is an immanent problem of taking samples from a sample. It is a problem which one can avoid by using anticipatory data mining because the general cluster features of CHAD allow testing a solution without re-generating the general cluster feature tree—re-generating the cluster feature tree would be the analogous operation to taking another sample.

Yet, testing the quality of a solution with another sample has also disadvantages in comparison with testing the quality of a solution with general cluster features: Testing the quality of a solution with another sample can estimate the error caused by the first sample, i.e. one tests if the likelihood that the assumption that the sample's solution is also a good solution of the entire data set is true exceeds a given error threshold. Yet, the relation between this test and the real facts in the data set are stochastic which means that the deviation of the estimated relations from the real relations is unlikely but potentially unbounded.

In contrast to that, testing the quality of a solution with general cluster

features can give deterministic bounds of potential error, which we will discuss next.

Determining the Quality With Cluster Features

Unlike a sample, a general cluster feature tree and a clustering feature tree, respectively, comprises a compressed representation of the entire original data set. We use the term *cluster feature* when we intend to describe common features of clustering features and general cluster features. If some feature apply only to either clustering features or general cluster features we use the terms clustering feature or general cluster feature, respectively.

In contrast to cluster features, a sample is just an excerpt of this data set. Therefore, when computing an item using cluster features, this computation includes all tuples—yet, in compressed form. Moreover, there can be no sampling error when using cluster features because cluster features are no samples but compressed representatives of a data set.

As all cluster features of the same level in a cluster feature tree represent all tuples in a data set, a statistic we compute using these cluster features is no estimated value of the entire data set. This statistic is the appropriate value of the data set when tuples of different clusters are not mixed in the same cluster feature—we will discuss mixture of clusters in the next paragraph. This condition is especially true for quality measures of a given cluster analysis. In other words, using cluster features can find the correct value of a quality measure while using a sample can only find an estimated value of a quality measure.

When there exist tuples in a cluster feature which are not all part of the same cluster, computed quality measures might be erroneous. Yet, using general cluster features make it evident when potential misassignment of tuples occurs because the bounding rectangle gives deterministic information about the extension of tuples in a general cluster feature. Moreover, it is possible to tell when all tuples are correctly assigned to a specific cluster. Only those general cluster features potentially having misassigned tuples in them can cause miscomputation of quality measures. Yet, one can use the elements stored in a general cluster feature to define a limit of the error made in worst case, or to compute the estimated error. The remainder of this section presents how to compute a *limit of error* and the *expected error*.

The total squared distance TD^2 is a quality measure for k-means clusters. It measures the sum of the sums of the squared distances of each tuple $\vec{x}$ to its nearest centroid $\vec{\mu}$ of a cluster. Therefore, the total squared distance TD_i^2 of a single cluster C with centroid $\vec{\mu}$ is

$$TD_i = \sum_{\vec{x} \in C} (\vec{x} - \vec{\mu})^2 = \underbrace{\sum_{\vec{x} \in C} \vec{x}^2}_{=cf.ss} - \sum_{\vec{x} \in C} (\underbrace{\vec{\mu}}_{=cf.\vec{ls}/cf.N})^2$$

By adding all cluster features that contain tuples of the i-th cluster C, we receive a cluster feature cf that represents all tuples of this cluster. By doing so, we

can express the total squared distance TD_i^2 of the i-th cluster in terms of the elements of this cluster feature as follows:

$$TD_i^2 = cf.ss - cf.N \cdot cf.\vec{ls}^2/cf.N^2 = cf.ss - cf.\vec{ls}^2/cf.N.$$

The total squared distance TD^2 of a clustering is the sum of all total squared distances TD_i^2 of all clusters, i.e.

$$TD^2 = \sum_i TD_i^2.$$

In other words, if we add all cluster features that contain all tuples of a cluster then we receive a single cluster feature for each cluster which contains all information necessary to compute the quality measure *total squared distance*[3].

Yet, if some of the cluster features we added to receive a cluster feature per cluster have tuples of other clusters within them, the summed up cluster feature is erroneous. Due to the bounding box of a general cluster feature we can check if there might exist tuples of other clusters in a general cluster feature. Therefore, we split the set of general cluster features into two sets by testing each general cluster feature if it might contain tuples of other clusters. The first set contains all general cluster features containing tuples of a single cluster only. The second cluster contains the remaining general cluster features.

Due to the bounding box of a general cluster feature we can also test to which clusters its tuples might belong to. Therefore, we can receive a general cluster feature of a cluster the tuples of which are definitely part of the cluster and a set of cluster features the tuples of which might be part of that cluster.

The lower bound of the total squared distance TD_i^2 of the i-th cluster is the total squared distance of the cluster feature one receives when adding all cluster features that contain only tuples of the i-th cluster—which one can test using the cluster features bounding rectangle.

In contrast to that, the upper bound of the total squared distance TD_i^2 of the i-th cluster is the total squared distance of the cluster feature one receives when adding all cluster features that might contain tuples of the i-th cluster, i.e. all cluster features that have been used to determine the lower bound plus all cluster features having a bounding rectangle that has an overlapping area with the extension of the i-th cluster.

When adding the lower bounds of total squared distance of all clusters, one receives the lower bound of total squared distance TD^2 of a clustering. Analogously, when adding the upper bounds of all TD_i^2, one receives the upper bound of total squared distance of a clustering.

In addition to that, it is possible to compute the *expected total squared distance* by estimating the expected contribution of those general cluster features

[3]For other quality measures of partitioning clustering, see [72]. One can compute several of them using only general cluster features. Thus, total squared distance is only a single representative quality measure of a set of quality measures that one can compute from general cluster features

that consist of several clusters' tuples to the total squared distance[4]. The expected total squared distance is the sum of the minimal total squared distance and the expected total distance of those general cluster features which might contain tuples of several clusters. As this estimation uses only the general cluster features that might contain tuples of more than a single cluster, stochastical influences are lesser compared with testing quality with sampling. Using sampling, all tuples are subject to a stochastic process. Contrary, using general cluster features, some general cluster features that represent only a subset of the data set are subject to the same stochastic process.

Computing quality measures using general cluster features is superior to other techniques of computing them because it needs no scans on the one hand and it guarantees upper and lower bounds on the other hand. Contrary, using a sample for testing the quality of a solution can only give an estimated value of a quality measure.

Therefore, CHAD uses a combination of techniques discussed in this section. As general cluster features are superior for testing quality, CHAD uses general cluster features to test the quality of results. It uses the centroids of general cluster features to find initial solutions. We will discuss this issue in the next subsection.

5.10.4 CHAD's Method of Initialisation

As iteratively selecting initial solutions increases the likelihood of finding the optimal solution, CHAD iteratively selects initial solutions to initialise its third phase. The number of initial solutions, that CHAD uses, is a parameter an analyst can specify. The experiments of this dissertation use ten randomly selected initial solutions which turned out to be a good value for the test data we used.

For finding an initial solution for k-means, CHAD randomly selects k cluster features within the same level of the specific cluster feature tree that has been selected from the general cluster feature tree to fit the need of a specific cluster analysis.

The centroid of a so-selected cluster feature is an initial mean for k-means.

Selecting cluster features from the same level within the cluster feature tree is necessary to avoid solutions where the number of tuples per cluster varies significantly. When choosing the level of which CHAD picks cluster features for initialisation, we considered the following issues:

First, the higher the level of a node is in a cluster feature tree the more the centroid of that cluster feature are closer to the centres of clusters.

[4]The computation of the total squared distance of a cluster uses the sum of squares of tuples of a cluster. Hence, the expected contribution of a cluster feature to the total squared distance of a cluster is the estimated sum of squares of those tuples that are part of this cluster. If the cluster feature might contain tuples of another cluster, too, we also determine the expected contribution of the cluster feature to the other cluster. Finally, we compute the total squared distance in the conventional way.

Second, if one wants to select more than one initial solution there must be more than k cluster features because each initial solution needs k cluster features.

Therefore, CHAD picks cluster feature from the first level below the top-level that has more than $2k$ nodes. The minimal number of nodes $2k$ has been chosen because it offers a good balance between aggregation level and diversity of cluster features. Having $2k$ nodes one can create $\binom{2k}{k}$ distinct initial solutions. If the analyst chooses a number of initial solutions that is higher than $\binom{2k}{k}$, CHAD uses the first level below the top-level of the specific cluster feature tree that has enough nodes to select the analyst-chosen number of distinct initial solutions.

Chapter 6

Anticipatory Classification and Anticipatory Association Rule Analysis

Contents

6.1 Introduction

The clustering algorithm CHAD we discussed in the previous chapter has been designed for cluster analyses in which analysts can experiment with data sets by interactively selecting subsets of a data set and apply partitioning clustering algorithms on it. The current implementation of CHAD uses k-means to partition data sets. Extending CHAD for *EM* clustering is the current project of Messerklinger's master thesis [48].

Yet, anticipatory data mining is a general principle. Moreover, it is not limited to cluster analyses, only. One can also use anticipatory data mining to improve speed and quality of classification algorithms. Additionally, one can use it to increase the speed of constructing frequent itemsets.

The statistics in a general cluster feature tree, pre-counted frequencies of the top frequent pairs of attribute values, and pre-selected auxiliary tuples are all that is needed to realise anticipatory data mining for classification analyses and association rule analyses.

Therefore, this chapter discusses how to use auxiliary statistics and auxiliary tuples for using anticipatory data mining for other applications but clustering. Section 6.2 discusses buffering auxiliary statistics of categorical attributes as CHAD omits storing statistics of categorical attributes. It also shows buffering auxiliary tuples. Each of the remaining sections surveys a technique to use auxiliary statistics or auxiliary tuples except clustering. Section 6.5 shows the derivation of a naïve Bayes classifier using only auxiliary statistics.

6.2 Buffering Auxiliary Tuples and Auxiliary Statistics

We already discussed the general principle of pre-selecting auxiliary tuples and pre-computing auxiliary statistics in Section 4.3.1 on page 111. The general idea is to maintain buffers of auxiliary tuples and pre-computed statistics, namely a buffer for typical representatives of a data set, a buffer for outliers, and a buffer for pre-counted frequencies of categorical attributes. As we have discussed handling of pre-computed statistics of numerical attributes in detail when discussing CHAD, we leave out describing their handling here.

This section presents the details to continuously maintain buffers for auxiliary tuples and auxiliary statistics of categorical attributes.

6.2.1 Buffering Auxiliary Tuples

Section 4.3.1 suggests to store tuples that are typical representatives of a data set as well as tuples that are atypical representatives or outliers. It also mentions to use index based sampling for buffering a representative set of tuples representing typical tuples.

In combination with CHAD it is easy to realise index based sampling as follows: As the first phase of CHAD incrementally maintains a tree of general cluster features with a maximum number of nodes, one can buffer a fixed set of tuples per node. For instance, if the tree has a maximum capacity of n nodes and one takes n' tuples of each node as representatives of that node, then a buffer with capacity to store $n \cdot n'$ tuples is sufficient to keep the typical representatives of a data set.

While there is enough space in the buffer reserved for a specific node, the first phase of CHAD can insert currently scanned tuples into the buffer. If the buffer is full, it has to randomly replace an existing tuple. This technique is known as reservoir sampling [38].

Maintaining the auxiliary tuples representing atypical tuples happens analogously with reservoir sampling—yet, with a single reservoir for all outliers because outliers are widely spread in the vector space of a data set. In other

words, a typical auxiliary tuple represents a given well-defined area of concentration in the vector space of a data set in contrast to an atypical auxiliary tuple that is a representative of a huge sparsely populated area in the vector space of the data set.

Using the sparse distribution of outliers, we can easily identify outliers as entries of leaf nodes that are

1. member of a leaf node that has a significantly higher extension than the average extension of leaf nodes, and
2. that have an above-average distance to the other members of this leaf node.

We determine the extension of a leaf node using the bounding rectangle of the node's cluster feature.

6.2.2 Buffering Auxiliary Statistics of Categorical Attributes

In contrast to ordinal or continuous attributes, categorical attributes have no ordering of attribute values. Hence, sorting them or compressing those tuples with similar values is impossible. Moreover, frequencies of specific attribute values in a set of data is the only remaining type of auxiliary statistic.

The frequency of an attribute value is needed as 1-itemset of association rule analyses and as frequency of a class in classification analyses. Yet for analyses that need to know the co-occurrence of attribute values such as classification analyses, the joint frequency of attribute value combinations is needed.

As it is unclear which frequencies of attribute values and combination of attribute values a future analysis will require, buffering auxiliary statistics must include all of these frequencies. As there might be a huge amount of frequencies when attributes have many distinct values, the amount of frequencies to count might exceed the number of items that can be kept in memory. Hence, it is necessary to limit the number of frequencies to keep.

Unique attributes are unsuitable to be used in an analysis but increase the number of frequencies to store. Hence, one should exclude unique attributes or attributes which are almost unique.

Additionally, as most analyses need only frequent attribute values, a potential solution is to store the top frequent pairs of attribute values in a buffer with fixed size and take the risk of having a frequency not at hand in rare cases.

For storing the top frequent pairs of attribute values, one can reserve a buffer with b entries and maintain it as follows: When a tuple is read it is divided in a set of pairs of attribute values with a pair for each combination of attributes. If a pair is already in the buffer, the according counter is increased. Otherwise, the algorithm inserts a new entry. As long as the buffer has enough free space, each pair of attribute values that is not yet part of the list is inserted into the list with frequency 1. If the buffer is full when trying to insert a new pair an item with low frequency is removed from the buffer.

To ensure that a newly inserted element of the list is not removed at the next insertion, we guarantee a minimum lifetime t_L of each element in the list. Thus, the algorithm removes the least frequent item that has survived at least t_L insertions.

One can realise this by splitting the buffer in two buffers: the first buffer is a FIFO buffer which has the size b_L representing the minimum lifetime, i.e. each element survives at least b_L insertion operations—and more insertions when there have been attribute value pairs that were already part of the list. The second buffer contains the remaining $b - b_L$ combinations. When the FIFO buffer is full, it is tested whether the oldest element of the FIFO buffer or the exceeds the element with minimal frequency of the second buffer. If the FIFO buffer's element is more frequent, it is moved from the FIFO buffer to the second buffer while the least frequent element is removed. If not, the oldest element of the FIFO buffer is removed. In both cases, the FIFO buffer gains the capacity for a new element.

6.3 Deriving a Probability Density Function from a General Cluster Feature Tree

Several applications we will discuss in the following sections need the probability density function of a specific attribute or the joint probability density function of a set of attributes.

Therefore, this section discusses how to derive a probability density function from a general cluster feature tree.

The leaf nodes of a general cluster feature tree represent all tuples of a data set and each node represents a subset of tuples. Hence, these subsets of tuples are a disjunctive and exhaustive partitioning of all tuples.

As the probability density function is unknown, we approximate it with a set of normally distributed variables. As mentioned in Section 5.4, approximating an arbitrary distribution with a set of normally distributed variables uses the well-known technique of kernel estimation in statistics [64]. The Gaussian density function is a good kernel function because it approximates many arbitrary functions quite well. For a detailed discussion of using Gaussian density function for density estimation, re-consider Section 5.4.

If we approximate a leaf nodes probability density function with normal distribution, we can formulate a probability density function for the tuples of each leaf node.

In the case of the probability density function of a single attribute we can formulate the probability density function f_i of the i-th leaf node that is normally distributed with μ_i as mean of the relevant attribute and σ_i as deviation of this attribute as follows:

$$f_i(x) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}.$$

The probability density function of a single attribute is the weighted sum of the probability density functions of all leaf nodes, i.e.

$$f(x) = \sum_i \frac{N_i}{N} f_i(x).$$

By integrating the probability density function $f(x)$ we can verify that $f(x)$ is a normalised probability density function as its result equals 1.

$$\int_{-\infty}^{\infty} f(x) = \int_{-\infty}^{\infty} \sum_i \frac{N_i}{N} f_i(x)dx = \frac{1}{N} \sum_i \left(N_i \underbrace{\int_{-\infty}^{\infty} f_i(x)dx}_{=1} \right) = \frac{\sum_i N_i}{N} = 1$$

When determining the probability density at any given point x_0, there is no need to test the probability density of all nodes as many of them would be zero at that given point. Especially, we can exclude all nodes where x_0 is outside the interval of the lower limit bl and the upper limit tr of that node. Therefore, we select only those nodes where the condition $bl \leq x_0 \leq tr$ holds. As a general cluster feature tree is organised similarly to an R^+-tree, we can efficiently query those nodes.

6.4 Using a General Cluster Feature Tree to Find Good Split Points of a Decision Tree

When a decision tree includes numerical attributes the number of potential split points is infinite, as we discussed in Subsection 2.3.3. Hence, a heuristic is needed which finds good split points.

Markus Humer searched in his master's thesis [35] for beneficial split strategies of Rainforest. He compared conventional split strategies and split strategies using only auxiliary data. His tests show that split strategies using only auxiliary data tend to deliver smaller and robuster decision trees which means that using auxiliary statistics performs a more efficient search for the optimal split point[1]. As auxiliary statistics are statistics of the data set they were taken from, they are independent from the training set, i.e. an ill chosen training set has few influence on finding good split points. Consequentially, the so-found split points are robust. This section demonstrates finding split points using the probability density function of an attribute. We will revise the afore-mentioned tests in the succeeding chapter.

