Nederlandse Organisatie voor Wetenschappelijk Onderzoek

The research reported in this thesis was supported by the Netherlands Organisation for Scientific Research (NWO grant no. 635.100.015).

SIKS Dissertation Series No. 2009-45
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

ISBN 978-90-393-5236-6
URL: http://igitur-archive.library.uu.nl

Making Pattern Mining Useful

Het bruikbaar maken van patroon mining
(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op gezag van de rector magnificus, prof.dr. J.C. Stoof, ingevolge het besluit van het college voor promoties in het openbaar te verdedigen op dinsdag 15 december 2009 des middags te 12.45 uur

door

Jilles Vreeken
geboren op 21 maart 1981 te Amsterdam

Promotor: Prof.dr. A.P.J.M. Siebes

Dit proefschrift werd (mede) mogelijk gemaakt met financiële steun van de Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO).

Contents

CHAPTER 1

Introduction

The discovery of patterns plays an important role in data mining. Data mining is the field of research concerned with the extraction of useful insights from large and detailed collections of data. The process of finding patterns in data is called *pattern mining*. A pattern can be any type of regularity displayed in that data, such as, e.g. which items are typically sold together, which genes are mostly active for patients of a certain disease, what type of customer is most likely to provide profit, etc, etc. Clearly, such patterns can provide useful insight.

Generally speaking, finding a pattern is easy. Discovering interesting patterns, that's where things get complicated. This thesis is about finding interesting patterns, and, more boldly, about making pattern mining useful. It is about how to discover few, but highly interesting patterns. And, prominently, it is about how to put these patterns to good use, solving a number of data mining problems.

But, before we discuss the actual content of the thesis, let us first informally discuss pattern mining and identify why it is not yet as useful as it could be.

Pattern mining

In order to find patterns, we need two ingredients, besides the data. First, we need to construct a notion of the kind of information we would like to extract, i.e. what should the pattern look like. Such a notion is called a pattern language. The second required ingredient is a computer program that will venture into the data and return the patterns. We call this the pattern miner. In general, constructing a naïve miner is easy. Building a *fast* miner can be a totally different story. As always, the devil is in the details.

Let us consider an example how pattern mining could be used, e.g. in medicine, for gaining insight in the causes of a particular disease. Normally, following the scientific method, a doctor would build a hypothesis, that is, an idea of what the cause could be. In other words, a pattern. For this hypothesis not to be a shot in the dark, the doctor needs to be able to oversee the symptoms, behaviours, etc, that the patients exhibit. That is, he or she must be able to 'see' the pattern. The hypothesis can then be tested, and so shown to be correct or not.

This works very well, up till the point where problems become too complicated, when it becomes impossible to gain sufficient overview. Good examples of such cases are so-called complex hereditary diseases. These are diseases in which the cause lies in the interplay between multiple genes, and any number of lifestyle and environmental factors. A prime example of such a disease is celiac disease [22], better known by the fact that it often leads to gluten intolerance. This gluten intolerance is the main environmental factor. Further, two genes are known to influence the disease, but together explain only 20% of the variation: there are more factors. An additional complication is that in some cases there are many variables (