The goal of shattering a set of tuples ideally in subsets of tuples that are homogeneous with each other is related with the goal of partitioning clustering.

Therefore, we use CHAD's partitioning clustering method to produce a set of partitions. We receive a set of clustering features where each cluster feature consists only of a single attribute—the attribute we want to split.

[1] The better a split point is the more homogeneous are the split partitions. Consequentially, there is less need to do another split.

One can use the cluster features of the so-found clusters to determine the points where the probability density functions of a pair of clusters assume the same value. Markus Humer's tests have evaluated that the points of equal probability density are good candidates of split points [35]. There is no need to test each pair of clusters. As there is only a single relevant attribute, it is better to order clusters according their mean and test each cluster with its left and right neighbour, only. Doing this for all clusters, we receive the minima of the probability density function of the split attribute.

The minima of the probability density function of the split attribute are good candidates for splitting.

Additionally, one can use the minima of the probability density function to define a histogram of the attribute of the probability density function.

Histograms can be used to make continuous attributes discrete by assigning a discrete value to each interval of the histogram.

6.5 Intermediate Results and Auxiliary Data for Naive Bayes Classification

Subsection 2.3.2 has introduced the concept of naïve Bayes classification. With the concepts of anticipatory data mining being introduced, this section shows how to apply the concept of anticipatory data mining to improve naïve Bayes classification.

It is very easy to deduce a naive Bayes-classifier using only pre-computed frequencies. A naive Bayes-classifier classifies a tuple $t = (t_1, \ldots, t_d)$ as the class $c \in C$ of a set of classes C that has the highest posterior probability $P(c|t)$.

The prior probabilities $P(t_i|c)$ and the probability $P(c)$ of class c are all that is needed to determine the posterior probabilities according to the formula

$$P(c|t) = P(c) \prod_{i=1}^{d} P(t_i|c). \tag{6.1}$$

Determining the probabilities on the right hand side of formula 6.1 requires the number of tuples, the total frequency of each attribute value of the class attribute, and the total frequencies of pairs of attribute values where one element of a pair is an attribute value of the class attribute, as the probabilities $P(c)$ and $P(t_i|c)$ are approximated by the total frequencies as $P(c) = \frac{F(c)}{n}$ and $P(t_i|c) = \frac{F(c \cap t_i)}{F(c)}$, respectively.

Frequency $F(c)$ is also the frequency of the 1-itemset $\{c\}$, which is an auxiliary statistic as we discussed in sectionbuffering. As count n is the sum of all frequencies of the class attribute, the frequency of pairs of attribute values is the only remaining item to determine.

Storing all potential combinations of attribute values is very expensive when there is a reasonable number of attributes but storing the top frequent combinations is tolerable. As the Bayes classifier assigns a tuple to the class that

maximises posterior probability, a class with infrequent combinations is rarely the most likely class because a low frequency in formula 6.1 influences the product more than several frequencies that represent the maximum probability of 1.

As a potential solution, one can store the top frequent pairs of attribute values in a buffer with fixed size and take the risk of having a small fraction of unclassified tuples.

Counting the frequency of attribute value pairs is only appropriate when the attributes to classify are ordinal or categorical because continuous attributes potentially have too many distinct attribute values.

If a continuous attribute shall be used for classification, the joint probability density function replaces the probabilities of pairs of attribute values in formula 6.1. One can determine the joint probability density function as described in Section 6.3.

The parameters necessary to determine the joint probability density function such as the covariance matrix are auxiliary statistics for the succeeding Bayes classification.

Hence, pre-computed frequencies and a set of pre-computed parameters of probability density function are all that is needed to derive a naïve Bayes classifier. Subsection 7.5.2 shows that the resulting classifiers have high quality.

6.6 Accelerating Algorithms Finding Association Rules

Efficient algorithms finding association rules such as Apriori or FP-growth find the frequent itemsets in a set of scans before searching for valid association rules using the set of frequent itemsets. Apriori needs multiple scans of the database in which it counts the frequency of itemsets with increasing length of itemsets—i.e., the first scan determines the frequent 1-itemsets, while the d-th scan determines the frequent d-itemsets. FP-growth needs the 1-itemsets to have an ordering of items when scanning the data a second time to build a frequent pattern tree.

By storing the frequencies of attribute values in anticipation, one receives the frequencies of 1-itemsets for free when starting an association rule analysis. In other words, Apriori can save exactly one scan. FP-growth can also save one scan which effectively halves the number of needed scans. As this condition is obvious when analysing both algorithms, we omit demonstrate with tests.

Chapter 7

Experimental Results

Contents

7.1 Overview of Evaluation and Test Series

This chapter presents experiments performed with two prototypes that implement the concepts of anticipatory data mining. To be more specific, there exist a prototype implementing CHAD and a prototype pre-processing the auxiliary statistics mentioned in chapter 6.

As we discussed in Section 1.3, evaluating concepts is an important issue in design research. Evaluating concepts with a proof-of-concept prototype is an important technique of evaluation but not the only one. Especially, it is possible to analytically show important properties of artefacts, i.e. properties of the method *anticipatory data mining*.

First of all and most important, the existence of a proof-of-concept prototype proves that it is possible to realise the concept *anticipatory data mining*. Furthermore, we use the prototypes to evaluate several sub-concepts shown in previous chapters. Each of these concepts has been tested by a set of test series.

Yet, tests using benchmark data can only show the behaviour of a prototype in a small subset of all potential data sets. Hence, one can use the test of a data set to have an idea how the tested prototype will react when analysing data that share similar characteristics with the data used for testing. Yet, there is no guarantee that all potential phenomena are covered by these tests.

Therefore, using experiments is only a technique of a set of techniques to evaluate the concepts of anticipatory data mining. The essential concepts are evaluated by a combination of experiment and analytical evaluation.

Additionally, there are characteristics of sub-concepts of anticipatory data mining which one can formally prove. Therefore, we omit testing those concepts which can be proved otherwise.

This section specifies a set of hypotheses each of them regarding a concept of this dissertation. Each test series matches one of these hypotheses. Yet, there are hypotheses already proven analytically in previous chapters. For the sake of completeness, this section also lists these hypotheses and references to the section containing the proof.

The hypotheses concern improvements of time and quality, where hypotheses concerning quality includes general hypotheses and hypotheses only relevant for a specific data mining technique.

7.1.1 Hypotheses Concerning Time

The hypotheses concerning time are as follows:

first phase of CHAD is scalable CHAD's first phase implements the premining phase of anticipatory data mining for numerical attributes. For guaranteeing high performance, it must be possible to incrementally update intermediate results and auxiliary data. For the same reason, CHAD's first phase must scale well in the number of tuples. To be more specific, it must scale linearly in the number of tuples. For this reason, there is a set of test series consisting of several tests with increasing number of tuples. The data used in this test series are synthetically generated data. The reason for using synthetic data is that there a too few very large benchmark test sets available. Table 7.1 shows the parameters of used test series.

Each test of a test series consists of six runs of the first phase of CHAD. Each of these runs uses a different set of data. Hence, at least six different data sets are needed. Each data set includes a finite set of Gaussian clusters. Table 7.3 gives an overview of used data sets.

The tests C_1 to C_{22} examine CHAD's scalability in the maximum number of nodes of the general cluster feature tree, respectively.

For being scalable, CHAD must scale linearly or better in the number of tuples, number of dimensions, and number of nodes.

Time needed for second phase of CHAD is negligible The second phase of CHAD derives a clustering feature tree from a general cluster feature tree to fit the needs of a given analysis. For doing so, CHAD has to scan the general cluster feature. To be more specific, depending on the selection predicate it can omit scanning branches that are irrelevant with respect to the selection predicate. As all operations happen in main memory, this task is very fast.

While performing the tests, the second phase of CHAD always lasted about a hundredth of a second which is the most fine-grained interval of the clock function of the machine used for testing. As the time needed to compute the result of CHAD's second phase is so small as it is, we omit examining it further.

CHAD's third phase is independent of the number of tuples Obviously, the time needed to compute the result of CHAD's third phase is independent of the number of tuples in the data set that has been pre-mined in CHAD's first phase because a clustering feature tree consists of a fixed maximum number of entries. In other words, scanning the clustering feature tree involves at maximum a number of entries that is significantly smaller than the number of tuples. Moreover, if the number of tuple increases, the number of entries cannot exceed the upper limit of entries. Hence, the third phase of CHAD must be independent of the number of tuples, i.e. its runtime complexity concerning the number of tuples is constant $\mathcal{O}(1)$.

We omit examining the minimum number of nodes that are necessary to apply CHAD because few hundred of nodes have shown to be sufficient for CHAD. Additionally, CHAD's third phase runs about a second with few hundred of nodes. Consequentially, limiting the number of nodes would make no sense because runtime improvement would be unnecessary.

Time needed for third phase of CHAD is low Although the time needed for CHAD's third phase is independent of the number of tuples in the data set, there exist other factors influencing the runtime of CHAD's third phase. Hence, there exists a test series showing that the runtime of CHAD depends on the tree size and is generally low enough such that an analyst can test a reasonable number of analyses with distinct parameters in short time.

7.1.2 General Hypotheses Concerning Quality

Quality is measurable and has a deterministic upper limit It is incorrect to assume that quality is measurable in each analysis. When using a sample to compute the result of an analysis, each quality measure that

used the sample for its computation measures the quality of the sample, only. Especially, it does not measure the quality in the total data set. Therefore, it is necessary to evaluate the quality of a result of an analysis externally, for instance with another sample that serves as estimate of the quality of the total data set. Yet, the so-determined quality is just an estimate of the real quality which might significantly differ from the measured quality. Moreover, if the sample has itself been taken from a sample then the sample of the sample is only capable to estimate the quality of the sample it was taken from—and not the total data set. In other words, if the initially taken sample is biased in any way, then all samples taken from it will deliver biased results without being able to detect this phenomenon.

In contrast to that, the pre-mined statistics are sufficient statistics for computing quality measures of the total data set. It is also possible to give deterministic limits of quality measures. We have discussed this issue extensively in Subsection 5.10.3 and have formally shown it there.

Approach is generally applicable Related approaches using trees of sufficient statistics are limited to analyses where the subset of a data set relevant for an analysis is identical with the data set used to determine the sufficient statistics. One could use the alternative to pre-process a sample with all attributes and perform selection and projection operations on this sample.

Yet, the sampling alternative might face the problem that the selected sample is empty—although there would be tuple in the data set that fulfill the selection predicate. In such a case, the second phase of CHAD returns a non-empty tree of clustering features.

CHAD is also superior to other approaches that offer no support for reusing sufficient statistics in multiple analyses with varying subsets of relevant data.

7.1.3 Hypotheses Concerning Quality of Clustering

quality of aggregated data is sufficient It is possible to shrink the time needed for computation of the second phase nearly to any short period of time by taking very small samples of the data or by limiting the number of nodes in the general cluster feature tree to a very small number. Yet, the resulting clustering model is likely to be of low quality. Hence, there is a set of test series that examines the minimal number of nodes the general cluster feature tree must have to produce results that share about the same quality with a clustering that is done using all tuples instead of their aggregated representations.

As sampling is a commonly-used alternative to using aggregated representations, another set of test series examines the minimum size of a sample that fulfills the same condition as shown above.

Both sets of test series are identical. They use data sets consisting of synthetic data and benchmark data which are available to the public. Synthetic data has the advantage that the optimal locations of the centroids are known but they are only somewhat useful for comparing aggregated representations and samples. As artificial data are typically unsorted and independently and identically distributed, sampling techniques face a very easy problem—which might be oversimplified. If there is no order within the data it is possible to take the first n tuples of a table as a sample for all tuples in that table. The so-taken sample would be unbiased. In general, it is necessary to assume that tuples are ordered making this sampling method inapplicable. Tuples might be ordered because the table has been sorted for quicker access of its contents or because the tuples have been inserted in any unintended order—for instance, if a company inserts all sales of a store in a bulk load operation, parts of the table are ordered.

With the optimal locations of clusters being known within the synthetic data sets, all tests with synthetic data use the number of clusters that is specified in the description of the used data set to check whether the known optimal solution is found. If we would test with another number of means, we could only receive suboptimal results.

Initialisation outweighs other effects Partitioning clustering algorithms need an initial solution to produce a final solution. Hereby, the initial solution determines the quality of the resulting solution.

We will show in a test series that the effect of the initial solution on quality outweighs all other effects. Therefore, we use a set of benchmark data sets a apply k-means on these data sets.

If we can show that the initial solution determines the quality of the result more than other effects, then slight losses in quality due to aggregation are tolerable as they are outweighed by the effect of the initial solution.

As we have extensively discussed in Section 5.10, trying independently finding initial solutions many times, is very likely to find the optimal initial solution. Hence, one can use some of the cost saving of anticipatory data mining to search more often for a good initial solution. As mentioned before, this proceeding is more likely to succeed than searching a single time with a large set of data.

As k-means is liable to the chosen initial solution in terms of speed and quality of results, each test is iterated six times—each time initialised with a different seed to receive a different initial solution each time.

For the optimal solution being unknown within benchmark data sets, the tests with benchmark data use the total sum of distances as criterion for quality.

7.1.4 Hypotheses Concerning Quality of Classification

Using auxiliary tuples as training set improves quality Markus Humer [35] implemented the concept of anticipatory data mining for classification using decision trees. His tests show that using auxiliary tuples as training set improves the accuracy of decision tree classification. We will discuss these results in a section of this chapter. These results also have been published in a paper [25].

Using auxiliary statistics results in highly-accurate Bayes classifiers The approach mentioned in Section 6.5 that derives a naïve Bayes classifier for free from a set of pre-buffered auxiliary statistics has been implemented as a prototype. Its results are published in a paper [25]. The results show that the accuracy of the so-determined classifiers is very high and comparable with a very large sample. Yet, one can receive the classifier for free, i.e. without accessing tuples from disc.

7.1.5 Hypotheses Concerning Association Rule Mining

Anticipatory data mining saves one scan By buffering frequencies of attribute values, one receives the 1-itemsets which one can use for association rule analyses to improve the speed of Apriori or FP-growth, respectively. In other words, the 1-itemset is an intermediate result that has been pre-mined in anticipation of future association rules analyses. As the saving is constantly a scan which one can verify by analysing Apriori and FP-growth, respectively, we omit showing this in a test series.

7.1.6 Description of Used Data Sets

Most tests demonstrating the quality of CHAD's results use data of real world problems such as climatical data or medical data.

Yet, some tests use synthetical data to show how CHAD handles data sets that are known to be hard to analyse.

All synthetic data sets used for testing have only numerical attributes but they vary in number and location of clusters. There are some data sets that can be easily used for k-means clustering, while there are other data sets the k-means algorithm is known to face problems with such kind of data sets. For instance, there is a data set containing no noise and clusters with low deviation—which is known to be an easy problem for k-means. Other data sets have clusters with high deviation and a high proportion of noise—which might cause k-means to produce bad results if it has not been initiated with a good initial solution. If there is a mix of clusters with high deviation and clusters with low deviation, k-means also produces bad results. Hence, there are some data sets with clusters of varying size and deviation.

Number	No. tuples ($x * 1000$)				
↓ of nodes	200	400	600	800	1000
100	C_1				C_{10}
200	C_2				C_{11}
400	C_3				C_{12}
800	C_4				C_{13}
1,600	C_5				C_{14}
3,200	C_6				C_{15}
6,400	C_7				C_{16}
12,800	C_8				C_{17}
25,600	C_9	C_{21}	C_{22}	C_{23}	C_{18}
51,200					C_{19}
102,400					C_{20}

Table 7.1: Names of tests with number of nodes resp. tuples of test series scaling

data set	description	details in table
D_α	easy k-means problem with few clusters having low deviation; noise is missing.	A.1, see page 221
D_β	set containing few clusters with medium deviations and a low proportion of noisy data; deviation of clusters only slightly varies.	A.2, see page 222
D_γ	same set as D_β except the proportion of noisy data is significantly higher.	A.3, see page 222
D_δ	data set without noise; there is one big cluster with a high deviation and there are several small clusters with low deviation.	A.4, see page 229
D_ϵ	data set with many clusters of similar deviation without noise.	A.5, see page 230
D_ζ	data set with several clusters of varying deviation without noise.	A.6, see page 231

Table 7.2: description of synthetic data sets

data set	description
El Niño	Spatio-temporal data storing the sensor values of a network of buoys in the Pacific ocean for several years. Data has been collected to explain or predict the weather anomaly called El Niño effect that is made responsible for calamities in coastal areas of the Pacific ocean. The data set is available to the public by the UCI KDD archive.
Telco	The Telco data set (*TEL*ecommunication *CO*mpany) is an anonymised sample of usage data of cellular phone customers. It contains a single tuple per customer and more than a hundred attributes per customer. There are approximately 10'000 customers. The data set is used to predict which customer will cancel one's contract. Due to legal issues and corporate interests these dissertation omits the description of the data set's attributes. Aggregated results can be found in this section and in a paper [25].

Table 7.3: description of synthetic data sets

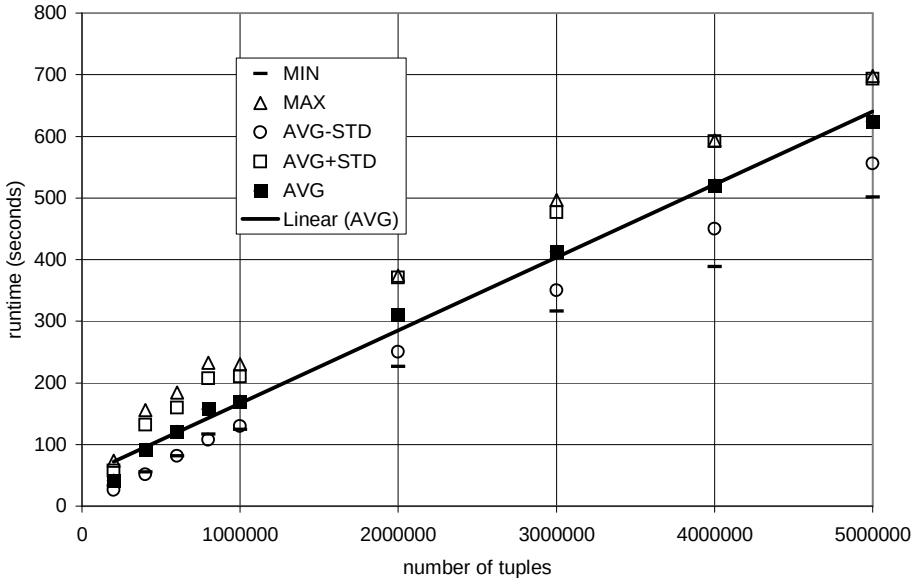


Figure 7.1: Scaling of CHAD's first phase in the number of tuples

7.2 Results of Tests Demonstrating Scalability of CHAD

This section discusses the results of the tests concerning the scalability of CHAD. For the sake of a compact description, all charts in this section are summaries of test series. For a complete description of test results, please confer section A.3 of the appendix.

For being scalable, the runtime needed for the first phase of CHAD must scale at worst linear with the number of tuples. Compared with the number of tuples other factors that influence CHAD's runtime such as maximum number of nodes or number of dimensions are small in fact tables of data warehouses—which is the typical scenario of anticipatory data mining. Hence, this section shows the scalability of CHAD in the number of tuples first.

The tests of test series C which tested the scaling of CHAD's first phase in the number of tuples show that CHAD scales linear in the number of tuples. Figure 7.1 shows the runtime of CHAD's first phase with rising number of tuples. Although all data sets have the same number of tuples the clustering of some data set happens faster than the clustering of other data sets. This means that runtime might depend on the scanned data and the order or access. However, the data set's influence and the sequence of access on runtime are minor as the low deviation of runtime from the average runtime in figure 7.1 indicates.

The tests C_1 to C_9 and C_{10} to C_{20} examined the effect of maximum number of nodes on runtime. The capacity of each node was 5 in all tests, i.e. each

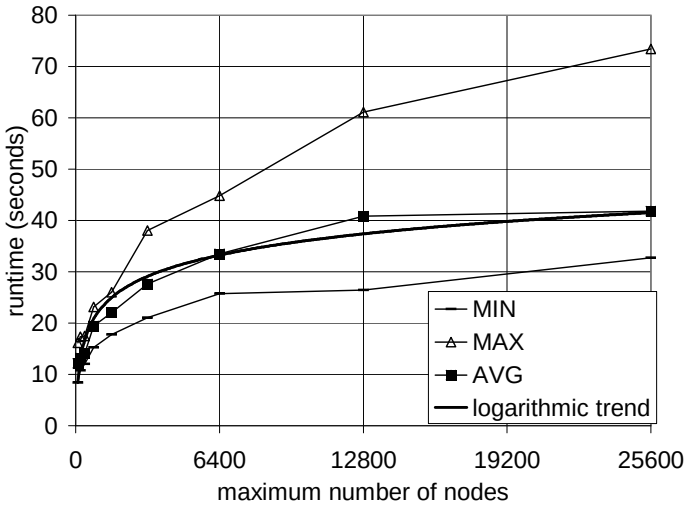


Figure 7.2: Logarithmic scaling of CHAD's first phase in the max number of nodes

node contains at least 5 and at maximum 9 general cluster features. A capacity of 5 gives a good balance between the number of inner nodes and leaf nodes: Inner nodes are important for quickly selecting nodes while the more leaf nodes a tree has the smaller is the threshold of a cluster feature tree with the consequence that tuples are less aggregated. The results show that runtime scales logarithmical in the maximum number of nodes, as depicted in Figure 7.2 and 7.3, respectively. Figure 7.3 also illustrates the effect that occurs when the tree no longer fits the main memory. When testing, the database management system and CHAD shared the same machine. In a single run of CHAD it happened that the main memory was insufficient to keep the general cluster feature tree and the cache of the database system in memory. Hence, the machine began swapping memory to disk causing a significant decrease in performance. Consequentially, it is necessary to choose the maximum number of nodes in a way that the general cluster feature tree and other programs fit into available main memory.

Figure 7.3 also shows that CHAD's first phase scales logarithmic for most of the tests in the test series

CHAD re-organises the general cluster feature tree each time the tree is full. Initially, its threshold is zero which it increases using linear regression method, i.e. it maintains a list of previous threshold values and the number of tuples the tree was able to store with that threshold. Each time it reorganises the tree it selects the threshold such that the expected capacity of tuples is expectedly twice the current capacity, i.e. the tuples the tree could insert before it became full. Figure 7.5 illustrates how the threshold steadily increases with increasing number of tuples scanned.

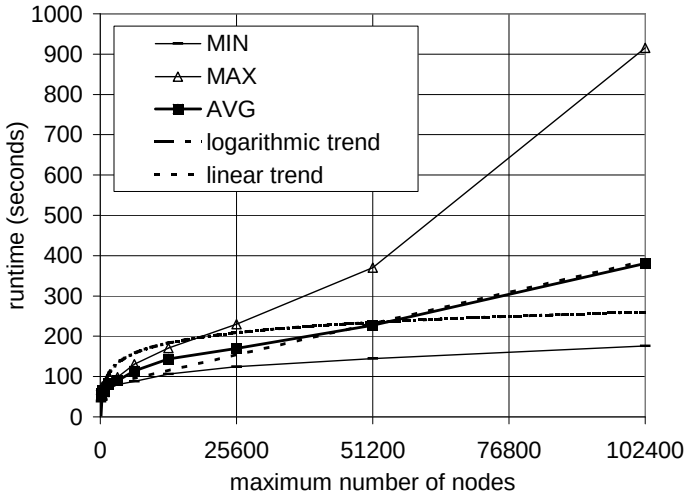


Figure 7.3: Decrease of performance if main memory is insufficient

The first reorganisation of the general cluster feature tree needs special consideration as the history of pairs of threshold and capacity is empty. CHAD follows the suggestion of BIRCH that slightly increases the threshold in a way that at least the most similar pair of entries of at least one node can be merged. This might cause BIRCH and CHAD that reorganisation is needed again very soon. As the capacity of BIRCH and CHAD is sensitive to the order in which the algorithms scan tuples, it might happen that the capacity after increasing the threshold is lower than before. Thus, we observe a leveling-off phenomenon when determining the first re-organisation. Figure 7.4 depicts threshold and number of nodes of the first the first 14 reorganisations of a general cluster feature tree. The first reorganisations overestimate or underestimate an optimal threshold, indicated by the absence of a trend during the first nine reorganisations. Yet, once the history of reorganisation contains a sufficient number of pairs of threshold and capacity, this phenomenon no longer occurs.

Obviously, reorganising has no significant effect on runtime. Otherwise, runtime could not increase linearly with the number of tuples. Hence, several reorganisations can be tolerated.

Finally, a test series analysed the runtime of CHAD needed for phases two and three. The runtime used for phase two is listed in Figure 7.4. The test series used all tuples of data set D_ζ which is the data set with the most clusters in it. Additionally, the number of nodes was 25600, which corresponds with a capacity of 230 000 entries. In quality tests, a much smaller number of entries showed to be sufficient to receive good results. Hence, the test series measures extreme values of runtime. For the same reason, Figure 7.4 also depicts a test series with an average sized general cluster feature tree having a capacity of more than 23 000 general cluster features—all other conditions remain the same.

Obviously, phase 2 is negligible compared with phase three, especially com-

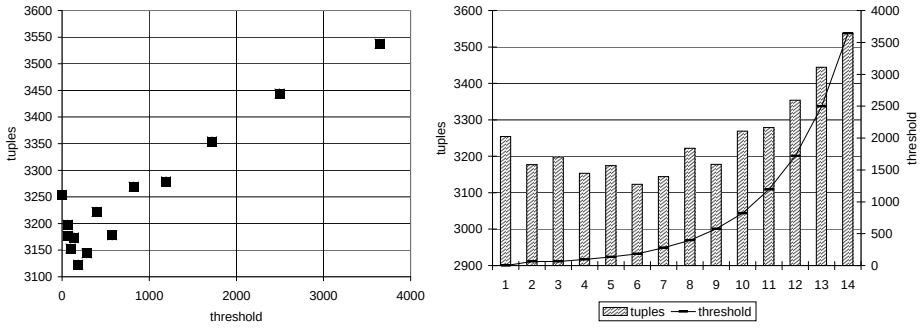


Figure 7.4: leveling-off during first reorganisation

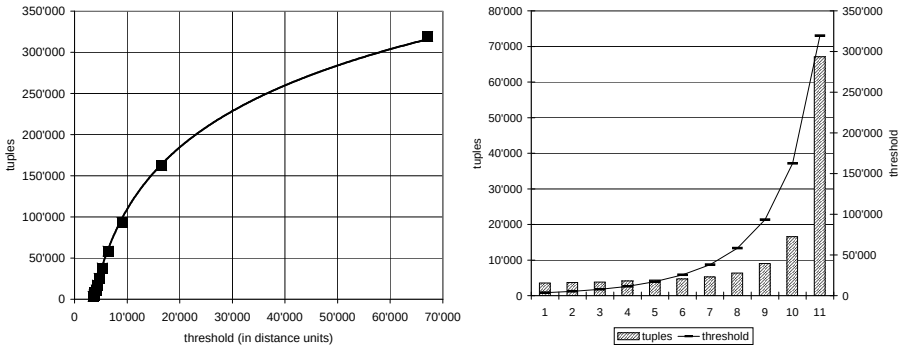


Figure 7.5: correlation of threshold and tree capacity in tuples

(a)

phase 2	init	optimal
all in seconds		
9.094	63.341	0.681
9.219	59.355	0.05
9.359	63.091	0.961
8.891	58.945	0.761
9.125	60.628	1.021
9.297	64.723	1.142

(b)

phase 2	init	optimal
all in seconds		
0.812	2.546	0.406
0.828	3.125	0.391
0.813	3.265	0.39
0.828	3.922	0.593
0.813	2.906	0.703
0.844	1.688	0.296

Table 7.4: Runtime of test series with (a) very large clustering feature tree and (b) medium-sized clustering feature tree

pared with the initialisation of phase 3. The test series show that the number of nodes of the cluster feature tree influence the runtime of phase three. CHAD's third phase spends most of its time to find a good initial solution using the strategy we discussed in Subsection 5.10.4 while it finds the final result very quickly.

Consequentially, one can improve the performance of CHAD's third phase by decreasing the number of nodes. One can receive a good balance of runtime and quality by using few hundreds of nodes for initialisation and a few thousands of nodes for the final run of k-means.

Especially, the runtime using a tree with average size requires a few seconds to compute a result which is low enough for an analyst to interactively examine the solution space. In other words, an analyst can experiment with data sets by including/exluding attributes as well as defining new attributes. CHAD can give the analyst a upper limit for the quality of each solution. Once a very good solution is found, the analyst can use a large tree to find the most accurate solution.

Dominik Fürst has examined the scaling behaviour of a single run of k-means using clustering features and found out that it scales linearly in numbers of nodes, dimensions, clusters, and the number of iterations needed [19, chapter 6]. The number of needed iterations slightly increases with the number of nodes but the better the initialisation is the less iterations are needed.

Summarising the scalability tests, a single run of the prototypes of anticipatory data mining including initialisation scales linearly or better in all relevant factors, namely number of dimensions, tuples, and tree size. To be more specific, only the construction of intermediate results and auxiliary data depends on the number of tuples while the time needed for using intermediate results and auxiliary data is independent of the number of tuples. As mentioned in previous chapters, it is possible to incrementally update general cluster feature trees and buffers with auxiliary tuples and auxiliary statistics. Hence, one can use anticipatory data mining during the load cycle of a data warehouse which means that auxiliary statistics and auxiliary tuples are at hand when an analyst needs them.

7.3 Effects of Clustering Features and Samples on Cluster Quality

This section discusses experiments analysing the quality of clustering algorithms using cluster features. As sampling is commonly used technique to speed-up algorithms when the number of tuples is very large, this section compares the tests of clustering algorithms using cluster features with tests of clustering algorithms using samples. It also compares these tests with tests using all tuples.

This section will evaluate the superiority of approaches using cluster features with a set of experiments of related work and experiments under the supervision of this dissertation's author.

Several authors have shown the superiority of cluster features when compared with approaches using sampling. Hence, we discuss their results in this section.

Huidong Jin et al. used EM clustering to analyse the forest cover type data set of the UCI KDD archive, a popular benchmark data set. Their approach improves EM clustering using cluster features. They compare their improved approach with the original clustering algorithm EM on the one hand and EM clustering using 15 % samples on the other hand. While the original algorithm takes about 170 thousand seconds to compute a result, the sampled approach needs only about 50 thousand seconds to solve the same task. Yet, their approach using cluster features needs only a fraction of the time the sampled algorithm needs, namely about 8 thousand seconds. They test the quality of results by comparing the accuracy of tuples, i.e. in the data set the real cluster affiliation of a tuple is known enabling comparing real affiliation and assigned affiliation of tuples.

Huidong Jin et al. found that the difference in accuracy of their approach and the sampling approach is at least 5 % in favour of their approach using cluster features [37, Figure 4].

Other test series of Huidong Jin et al. with synthetical and benchmark data show the same trend. Yet, in these tests the difference in runtime is less significant.

Bradley at el. also used EM clustering in combination with cluster features to analyse synthetical and benchmark data [6]. Each time, they compared a sample of a given size with a set of cluster features having the same size. Additionally, they used the uncompressed and un-sampled data set in an additional test. As the real cluster were unknown they used the information gain of a cluster to indicate the homogeneity of clusters.

In all tests, the approach with cluster features returns the best results of all tests [6, tables 2 and 3]. Moreover, the results were better than EM using all tuples which can be explained that cluster features are less vulnerable to outliers.

When generating data synthetically the creator of the data has the opportunity to generate the data in a way such that the optimal solution is known—which is not the case in most real world data due to the huge amount of potential

total deviation	tuples	average runtime (s)	
416.66	5'000'000	3727.39 + 68.8	CHAD
430.14	5'000'000	15'790	k-means all tuples
428.11	2'000'000	4'500	k-means 40 % sample
424.74	800'000	1'925	k-means 16 % sample

Table 7.5: Total deviation from optimal solution of CHAD and sampled k-means

solutions.

With the optimal solution being known it is possible to check if an algorithm finds this solution—or if not, if the found solution is at least in the vicinity of the optimal solution.

Table 7.5 depicts a comparison of test series using CHAD and k-means on the data set SHF (see appendix for details). The table depicts the total deviation of the found means $\vec{\mu}_i$ and the means of the optimal solution $\vec{\mu}_i^0$, which is the sum of the Manhattan distances of all pairs $(\vec{\mu}_i, \vec{\mu}_i^0)$, i.e. $\sum_i ||\vec{\mu}_i - \vec{\mu}_i^0||$. The smaller the deviation the better is the solution. CHAD used 130'000 leaf node entries in this test. The table shows that CHAD delivers the best solution but needs less time than an average sized sample. To be more specific, the first phase of CHAD needs the majority of its runtime while the remaining phases need only a fraction of the time which the first phase needs.

Yet, Table 7.5 also shows that increasing sample size decreases the quality of the result which is a non-expected result because typically one would expect increasing quality. Assuming that quality rises with increasing number of tuples, there must be other effects on quality. Each of the tests shown in 7.5 has its own initialisation. Therefore, the next section discusses a set of tests examining the effect of initialisation and other effects on quality.

7.4 Comparing Effects on Quality

In the previous section, we have seen that clustering using clustering features returns better results than clustering of samples. This means, that sampling error typically outweighs the error of wrongfully assigned tuples when using clustering features. Yet, all previously mentioned quality tests but the last test series we discussed in the previous section used all the same initial solution. Hence, the effect of initialisation on clustering has been hold out until now.

Therefore, this section examines the effect of initialisation on the quality of clustering. Moreover, it compares the effect of initialisation with other effects on quality. As sampling has shown lower quality in the last section, we compare sampling effect on quality with initialisation effect on quality.

As a sample is only a part of the data set from which the sample has been taken, it is possible that tuples that are important for the correct result are not part of the sample—causing bad results. We call the error that occurs due to an ill-chosen sample *sampling error*.

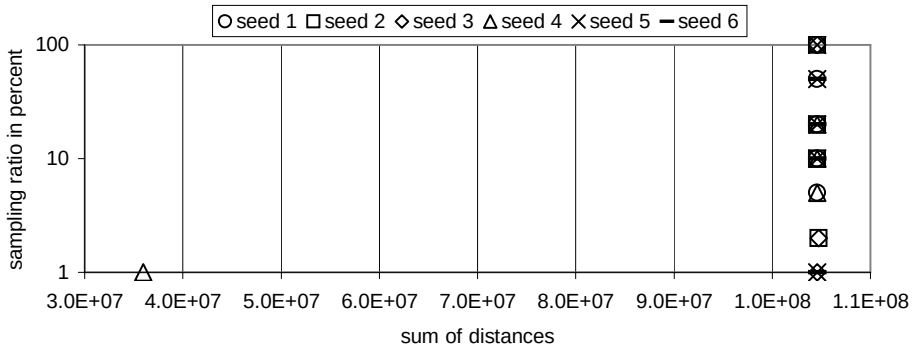


Figure 7.6: El Niño test series with three clusters

Yet, sampling error is only one component of the total error of an optimisation algorithm like k-means. k-means is known to be significantly dependant of a good initialisation.

In order to take apart the effects of sampling and initialisation on a test, each test is repeated for six times—each time with a different seed but the same sample. The quality of tests that use the same sample can only differ due to different initialisations. Hence, initialisation error is the only error component that can manifest itself within these tests.

The test series documented in this section used the El Niño data set, a popular benchmark data set from the UCI KDD archive. The schema of this benchmark data can be found in the appendix. All tests have been performed on the same machine, a PC with 1.8 GHz Pentium IV processor and 512 MB RAM.

Some tests failed to deliver a proper results, i.e. the algorithm returned less than k clusters. An ill-chosen initialisation can be the reason that an initial partitioning is empty causing the algorithm to return less than k clusters. Tests returning no result were omitted but can easily be identified. Figure 7.13 shows that there are only two test results for the 2-percent sample. As there have been six tests, four tests must have failed to deliver k means.

The k-means algorithm used in the test series initialises itself by taking a random sample of the data to analyse and apply itself once to this sample. As the data that is about to be analysed is already a sample, the data used for initialisation are a sample of a sample. We will call this method *initially clustering a sample*.

In test series with a sample ratio less or equal 2 % another initialisation method was chosen because the sample of the sample did not contain enough tuples. In such a case, the initialisation consisted of k randomly taken tuples of the data. This was the case for tests where the sampled data were a one or two percent sample of the original data, We will call the method used for small samples *randomly selecting tuples*.

The test results show that the two initialisation methods differ in the likeli-

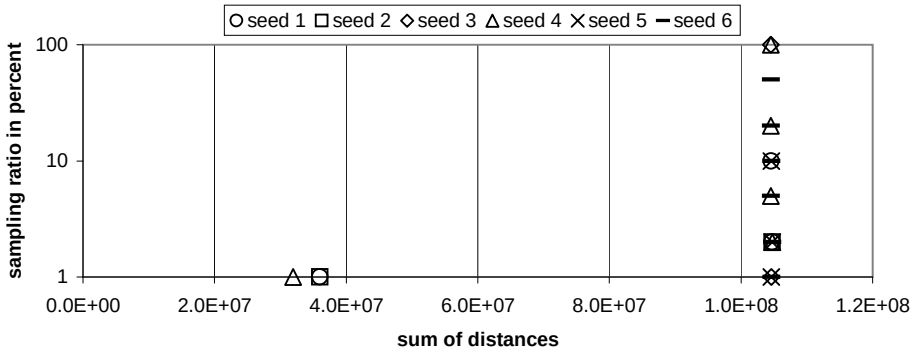


Figure 7.7: El Niño test series with four clusters

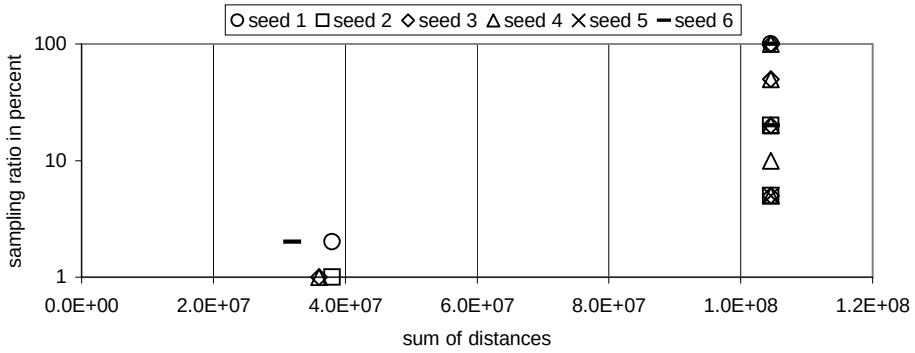


Figure 7.8: El Niño test series with five clusters

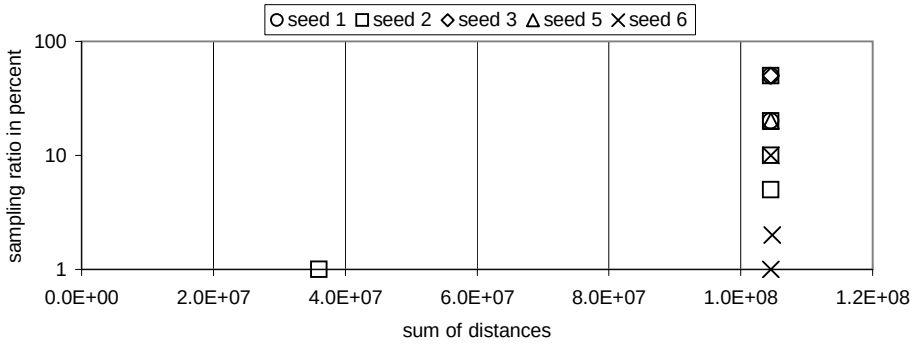


Figure 7.9: El Niño test series with six clusters

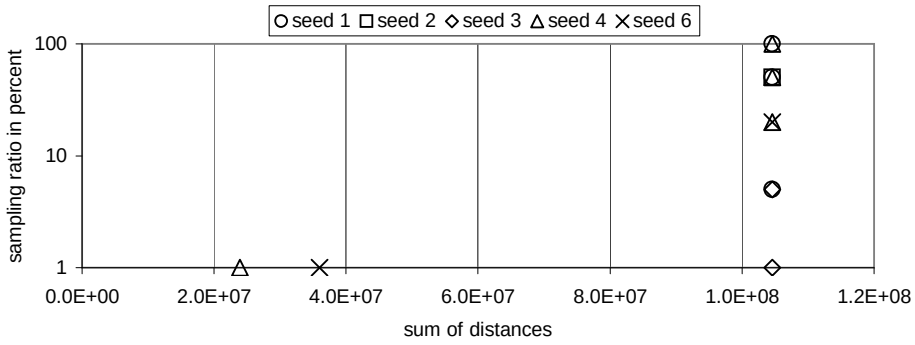


Figure 7.10: El Niño test series with seven clusters

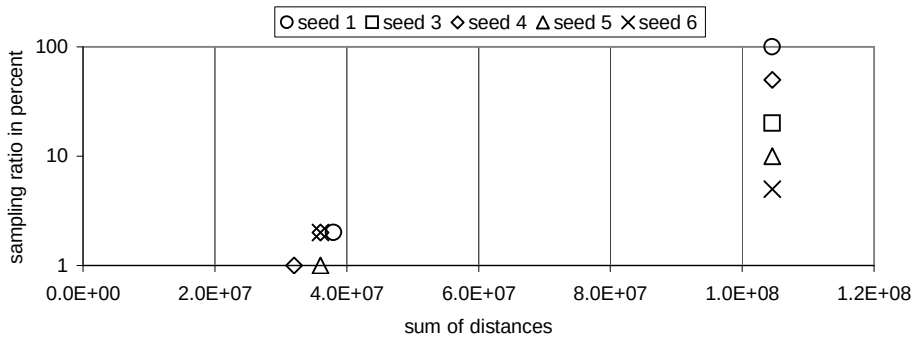


Figure 7.11: El Niño test series with eight clusters

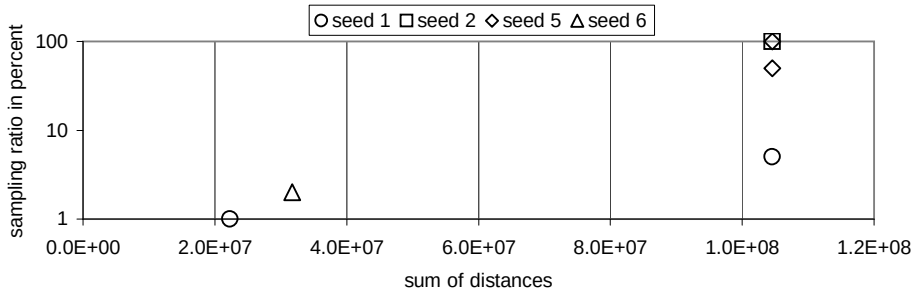


Figure 7.12: El Niño test series with nine clusters

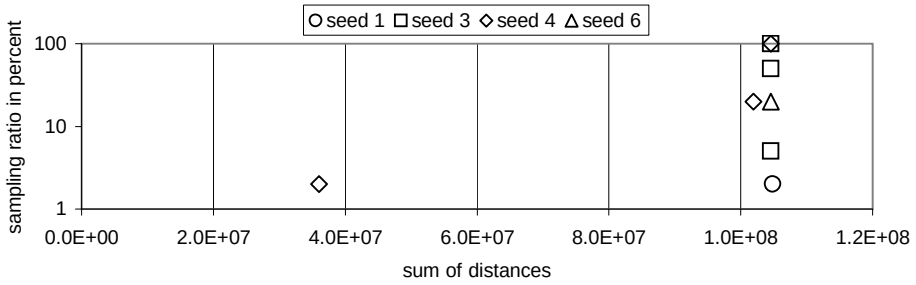


Figure 7.13: El Niño test series with ten clusters

hood distribution of returning good results.

The initialisation method *randomly selecting tuples* seems to deliver better results as shown in figures 7.6, 7.7, 7.8, 7.10, 7.11, and 7.12. Yet, Figure 7.15 shows that the initialisation method *randomly selecting tuples* is not better in general. Figure 7.15 shows the same results as shown in Figure 7.6 with the only difference that the best solution is omitted. With the best solution omitted, we observe in Figure 7.6 that quality increases with increasing sampling ratio.

As Figure 7.15 shows, the initialisation method *randomly selecting tuples* also delivers the worst results. In addition to that, Figures 7.10,7.11, 7.12, and 7.13 show that there are many tests with improper results. One can notice tests with improper results only due to their absence in the figures, e.g. Figure 7.13 depicts only tests with three different seeds while there have been performed tests with six different seeds.

Summarising the observed phenomena above, the initialisation method *randomly selecting tuples* has a higher deviation in the resulting quality. That means, if we have the time for several tries, the chance to get a better result is higher for the initialisation method *randomly selecting tuples* than the method *initially clustering a sample*.

Yet, all figures show that the influence of initialisation on quality is by far greater than any other influence, in special the influence of sampling.

Figure 7.14, which contains the test series that search for two clusters, has been left out of the discussion until now because it shows no initialisation effect at all. Regardless the chosen initialisation, each test of the test series with two clusters to be found returns the same result for a sample.

The effect caused by sampling is the only remaining influence on the quality of the tests with two clusters. One can explain this phenomenon by the complexity of the clustering task: The more clusters have to be found the more local minima exist to trap a clustering algorithm. In the lower extreme case that searches for a single cluster, the result is deterministic, i.e. there are no local minima.

As Figure 7.14 shows, there is a tendency that quality improves with increasing sampling ratio. This trend is obviously subject to a statistical process with high deviation as indicated by the 5-percent sample and the 2-percent sample.

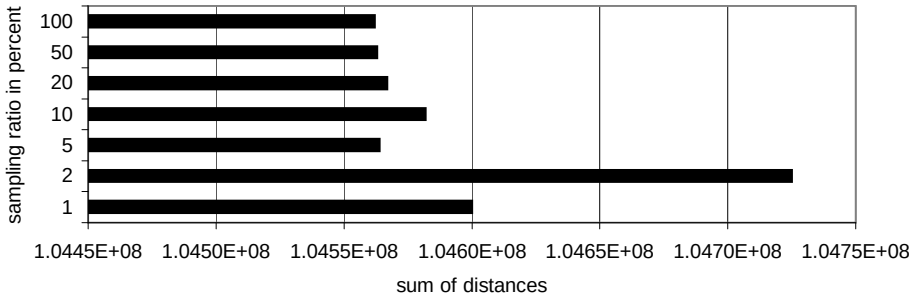


Figure 7.14: El Niño test series with two clusters

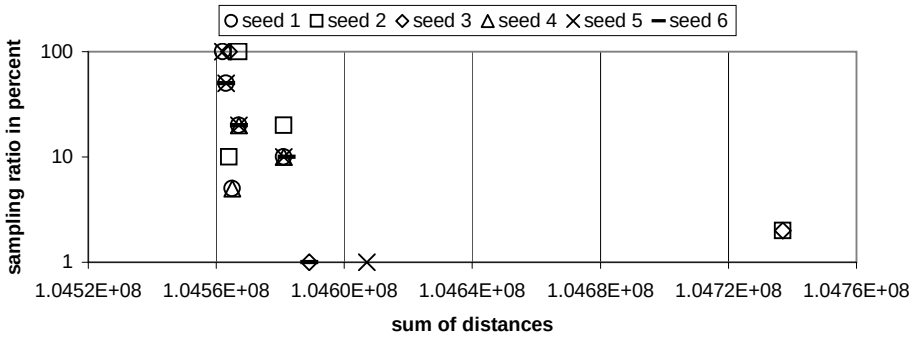


Figure 7.15: El Niño test series with three clusters without the best initialisation

When taking only those tests into account that use the initialisation method for large samples, the same trend becomes apparent. The initialisation method used for large samples has only a small deviation in the resulting quality. Thus, the initialisation method has approximately the same effect on each test within the same sample. Hence, we observe the effect of sampling on quality more clearly. Figures 7.16, 7.17, 7.18, 7.19, 7.20, 7.21, and 7.22 show the trend to better quality with rising sampling ratio.

Summarising the test series of this section, we observed that the effect of initialisation on quality can significantly outweigh the effect of other effects on quality. To be more specific, it can outweigh the effects of sampling and aggregation by far if there are local minima in the data set—which is typically the case in real world data sets. As mentioned in Section 5.10, repeatedly trying a clustering algorithm with different initialisation is very likely not to be trapped by local minima at least once. In other words, it has a high chance to find the global optimum at least once. Therefore, CHAD's initialisation method of its third phase that involves many runs of k-means with different initialisation has a good chance not to be trapped by local minima.

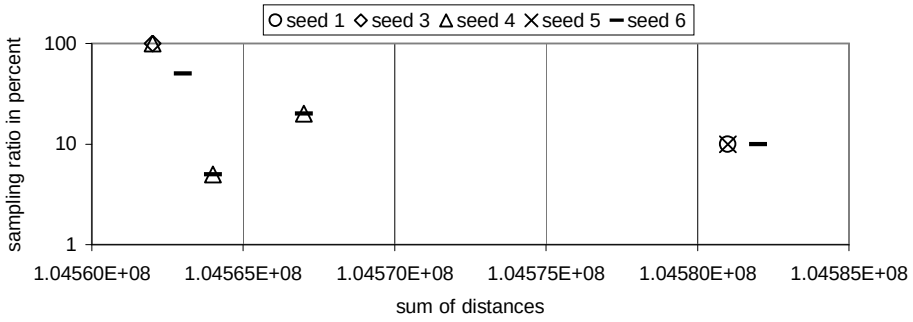


Figure 7.16: El Niño test series with four clusters and high sampling ratios

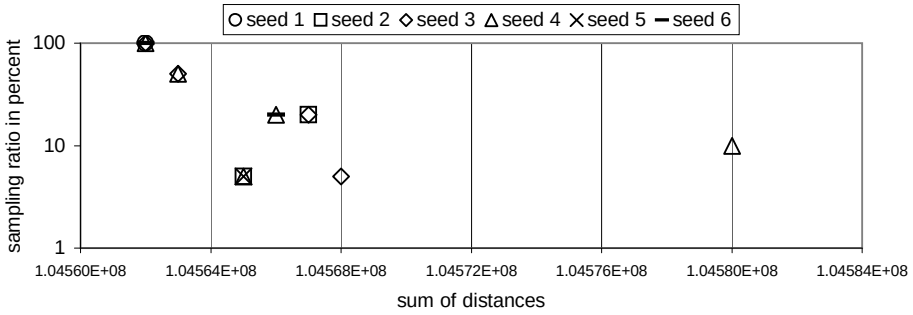


Figure 7.17: El Niño test series with five clusters and high sampling ratios

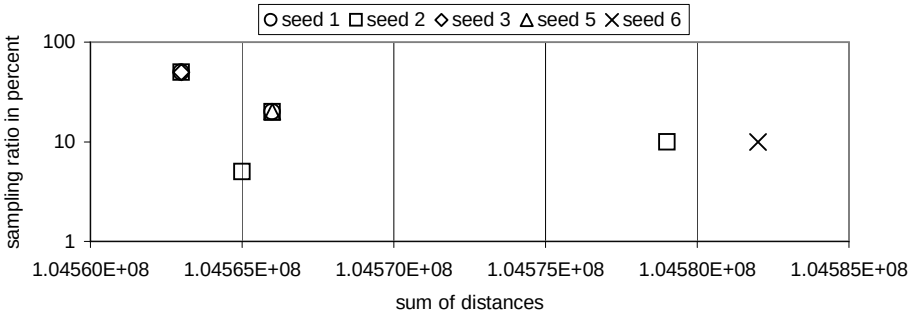


Figure 7.18: El Niño test series with six clusters and high sampling ratios

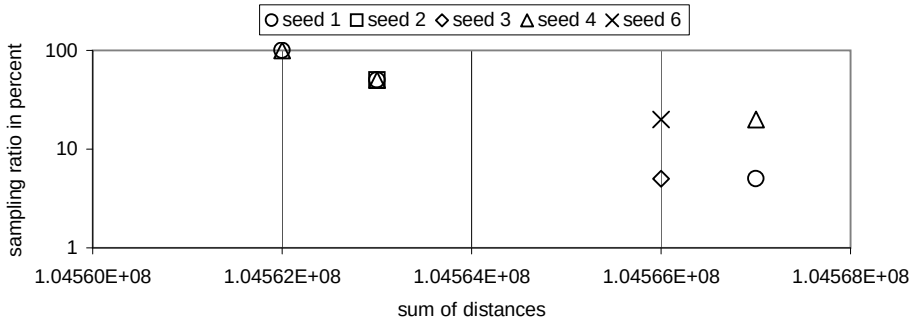


Figure 7.19: El Niño test series with seven clusters and high sampling ratios

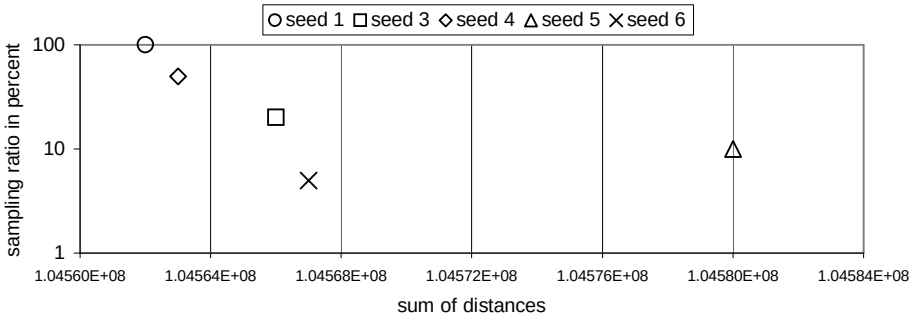


Figure 7.20: El Niño test series with eight clusters and high sampling ratios

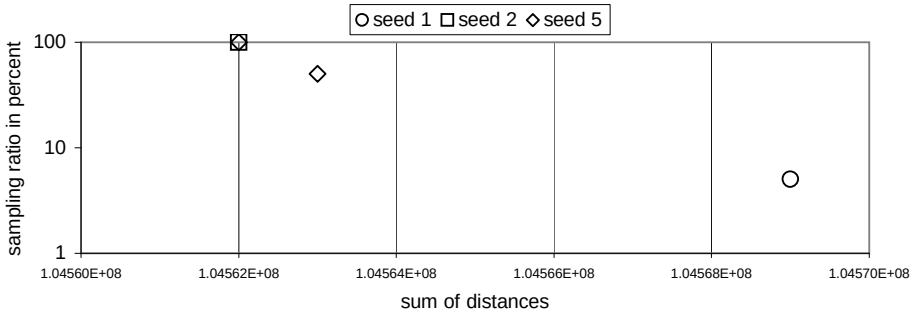


Figure 7.21: El Niño test series with nine clusters and high sampling ratios

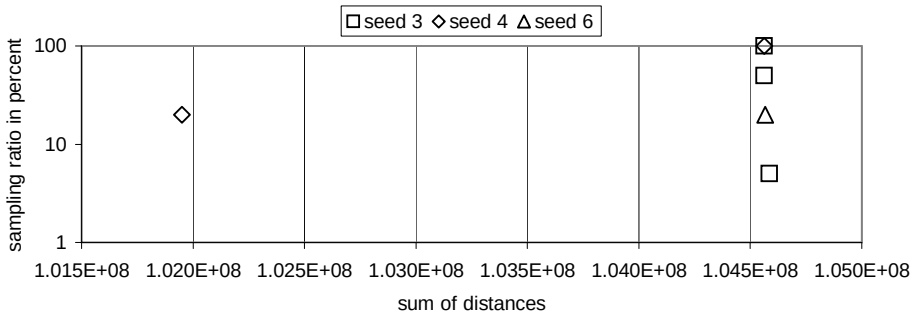


Figure 7.22: El Niño test series with ten clusters and high sampling ratios

7.5 Using Pre-computed Items as Intermediate Results or Auxiliary Data for Succeeding *KDD* Instances

The principles of anticipatory data mining are widely applicable. To be more specific, they are not limited to clustering. Thus, this section documents the results of a series of experiments that use pre-computed statistics and pre-specified specific tuples that could potentially be used as intermediate results and auxiliary data for succeeding data mining algorithms.

Classification is a major data mining technique. Therefore, a set of scenarios examines the effect of pre-computed intermediate results and auxiliary data on classification.

Subsections 7.5.1 and 7.5.2 survey a set of experiments we performed for a paper [25].

Subsection 7.5.1 discusses a set of experiments Markus Humer performed in the project of his master's thesis under the supervision of this dissertation's author. He implemented the concept of anticipatory data mining for decision tree classification. He adapted the classification algorithm Rainforest [21] to use auxiliary data consisting of auxiliary tuples and auxiliary statistics. Subsection 7.5.1 discusses the improvement on classification accuracy by using these auxiliary data.

Subsection 7.5.2 discusses a set of experiments the author of this dissertation performed for the above mentioned paper [25]. The experiments show how to construct naïve Bayes classifiers using only pre-computed intermediate results consisting of frequent pairs of attribute values. We compare the classification accuracy using these pre-computed intermediate results with the conventional way to train a naïve Bayes classifier.

7.5.1 Benefit for Decision Tree Classification

This section surveys tests to initialise the approach mentioned in Section 6.4 which uses partitioning clustering algorithm to find good split points for decision tree classification. This section also surveys tests using auxiliary tuples as

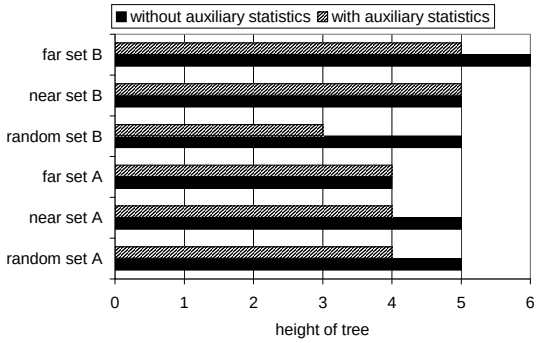


Figure 7.23: Height of decision tree

training set of decision tree classifiers to show the superiority of a so-selected training set versus randomly selected training sets.

For demonstrating the benefit of auxiliary data for decision tree classification we modified the Rainforest classification algorithm to use auxiliary data. The data set used for testing is a synthetical data set having two dozens attributes of mixed type. The class attribute has five distinct values. Ten percent of the data is noisy to make the classification task harder. Additionally, classes overlap.

In a anteceding step we applied k-means to find the best-fitting partitioning of tuples. The partitioning of tuples into k clusters returns sets of k clustering features one can use to receive a good approximate of the probability density function. Again, the minima of the probability density function are good split points of a decision tree to split numerical attributes.

We also use these partitions to select tuples from the set of auxiliary tuples that are typical representatives and outliers of the data set: We consider tuples that are near a cluster's centre as a typical representative. Analogically, we consider a tuple that is far away of a cluster's centre as an outlier. Due to their distance to their cluster's centre we call them *near* and *far* tuples.

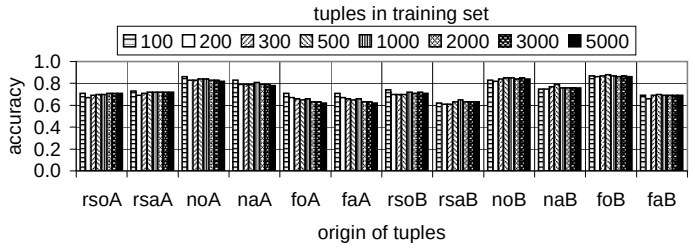
We tested using both kinds of auxiliary tuples as training set instead of selecting the training set randomly.

We used these statistics for determining split points in a succeeding run of Rainforest. After each run we compared the results of these tests with the results of the same tests without using auxiliary statistics.

Compactness of tree and accuracy are the measures we examined. Compacter trees tend to be more resistant to over-fitting, e.g. [18, p 49]. Hence, we prefer smaller trees. We measure the height of a decision tree to indicate its compactness.

Using statistics of distribution for splitting returns a tree that is lower or at maximum as high as the tree of the decision tree algorithm that uses no auxiliary statistics, as indicated in Figure 7.23.

The influence of using auxiliary statistics on accuracy is ambiguous, as shown in Figure 7.24. Some tests show equal or slightly better accuracy, others show



legend	without auxiliary statistics		with auxiliary statistics	
	data set A	data set B	data set A	data set B
random sample	rsoA	rsoB	rsaA	rsaB
near tuples	noA	noB	naA	naB
far tuples	foA	foB	faA	faB

Figure 7.24: Accuracy of decision tree classifier

sample 10 %

estimate \ actual class	false	true
false	7683	1394
true	487	97

sample 20 %

estimate \ actual class	false	true
false	7885	1449
true	321	52

sample 50 %

estimate \ actual class	false	true
false	8195	1494
true	130	25

auxiliary buffer 400

estimate \ actual class	false	true
false	8054	1461
true	140	35

auxiliary buffer 1000

estimate \ actual class	false	true
false	8124	1471
true	108	31

Table 7.6: Results of Naïve Bayes Classification in detail

worse accuracy than using no auxiliary statistics.

However, using auxiliary tuples as training set significantly influences accuracy. Figure 7.24 shows that choosing tuples of the near set is superior to choosing tuples randomly.

Considering each test series individually we observe that the number of tuples only slightly influences accuracy. Except for the test series foA and faA, accuracy is approximately constant within a test series.

Thus, selecting the training set from auxiliary tuples is more beneficial than increasing the number of tuples in the training set. We suppose that the chance that there are noisy data in the training set is smaller when we select less tuples or select tuples out of the near set.

Summarising, if one is interested in compact and accurate decision trees then selecting training data out of the near data set in combination with using statistics about distribution for splitting is a good option.

7.5.2 Benefit for Naïve Bayes Classification

For demonstrating the benefit of pre-computing frequencies of frequent combinations for naïve Bayes classification, we compared naïve Bayes classifiers using

test	S10%	S20%	S50%	aux400	aux1000
accuracy (%)	80.5	81.8	83.5	83.5	83.8
classified (%)	96.6	97.1	98.4	96.9	97.3
total accuracy (%)	77.8	79.4	82.2	80.9	81.2
buffer size				400	1000
pairs of class attribute *churn*				107	248

Table 7.7: Classification accuracy of Bayes classifier with pre-computed frequencies

a buffer of pre-computed frequencies with naïve using the traditional way of determining a naïve Bayes classifier.

We trained naïve Bayes classifiers on a real data set provided by a mobile phone company to us. We used data with demographical and usage data of mobil phone customers to predict whether a customer is about to churn or not. Most continuous and ordinal attributes such as age and sex have few distinct values. Yet, other attributes such as city have several hundreds of them. We used all non-unique categorical and ordinal attributes.

For checking the classifiers' accuracy, we reserved 20 % of available tuples or $9'999$ tuples as test data. Further, we used the remaining tuples to draw samples of different size and to store the frequencies of frequent combinations in a buffer.

For storing the top frequent pairs of attribute values, we performed two tests. In the first test, we reserved a buffer to store the frequencies of 400 pairs of attribute and attribute value combinations. The capacity of the buffer in the second test was 1000. If the buffer is full when trying to insert a new pair an item with low frequency is removed from the buffer.

To ensure that a newly inserted element of the list is not removed at the next insertion, we guarantee a minimum lifetime t_L of each element in the list. We realised the guaranty of minimum lifetime by applying the method we discussed in Subsection 6.2.2.

Estimating the class of a tuple needs the frequency of all attribute values of that tuple in combination with all values of the class attribute. If a frequency is not present, classification of that tuple is impossible.

A frequency can be unavailable because either (a) it is not part of the training set or (b) it is not part of the frequent pairs of the buffer. While option (b) can be solved by increasing the buffer size, the problem of option (a) is an immanent problem of Bayes classification.

Therefore, we used variable buffer sizes and sampling rates in our test series. We tested with a buffer with space for 400 pairs and 1000 pairs of attribute values. As the buffer with capacity of 1000 pairs became not full, we left out tests with larger buffers.

Table 7.7 contains the results of tests with sampling S10%, S20%, and S50% and results of tests with buffers as auxiliary data aux400 and aux1000 in summarised form. Table 7.6 lists these results in detail.

Table 7.7 shows that small buffers are sufficient for generating Bayes classi-

fiers with high accuracy.

Although the tests show that classification accuracy is very good when frequencies of combinations are kept in the buffer, there are few percent of tuples that cannot be classified. Thus, we split accuracy in Table 7.7 in accuracy of tuples that could be classified and accuracy of all tuples. The tests show that the buffer size influences the number of classified tuples. They also show that small buffers have a high classification ratio.

Thus, small buffers are sufficient to generate naïve Bayes classifiers having high total accuracy using exclusively intermediate results.

7.6 Summary of Evaluation

Our experiments have shown that the costs for computing intermediate results and auxiliary data are low—even, if one stores a broad range of different types of intermediate results such as the top frequent attribute value pairs and auxiliary data such as typical members of clusters to disk. In contrast to that, the benefit of intermediate results and auxiliary data is high.

We have shown that quality of clustering is higher when using clustering features instead of samples. Yet, we have also shown that the effect of sampling as well as the effect of using clustering features on the result of a k-means cluster analysis is negligible when compared with the effect of initialisation on a k-means cluster analysis. Hence, finding a good initialisation is crucial for analysing clusters.

Yet, anticipatory data mining is ideally suited to support an analyst finding the optimal clustering solution because

- it is so fast that an analyst can iteratively analyse data in order to find an optimally fitting clustering,
- it also enables checking the quality of results which is impossible in approaches based on sampling.

Therefore, we can automate the process of finding the optimal solution by iteratively creating a potential solution and testing it.

We have shown that classification algorithms using auxiliary data compute classifiers having higher quality than those algorithms using no auxiliary data. To be more specific, decision tree algorithms find compacter decision trees when using auxiliary statistics for splitting numerical attributes. Moreover, selecting auxiliary tuples as training sets of a classification algorithm increases the accuracy of classifiers. Finally, one can use auxiliary statistics to derive highly accurate Bayes classifiers for free.

Summarising the statements above, pre-computing intermediate results and auxiliary data is a cheap option with high potential improvement of runtime or quality—or both.

Appendix A

Description of Tests

A.1 data sets

data set D_α												
										number of clusters		4
										number of dimensions		10
										number of tuples		5,000,000
										percentage of noise		0 %
clusters												
clus.	$\vec{\mu_{x,1}}$	$\vec{\mu_{x,2}}$	$\vec{\mu_{x,3}}$	$\vec{\mu_{x,4}}$	$\vec{\mu_{x,5}}$	$\vec{\mu_{x,6}}$	$\vec{\mu_{x,7}}$	$\vec{\mu_{x,8}}$	$\vec{\mu_{x,9}}$	$\vec{\mu_{x,10}}$	dev.	perc.
$\vec{\mu_1}$	1	-0.7	0	1	0	-1	1	-1	0	1	0.25	20 %
$\vec{\mu_2}$	1	1	-1	-1	0	0	-1	1	0	-1	0.25	20 %
$\vec{\mu_3}$	-0.8	-1	1	0	0	1	0	0	1	1	0.25	20 %
$\vec{\mu_4}$	-1	1	0	0	0	-1	-1	-1	0	-1	0.25	20 %

Table A.1: parameters of synthetical data set D_α

data set D_β												
number of clusters												5
number of dimensions												10
number of tuples												5,000,000
percentage of noise												10 %
clusters												
clus.	$\vec{\mu_{x,1}}$	$\vec{\mu_{x,2}}$	$\vec{\mu_{x,3}}$	$\vec{\mu_{x,4}}$	$\vec{\mu_{x,5}}$	$\vec{\mu_{x,6}}$	$\vec{\mu_{x,7}}$	$\vec{\mu_{x,8}}$	$\vec{\mu_{x,9}}$	$\vec{\mu_{x,10}}$	dev.	perc.
$\vec{\mu_1}$	1	1	2	0	1	1	0	0	0	0	0.25	20 %
$\vec{\mu_2}$	-1	2	1	0	-1	1.2	-1	-1	0	1	0.25	20 %
$\vec{\mu_3}$	0	-1	-2	0	0	3	1.2	-2	0	0	0.25	20 %
$\vec{\mu_4}$	2	-1.2	1	1	1.5	-2	3	1	0	0	0.25	20 %
$\vec{\mu_5}$	-2	0	-1	2	2	-1.8	5	1.1	0	1.5	0.25	20 %
Note: percentages of data in clusters refer to non-noisy data only, so they sum up to 100%. Assuming the percentage of noise is 10%, a percentage of 20% of a cluster denotes 20% in the remaining 90%—which is only a percentage of 18% in the entire data set. Noise is uniformly distributed between the lower left corner and the upper right corner of the bounding rectangle that contains all non-noisy data. In test set D_β the according corners are defined by the points $(-3,-3,-3,-3,-3,-3,-3,-3,-3,-3)^T$ and $(3,3,3,3,3,3,3,3,3,3)^T$												

Table A.2: parameters of synthetical data set D_β

data set D_γ												
number of clusters												5
number of dimensions												10
number of tuples												5,000,000
percentage of noise												30 %
clusters												
clus.	$\vec{\mu_{x,1}}$	$\vec{\mu_{x,2}}$	$\vec{\mu_{x,3}}$	$\vec{\mu_{x,4}}$	$\vec{\mu_{x,5}}$	$\vec{\mu_{x,6}}$	$\vec{\mu_{x,7}}$	$\vec{\mu_{x,8}}$	$\vec{\mu_{x,9}}$	$\vec{\mu_{x,10}}$	dev.	perc.
$\vec{\mu_1}$	1	1	2	0	1	1	0	0	0	0	0.25	20 %
$\vec{\mu_2}$	-1	2	1	0	-1	1.2	-1	-1	0	1	0.25	20 %
$\vec{\mu_3}$	0	-1	-2	0	0	3	1.2	-2	0	0	0.25	20 %
$\vec{\mu_4}$	2	-1.2	1	1	1.5	-2	3	1	0	0	0.25	20 %
$\vec{\mu_5}$	-2	0	-1	2	2	-1.8	5	1.1	0	1.5	0.25	20 %
Note: percentages of data in clusters refer to non-noisy data only, so they sum up to 100%. Assuming the percentage of noise is 30%, a percentage of 20% of a cluster denotes 20% in the remaining 70%—which is only a percentage of 14% in the entire data set. Noise is uniformly distributed between the lower left corner and the upper right corner of the bounding rectangle that contains all non-noisy data. In test set D_γ the according corners are defined by the points $(-3,-3,-3,-3,-3,-3,-3,-3,-3,-3)^T$ and $(3,3,3,3,3,3,3,3,3,3)^T$												

Table A.3: parameters of synthetical data set D_γ

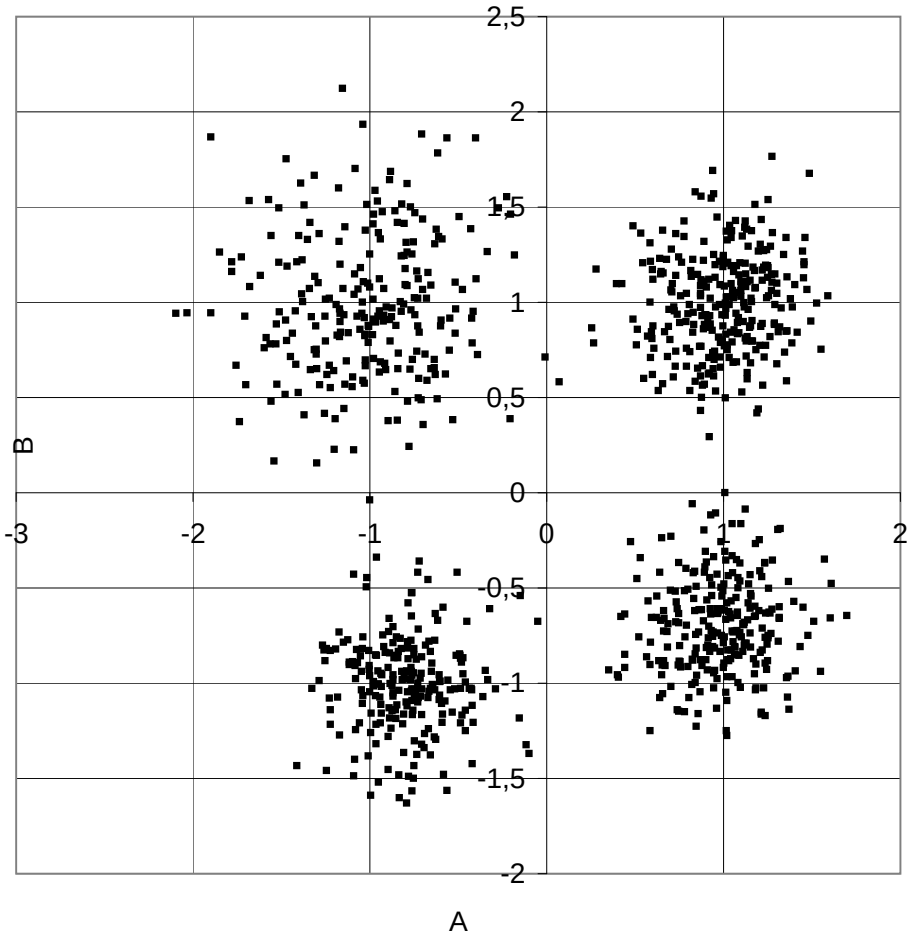


Figure A.1: first 1000 tuples of test set D_α shown in the first two dimensions

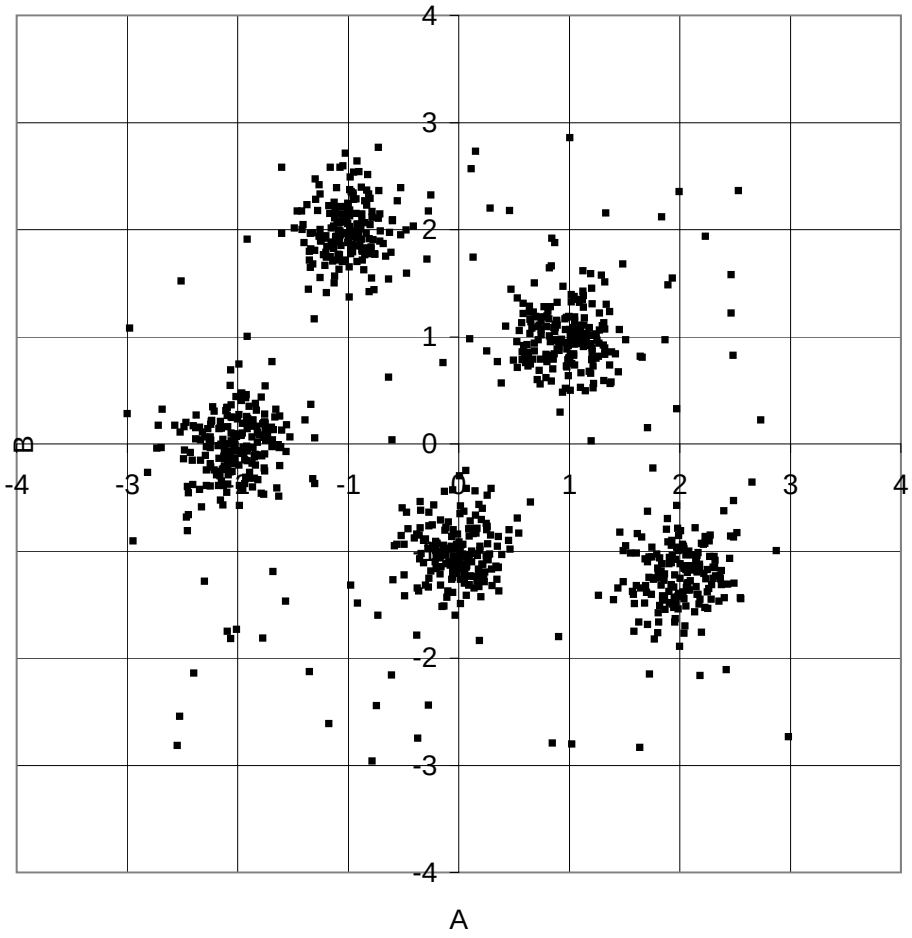


Figure A.2: first 1000 tuples of test set D_β shown in the first two dimensions

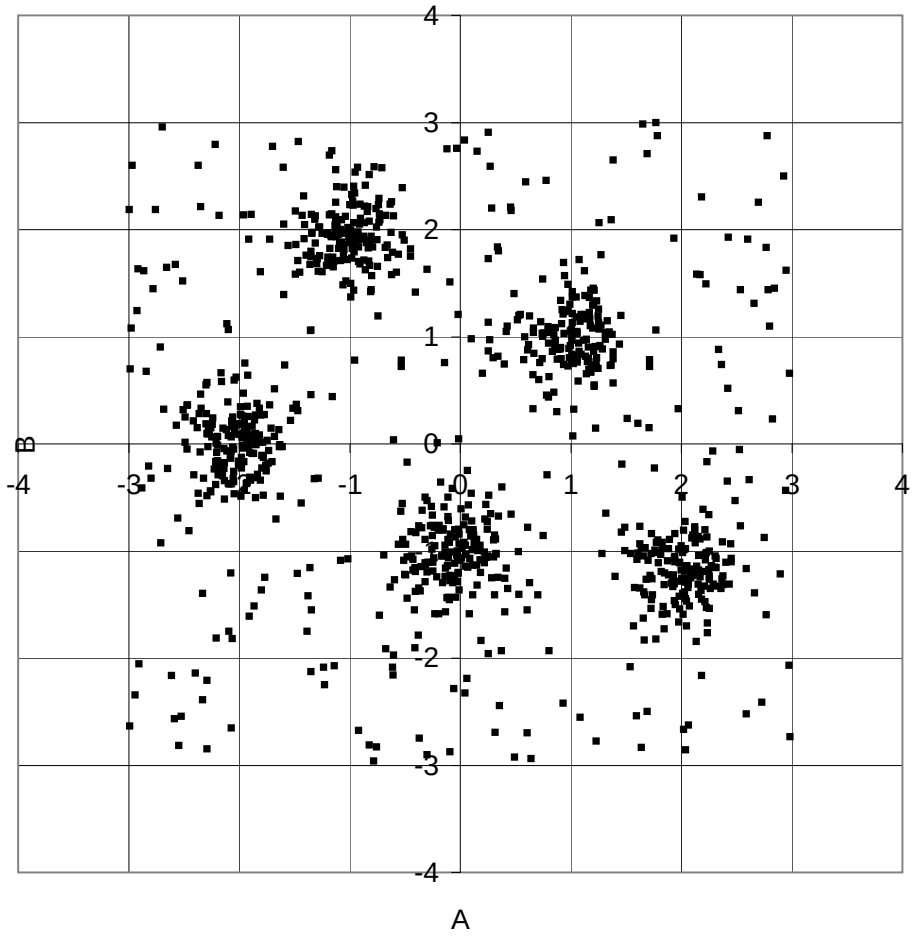


Figure A.3: first 1000 tuples of test set D_γ shown in the first two dimensions

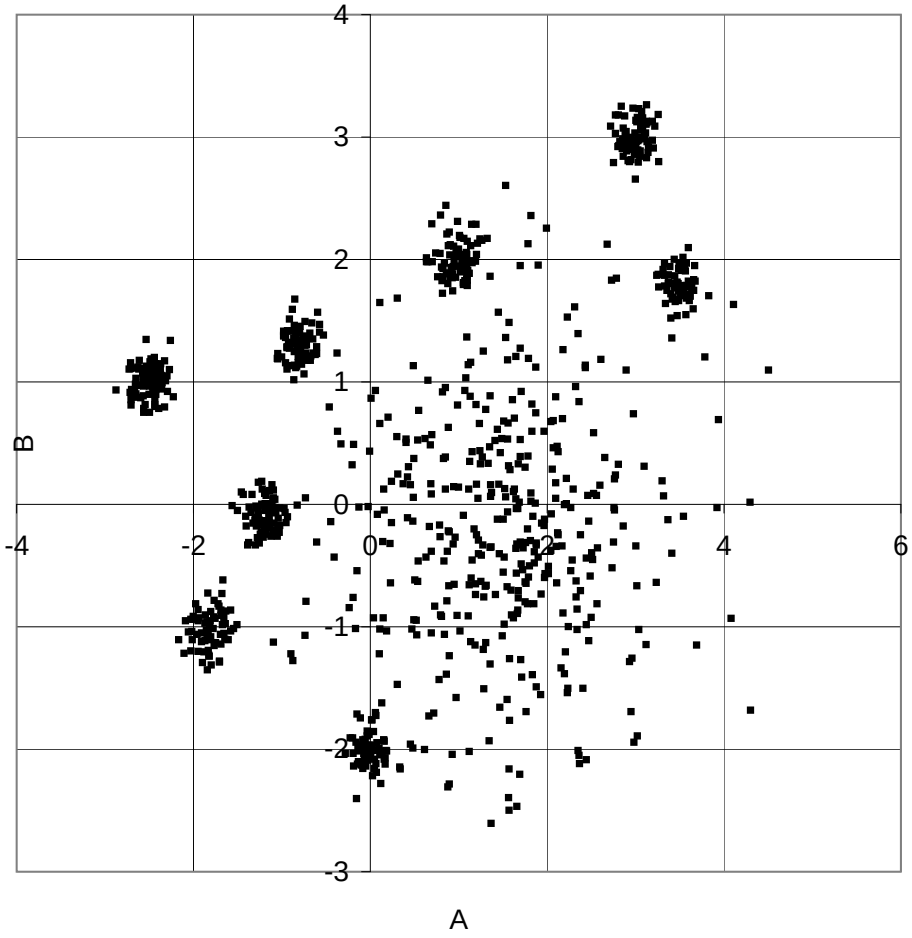


Figure A.4: first 1000 tuples of test set D_δ shown in the first two dimensions

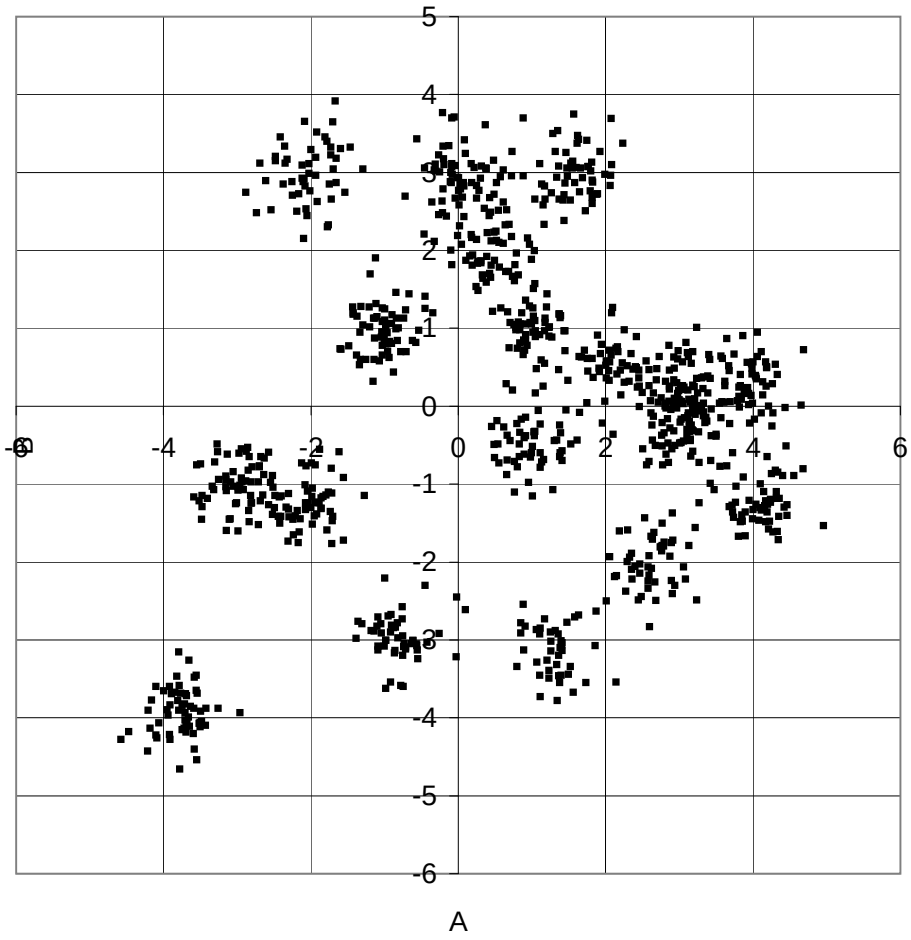


Figure A.5: first 1000 tuples of test set D_ϵ shown in the first two dimensions

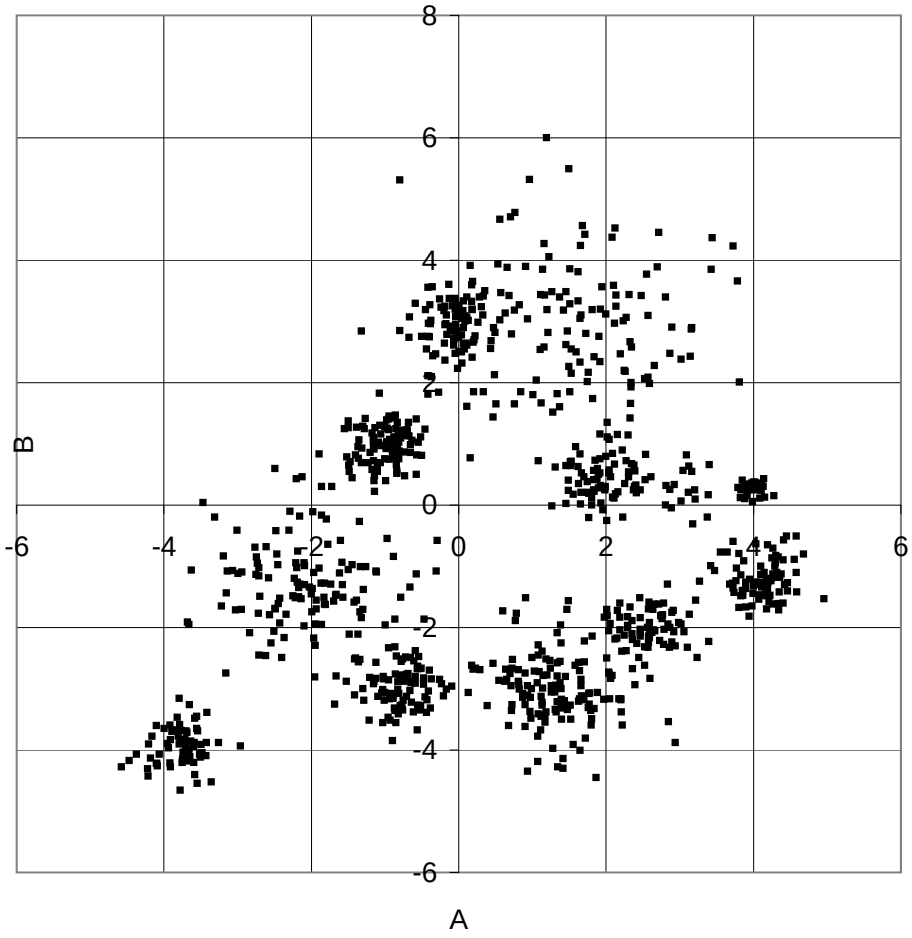


Figure A.6: first 1000 tuples of test set D_ζ shown in the first two dimensions

data set D_δ												
number of clusters												9
number of dimensions												10
number of tuples												5,000,000
percentage of noise												0 %
clusters												
clus.	$\vec{\mu_{x,1}}$	$\vec{\mu_{x,2}}$	$\vec{\mu_{x,3}}$	$\vec{\mu_{x,4}}$	$\vec{\mu_{x,5}}$	$\vec{\mu_{x,6}}$	$\vec{\mu_{x,7}}$	$\vec{\mu_{x,8}}$	$\vec{\mu_{x,9}}$	$\vec{\mu_{x,10}}$	dev.	perc.
$\vec{\mu_1}$	1.5	-0.2	0	0.1	-2	1.5	1	-1	0	0	1.0	40 %
$\vec{\mu_2}$	0	-2	3	0.6	-0.5	0	2	-1	1.2	-1	0.125	7.5 %
$\vec{\mu_3}$	-1.8	-1	-3	0.8	1	1.5	1	0	3	1	0.125	7.5 %
$\vec{\mu_4}$	-1.2	-0.1	2.7	2	2.8	1.4	2	0	1	3	0.125	7.5 %
$\vec{\mu_5}$	-2.5	1	-2.7	-0.8	2	0.3	1	1	-1	4	0.125	7.5 %
$\vec{\mu_6}$	-0.8	1.3	2.5	2.5	3.2	0	2	1	-1.3	3.2	0.125	7.5 %
$\vec{\mu_7}$	1	2	1	2	4	1	0	0	-1.9	-1.1	0.125	7.5 %
$\vec{\mu_8}$	3	3	-0.8	3	5.2	0.5	0	-1	0	0	0.125	7.5 %
$\vec{\mu_9}$	3.5	1.8	-2	2	6	0	0	1	0	-3	0.125	7.5 %

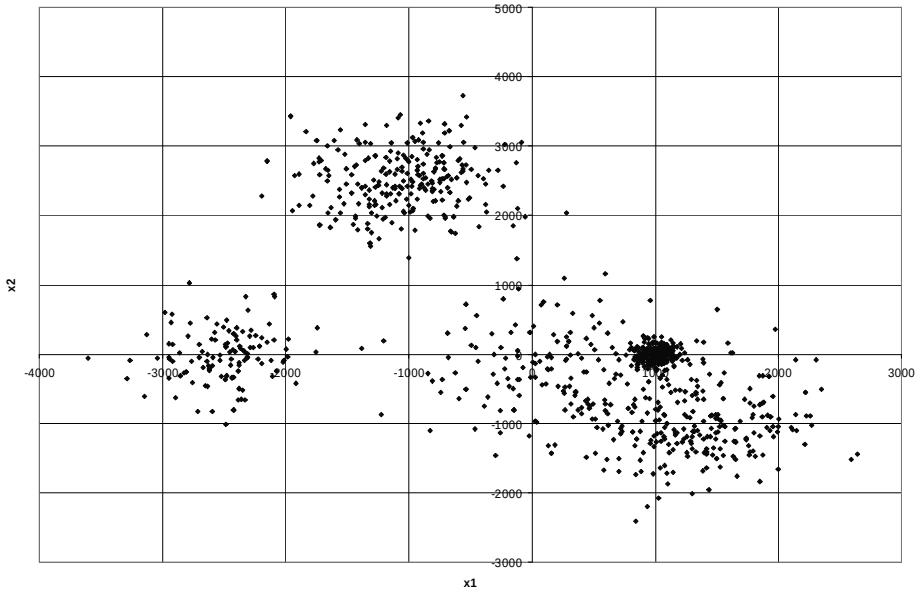
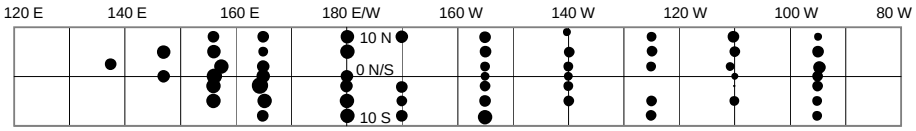


Figure A.7: first 1000 tuples of test set SHF shown in the most-discriminating dimensions

data set D_ϵ												
										number of clusters		18
										number of dimensions		10
										number of tuples		5,000,000
										percentage of noise		0 %
location, deviation and percentage of clusters												
clus.	$\vec{\mu_{x,1}}$	$\vec{\mu_{x,2}}$	$\vec{\mu_{x,3}}$	$\vec{\mu_{x,4}}$	$\vec{\mu_{x,5}}$	$\vec{\mu_{x,6}}$	$\vec{\mu_{x,7}}$	$\vec{\mu_{x,8}}$	$\vec{\mu_{x,9}}$	$\vec{\mu_{x,10}}$	dev.	perc.
$\vec{\mu_1}$	-3.8	-4	1.8	-2.2	-0.4	-0.7	-1.5	0.4	2.7	0.3	0.3	5.5 %
$\vec{\mu_2}$	-2	-1.2	-0.2	-2	1.5	5	-0.4	0.8	2.5	1.5	0.3	5.5 %
$\vec{\mu_3}$	0	3	1.9	0.2	1.7	-3.9	2	-3	-1	2.3	0.3	5.5 %
$\vec{\mu_4}$	4	0.3	0.2	1	-2.4	-1.4	1	-1.9	-2	1.6	0.3	5.5 %
$\vec{\mu_5}$	1.3	-3.1	0	-0.8	-2.8	1.3	-0.7	-2	0.3	4.7	0.3	5.5 %
$\vec{\mu_6}$	2	0.5	1.8	1.7	2.5	-2.2	-1.2	-2.7	-0.6	3.8	0.3	5.5 %
$\vec{\mu_7}$	-0.8	-3	-1.1	-1	1	-1.5	-1.5	-0.3	0.3	5.3	0.3	5.5 %
$\vec{\mu_8}$	1	1	1.2	4.2	-3.5	0.9	4	2.5	4	-2.3	0.3	5.5 %
$\vec{\mu_9}$	1	-0.4	2.7	1.3	-2.5	4	3.7	5	-1.3	4.2	0.3	5.5 %
$\vec{\mu_{10}}$	-2	3	-0.3	2	3	-0.8	-1	3	2.5	-0.6	0.3	5.5 %
$\vec{\mu_{11}}$	0.4	2	-1.1	1.7	3.3	0.9	-2	-2.3	-0.8	-0.3	0.3	5.5 %
$\vec{\mu_{12}}$	3	-0.2	2.4	-0.2	1	2	0.5	0.3	1.4	-1	0.3	5.5 %
$\vec{\mu_{13}}$	-3	-1	-2.5	0.5	5	2.4	4.2	-0.2	1	-3	0.3	5.5 %
$\vec{\mu_{14}}$	-1	1	3.2	-2	1.8	-0.4	2.1	1.3	-2	-3	0.3	5.5 %
$\vec{\mu_{15}}$	1.6	3	-1.7	3	3.6	-2.6	2.9	-1	3	-2.7	0.3	5.5 %
$\vec{\mu_{16}}$	4.1	-1.2	-2.2	-3.1	-0.3	4	2.6	3	-2.9	1.8	0.3	5.5 %
$\vec{\mu_{17}}$	2.6	-2	1	4.5	-0.3	-0.4	-3	-2.3	2	2	0.3	5.5 %
$\vec{\mu_{18}}$	3	0.3	-2.7	0	-1.2	-3.2	1.3	4.1	-4	-1.3	0.3	6.5 %

Table A.5: parameters of synthetical data set D_ϵ



The radius of a point denotes the air temperature at that location. The larger the radius the higher is the temperature.

Figure A.8: air temperature and position of buoy of one day in the El Niño data set

data set D_ζ												
										number of clusters		12
										number of dimensions		10
										number of tuples		5,000,000
										percentage of noise		0 %
location, deviation and percentage of clusters												
clus.	$\vec{\mu_{x,1}}$	$\vec{\mu_{x,2}}$	$\vec{\mu_{x,3}}$	$\vec{\mu_{x,4}}$	$\vec{\mu_{x,5}}$	$\vec{\mu_{x,6}}$	$\vec{\mu_{x,7}}$	$\vec{\mu_{x,8}}$	$\vec{\mu_{x,9}}$	$\vec{\mu_{x,10}}$	dev.	perc.
$\vec{\mu_1}$	-3.8	-4	1.8	-2.2	-0.4	-0.7	-1.5	0.4	2.7	0.3	0.3	6.3 %
$\vec{\mu_2}$	-2	-1.2	-0.2	-2	1.5	5	-0.4	0.8	2.5	1.5	0.7	12 %
$\vec{\mu_3}$	0	3	1.9	0.2	1.7	-3.9	2	-3	-1	2.3	0.3	6.3 %
$\vec{\mu_4}$	4	0.3	0.2	1	-2.4	-1.4	1	-1.9	-2	1.6	0.1	8.3 %
$\vec{\mu_5}$	1.3	-3.1	0	-0.8	-2.8	1.3	-0.7	-2	0.3	4.7	0.6	3.3 %
$\vec{\mu_6}$	2	0.5	1.8	1.7	2.5	-2.2	-1.2	-2.7	-0.6	3.8	0.3	15 %
$\vec{\mu_7}$	-0.8	-3	-1.1	-1	1	-1.5	-1.5	-0.3	0.3	5.3	0.3	8.3 %
$\vec{\mu_8}$	-1	1	3.2	-2	1.8	-0.4	2.1	1.3	-2	-3	0.3	8.3 %
$\vec{\mu_9}$	1.6	3	-1.7	3	3.6	-2.6	2.9	-1	3	-2.7	1.0	12 %
$\vec{\mu_{10}}$	4.1	-1.2	-2.2	-3.1	-0.3	4	2.6	3	-2.9	1.8	0.3	8.3 %
$\vec{\mu_{11}}$	2.6	-2	1	4.5	-0.3	-0.4	-3	-2.3	2	2	0.3	8.3 %
$\vec{\mu_{12}}$	3	0.3	-2.7	0	-1.2	-3.2	1.3	4.1	-4	-1.3	0.3	1.6 %

Table A.6: parameters of synthetical data set D_ζ

attribute	*description*
obc	**ob**servation **c**ount, an additionally inserted unique identifier to serve as primary key—is not part of the original data set in [31]
date	day of measurement in the form YYYYMMDD
latitude	geographical latitude, positive value denotes that measure was taken in the northern hemisphere
longitude	geographical longitude, positive value denote coordinate is in the East
zonwinds	speed of zonal winds, zonal winds with positive speed blow from west to east
merwinds	speed of meridional winds, meridional winds with positive speed blow from south to north
humidity	measured humidity in percent, i.e. range is $[0, 100]$
airtemp	air temperature in degrees Celsius
subseatemp	average temperature under water in degrees Celsius

Table A.7: schema of the El Niño data set

test name	dimensions	nodes	tuples
C_1	10	100	200,000
C_2	10	200	200,000
C_3	10	400	200,000
C_4	10	800	200,000
C_5	10	1,600	200,000
C_6	10	3,200	200,000
C_7	10	6,400	200,000
C_8	10	12,800	200,000
C_9	10	25,600	200,000
C_{10}	10	100	1,000,000
C_{11}	10	200	1,000,000
C_{12}	10	400	1,000,000
C_{13}	10	800	1,000,000
C_{14}	10	1,600	1,000,000
C_{15}	10	3,200	1,000,000
C_{16}	10	6,400	1,000,000
C_{17}	10	12,800	1,000,000
C_{18}	10	25,600	1,000,000
C_{19}	10	51,200	1,000,000
C_{20}	10	102,400	1,000,000
C_{21}	10	25,600	400,000
C_{22}	10	25,600	600,000
C_{23}	10	25,600	800,000
C_{24}	10	25,600	2,000,000
C_{25}	10	25,600	3,000,000
C_{26}	10	25,600	4,000,000
C_{27}	10	25,600	5,000,000

Table A.8: test series C

A.2 test series

Each test has been executed six times, once per data set $D_\alpha, D_\beta, D_\gamma, D_\delta, D_\epsilon, D_\zeta$ and each time with different seed.

machine I	machine II	machine III	machine IV	machine V	machine VI	block
αC_{27}	βC_{27}	γC_{27}	δC_{27}	ϵC_{27}	ζC_{27}	C1
βC_{26}	γC_{26}	δC_{26}	ϵC_{26}	ζC_{26}	αC_{26}	C2
βC_{20}	γC_{20}	δC_{20}	ϵC_{20}	ζC_{20}	αC_{20}	
γC_{25}	δC_{25}	ϵC_{25}	ζC_{25}	αC_{25}	βC_{25}	C3
γC_{24}	δC_{24}	ϵC_{24}	ζC_{24}	αC_{24}	βC_{24}	
δC_{23}	ϵC_{23}	ζC_{23}	αC_{23}	βC_{23}	γC_{23}	C4
δC_{22}	ϵC_{22}	ζC_{22}	αC_{22}	βC_{22}	γC_{22}	
δC_{21}	ϵC_{21}	ζC_{21}	αC_{21}	βC_{21}	γC_{21}	
δC_{9}	ϵC_{9}	ζC_{9}	αC_{9}	βC_{9}	γC_{9}	
δC_{8}	ϵC_{8}	ζC_{8}	αC_{8}	βC_{8}	γC_{8}	
δC_{7}	ϵC_{7}	ζC_{7}	αC_{7}	βC_{7}	γC_{7}	
δC_{6}	ϵC_{6}	ζC_{6}	αC_{6}	βC_{6}	γC_{6}	
δC_{5}	ϵC_{5}	ζC_{5}	αC_{5}	βC_{5}	γC_{5}	
δC_{4}	ϵC_{4}	ζC_{4}	αC_{4}	βC_{4}	γC_{4}	
δC_{3}	ϵC_{3}	ζC_{3}	αC_{3}	βC_{3}	γC_{3}	
δC_{2}	ϵC_{2}	ζC_{2}	αC_{2}	βC_{2}	γC_{2}	
δC_{1}	ϵC_{1}	ζC_{1}	αC_{1}	βC_{1}	γC_{1}	
ϵC_{19}	ζC_{19}	αC_{19}	βC_{19}	γC_{19}	δC_{19}	C5
ϵC_{18}	ζC_{18}	αC_{18}	βC_{18}	γC_{18}	δC_{18}	
ϵC_{17}	ζC_{17}	αC_{17}	βC_{17}	γC_{17}	δC_{17}	
ϵC_{16}	ζC_{16}	αC_{16}	βC_{16}	γC_{16}	δC_{16}	
ϵC_{15}	ζC_{15}	αC_{15}	βC_{15}	γC_{15}	δC_{15}	
ζC_{14}	αC_{14}	βC_{14}	γC_{14}	δC_{14}	ϵC_{14}	C6
ζC_{13}	αC_{13}	βC_{13}	γC_{13}	δC_{13}	ϵC_{13}	
ζC_{12}	αC_{12}	βC_{12}	γC_{12}	δC_{12}	ϵC_{12}	
ζC_{11}	αC_{11}	βC_{11}	γC_{11}	δC_{11}	ϵC_{11}	
ζC_{10}	αC_{10}	βC_{10}	γC_{10}	δC_{10}	ϵC_{10}	

Meaning of items: αC_9: test C_9 was tested with data set D_α. Details of test C_9 can be found in Table A.8.

Table A.9: allocation of test series C on available machines

	sequence
machine I	C1,C2,C3,C4,C5,C6
machine II	C1,C2,C3,C4,C5,C6
machine III	C1,C2,C3,C4,C5,C6
machine IV	C1,C2,C3,C4,C5,C6
machine V	C1,C2,C3,C4,C5,C6
machine VI	C1,C2,C3,C4,C5,C6

Table A.10: schedule of machines

C1	α	β	γ	δ	ϵ	average
C_{27}	605.640	674.672	697.542	642.701	501.610	624.433

Table A.11: runtime of tests of block C1 in seconds

C2	β	γ	δ	ϵ	ζ	average
C_{26}	593.125	562.328	547.753	388.840	511.787	520.767
C_{20}	175.578	248.031	261.502	302.829	915.561	380.700

Table A.12: runtime of tests of block C2 in seconds

A.3 results in detail

C3	γ	δ	ϵ	ζ	α	average
C_{25}	496.625	461.009	316.327	415.057	377.701	413.344
C_{24}	373.735	370.685	226.457	328.343	252.971	310.438

Table A.13: runtime of tests of block C3 in seconds

C4	δ	ϵ	ζ	α	β	average
C_{23}	232.937	116.947	200.724	116.573	118.276	157.091
C_{22}	183.906	93.136	147.415	81.386	95.936	120.356
C_{21}	155.656	66.028	122.447	55.607	58.021	91.552
C_9	36.047	33.139	73.402	33.608	32.760	41.791
C_8	55.469	33.374	61.107	27.858	26.417	40.845
C_7	42.657	27.499	44.795	25.733	26.870	33.511
C_6	29.078	21.437	38.061	21.061	28.417	27.611
C_5	22.000	21.374	25.999	17.733	23.199	22.061
C_4	17.516	23.124	22.296	15.234	19.168	19.468
C_3	12.000	12.406	17.499	14.155	14.170	14.046
C_2	10.797	11.827	17.375	13.031	13.044	13.215
C_1	8.391	12.609	16.124	12.249	11.639	12.202

Table A.14: runtime of tests of block C4 in seconds

C5	ϵ	ζ	α	β	γ	average
C_{19}	178.812	298.938	144.524	146.680	369.873	227.765
C_{18}	134.922	203.068	124.713	156.804	229.961	169.894
C_{17}	106.641	153.742	124.306	164.523	170.782	143.999
C_{16}	88.203	131.056	103.917	123.353	124.500	114.206
C_{15}	80.109	93.979	86.090	98.839	89.594	89.722

Table A.15: runtime of tests of block C5 in seconds

C6	ζ	α	β	γ	δ	average
C_{14}	101.750	73.043	86.652	88.480	69.828	83.951
C_{13}	78.234	79.965	51.388	66.168	55.234	66.198
C_{12}	75.719	65.122	43.044	70.184	74.078	65.629
C_{11}	50.828	66.152	47.639	62.169	66.328	58.623
C_{10}	45.187	57.810	59.731	66.997	38.516	53.648

Table A.16: runtime of tests of block C6 in seconds

Bibliography

[1] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In Johann Christoph Freytag, Peter C. Lockemann, Serge Abiteboul, Michael J. Carey, Patricia G. Selinger, and Andreas Heuer, editors, *VLDB 2003: Proceedings of 29th International Conference on Very Large Data Bases, September 9–12, 2003, Berlin, Germany*, pages 81–92, Los Altos, CA 94022, USA, 2003. Morgan Kaufmann Publishers.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[3] Ibraheem Al-Furaih, Srinivas Aluru, Sanjay Goil, and Sanjay Ranka. Parallel construction of multidimensional binary search trees. In *ICS '96: Proceedings of the 10th international conference on Supercomputing*, pages 205–212, New York, NY, USA, 1996. ACM Press.

[4] K. Alsabti, S. Ranka, and V. Singh. An efficient k-means clustering algorithm. In *Proc. First Workshop on High-Performance Data Mining*, 1998.

[5] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 49–60, New York, NY, USA, 1999. ACM Press.

[6] Paul S. Bradley, Usama M. Fayyad, and Cory Reina. Scaling clustering algorithms to large databases. In *Knowledge Discovery and Data Mining*, pages 9–15, 1998.

[7] Stefan Brecheisen, Hans-Peter Kriegel, and Martin Pfeifle. Multi-step density-based clustering. *Knowl. Inf. Syst.*, 9(3):284–308, 2006.

[8] Doina Caragea. *Learning Classifiers from Distributed, Semantically Heterogeneous, Autonomous Data Sources*. PhD thesis, Iowa State University, 2004.

[9] Tom Chiu, DongPing Fang, John Chen, Yao Wang, and Christopher Jeris. A robust and scalable clustering algorithm for mixed type attributes in large database environment. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 263–268. ACM Press, 2001.

[10] Alfredo Cuzzocrea. Providing probabilistically-bounded approximate answers to non-holistic aggregate range queries in olap. In *DOLAP '05: Proceedings of the 8th ACM international workshop on Data warehousing and OLAP*, pages 97–106, New York, NY, USA, 2005. ACM Press.

[11] Antonios Deligiannakis, Yannis Kotidis, and Nick Roussopoulos. Compressing historical information in sensor networks. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 527–538, New York, NY, USA, 2004. ACM Press.

[12] A. P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood via the EM algorithm. *Journal of the Royal Statistical Society*, (39):1–38, 1977.

[13] Alin Dobra and Johannes Gehrke. Secret: a scalable linear regression tree algorithm. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 481–487, New York, NY, USA, 2002. ACM Press.

[14] William DuMouchel and Daryl Pregibon. Empirical bayes screening for multi-item associations. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 67–76, New York, NY, USA, 2001. ACM Press.

[15] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.

[16] Martin Ester and Joerg Sander. *Knowledge Discovery in Databases*. Springer Verlag, Heidelberg, 2000.

[17] Edward W. Forgy. Cluster analysis of multi-variate data: Efficiency vs. interpretability of classifications. *Biometrics*, 21:768–769, 1965.

[18] Alex. A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Spinger-Verlag, Berlin, 2002.

[19] Dominik Fürst. Effizientes k-clustering: K-centroids clustering auf basis von hierarchisch aggregierten daten. Master's thesis, Institut für Wirtschaftsinformatik, Abteilung Data & Knowledge Engineering, 2004.

[20] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Mining data streams under block evolution. *SIGKDD Explor. Newsl.*, 3(2):1–10, 2002.

[21] Johannes Gehrke, Raghu Ramakrishnan, and Venkatesh Ganti. Rainforest - a framework for fast decision tree construction of large datasets. *Data Mining and Knowledge Discovery*, 4(2/3):127–162, 2000.

[22] Corrado Gini. Variabilità e mutabilità. *Studi Economico-Giuridici della Facoltà di Giurisprudenza dell'Università di Cagliari*, 3:3–159, 1912. Reprinted, with few variations, in C. Gini, Memorie di metologia statistica, vol. I, Veschi, Roma, 1955, pp. 211-282.

[23] Clark Glymour, David Madigan, Daryl Pregibon, and Padhraic Smyth. Statistical themes and lessons for data mining. *Data Min. Knowl. Discov.*, 1(1):11–28, 1997.

[24] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. The dimensional fact model: A conceptual model for data warehouses. *International Journal of Cooperative Information Systems*, 7(2-3):215–247, 1998.

[25] Mathias Goller, Markus Humer, and Michael Schrefl. Beneficial sequential combination of data mining algorithms. In *Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS 2006)*, 2006.

[26] Mathias Goller and Michael Schrefl. Anticipatory clustering. In *Proceedings of the IASTED International Conference on Databases and Applications 2004, Innsbruck, Austria, February 17 - 19*. Acta Press, Calgary, Canada, 2004.

[27] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15, 2003.

[28] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: an efficient clustering algorithm for large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 73–84, June 1998.

[29] Greg Hamerly and Charles Elkan. Alternatives to the k-means algorithm that find better clusterings. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 600–607. ACM Press, 2002.

[30] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In Weidong Chen, Jeffrey Naughton, and Philip A. Bernstein, editors, *2000 ACM SIGMOD Intl. Conference on Management of Data*, pages 1–12. ACM Press, 05 2000.

[31] S. Hettich and S. D. Bay. *The UCI KDD Archive [http://kdd.ics.uci.edu]*. University of California, Department of Information and Computer Science., Irvine, CA, 1999.

[32] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):45–73, 2004.

[33] Jochen Hipp and Ulrich Güntzer. Is pushing constraints deeply into the mining algorithms really what we want?: an alternative approach for association rule mining. *ACM SIGKDD Explorations Newsletter*, 4(1):50–55, 2002.

[34] Zhexue Huang. A fast clustering algorithm to cluster very large categorical data sets in data mining. In *Research Issues on Data Mining and Knowledge Discovery*, pages 0–, 1997.

[35] Markus Humer. Kombiniertes data mining - klassifikation unter verwendung von durch clustering gewonnenen hilfsinformationen. Master's thesis, Institut für Wirtschaftsinformatik, Abteilung Data & Knowledge Engineering, 2004.

[36] Micheline Kamber Jiawei Han. *Data Mining: Concepts an Techniques*. Morgan Kaufmann, 2000.

[37] Huidong Jin, Man Leung Wong, and Kwong-Sak Leung. Scalable model-based clustering by working on data summaries. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA*. IEEE Computer Society, 2003.

[38] Theodore Johnson, S. Muthukrishnan, and Irina Rozenbaum. Sampling algorithms in a stream operator. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 1–12, New York, NY, USA, 2005. ACM Press.

[39] Michael Jordan. A statistical approach to decision tree modeling. In *Proceedings of the Seventh ACM Conference on Computational Learning Theory*, pages 13–20, 1994.

[40] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine Piatko, Ruth Silverman, and Angela Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis ans Machine Intelligence*, 24(7), 2002.

[41] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons, 1990.

[42] Mika Klemettinen, Heikki Mannila, Pirjo Ronkainen, Hannu Toivonen, and A. Inkeri Verkamo. Finding interesting rules from large sets of discovered association rules. In Nabil R. Adam, Bharat K. Bhargava, and Yelena Yesha, editors, *Third International Conference on Information and Knowledge Management (CIKM'94)*, pages 401–407. ACM Press, 1994.

[43] Bill Kuechler and Vijay Vaishnavi. Design research in information systems. www.isworld.org/researchdesign/drisisworld.htm, accessed at 2005-08-16.

[44] Laks V. S. Lakshmanan, Jian Pei, and Yan Zhao. QC-trees: an efficient summary structure for semantic OLAP. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 64–75, New York, NY, USA, 2003. ACM Press.

[45] Bing Liu, Wynne Hsu, and Yiming Ma. Mining association rules with multiple minimum supports. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 337–341, New York, NY, USA, 1999. ACM Press.

[46] J. MacQueen. Some methods for classification and multivariate observations. In *Proceedings of the 5th Berkeley Symp. Math. Statist, Prob.*, pages 1:281–297, 1967.

[47] Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decis. Support Syst.*, 15(4):251–266, 1995.

[48] Julia Messerklinger. Project discussions concerning master thesis. private communication, March 2006.

[49] Andrew W. Moore and Mary S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1998.

[50] Biswadeep Nag, Prasad M. Deshpande, and David J. DeWitt. Using a knowledge cache for interactive discovery of association rules. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 244–253, New York, NY, USA, 1999. ACM Press.

[51] Olfa Nasraoui, Cesar Cardona Uribe, Carlos Rojas Coronel, and Fabio Gonzalez. Tecno-streams: Tracking evolving clusters in noisy data streams with a scalable immune system learning model. volume 00, page 235, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[52] Samer Nassar, Jörg Sander, and Corrine Cheng. Incremental and effective data summarization for dynamic hierarchical clustering. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 467–478, New York, NY, USA, 2004. ACM Press.

[53] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In Jorgeesh Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile proceedings*, pages 144–155, Los Altos, CA 94022, USA, 1994. Morgan Kaufmann Publishers.

[54] Raymond T. Ng, Laks V. S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 13–24, New York, NY, USA, 1998. ACM Press.

[55] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. High-performance clustering of streams and large data sets. In *Proc. of the 2002 Intl. Conf. on Data Engineering (ICDE 2002), February 2002*, 2002.

[56] Carlos Ordonez. Clustering binary data streams with k-means. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 12–19. ACM Press, 2003.

[57] Carlos Ordonez. Horizontal aggregations for building tabular data sets. In *DMKD '04: Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 35–42, New York, NY, USA, 2004. ACM Press.

[58] Carlos Ordonez and Edward Omiecinski. Frem: fast and robust em clustering for large data sets. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 590–599. ACM Press, 2002.

[59] Raghu Ramakrishnan Ramakrishnan Srikant Paul Bradley, Johannes Gehrke. Scaling mining algorithms to large databases. *Communications of the ACM*, 45(8):38–43, August 2002.

[60] Chang-Shing Perng, Haixun Wang, Sheng Ma, and Joseph L. Hellerstein. Discovery in multi-attribute data with user-defined constraints. *ACM SIGKDD Explorations Newsletter*, 4(1):56–64, 2002.

[61] Dorian Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann, 1999.

[62] Yaron Rachlin, John Dolan, and Pradeep Khosla. Learning to detect partially labeled people. In *Proceedings of the 2003 International Conference on Intelligent Robots and Systems (IROS'03)*, 2003.

[63] Stefan Schaubschläger. Effizientes clustering von horizontal verteilten daten. Master's thesis, Institute for Business Informatics/DKE at Johannes-Kepler-University Linz, Austria, 2005.

[64] D.W. Scott and S.R. Sain. *Multi-Dimensional Density Estimation*. Elsevier, Amsterdam.

[65] Noam Shental, Aharon Bar-Hillel, Tomer Hertz, and Daphna Weinshall. *Advances in Neural Information Processing Systems 16*, chapter Computing gaussian mixture models with EM using equivalence constraints. MIT Press, 2003.

[66] Noam Shental, Tomer Hertz, Aharon Bar-Hillel, and Daphna Weinshall. Computing gaussian mixture models with EM using using side-information. In *Proc. of the workshop "The Continuum from labeled to unlabeled data in machine learning and data mining" at ICML 2003.*, 2003.

[67] Rui Shi, Wanjun Jin, and Tat-Seng Chua. A novel approach to auto image annotation based on pairwise constrained clustering and semi-naïve Bayesian model. In *MMM*, pages 322–327, 2005.

[68] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Trans. On Knowledge And Data Engineering*, 8:970–974, 1996.

[69] Ting Su and Jennifer Dy. A deterministic method for initializing k-means clustering. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'04)*, pages 784–786, 2004.

[70] Sam Y. Sung, Zhao Li, Chew L. Tan, and Peter A. Ng. Forecasting association rules using existing data sets. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1448–1459, 2003.

[71] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right interestingness measure for association patterns. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 32–41, New York, NY, USA, 2002. ACM Press.

[72] Michalis Vazirgiannis, Maria Halkidi, and Dimitrios Gunopulos. *Uncertainty Handling and Quality Assessment in Data Mining*. Springer, 2003.

[73] C. J. Matheus W. J. Frawley, G. Piatetsky-Shapiro. Knowledge discovery in databases: An overview. In W. J. Frawley G. Piatetsky-Shapiro, editor, *Knowledge Discovery in Databases*, pages 1–27. AAAI Press / The MIT Press, 1991.

[74] Geoffrey I. Webb. Efficient search for association rules. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–107, New York, NY, USA, 2000. ACM Press.

[75] Max Welling and Kenichi Kurihara. Bayesian k-means as a maximisation-expectation algorithm. to appear in Proc. of the SIAM Conference in Data Mining, April 2006.

[76] David A. White and Ramesh Jain. Similarity indexing with the ss-tree. In *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*, pages 516–523, Washington, DC, USA, 1996. IEEE Computer Society.

[77] Xintao Wu, Daniel Barbara, and Yong Ye. Screening and interpreting multi-item associations based on log-linear modeling. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 276–285, New York, NY, USA, 2003. ACM Press.

[78] Evangelos Xevelonakis. Developing retention strategies based on customer profitability in telecommunications: An empirical study. *Database Marketing & Customer Strategy Management*, 12:226–242, January 2005.

[79] Bin Zhang, Gary Kleyner, and Meichun Hsu. A local search approach to K-clustering. HP Labs Technical Report HPL-1999-119, Hewlett-Packard Laboratories, 1999.

[80] Tian Zhang. *Data Clustering and Density Estimation for Very Large Datasets Plus Applications*. PhD thesis, UW-Madison Computer Sciences Department, December 1997.

[81] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 103–114, 1996.

[82] Zijian Zheng, Ron Kohavi, and Llew Mason. Real world performance of association rule algorithms. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 401–406, New York, NY, USA, 2001. ACM Press.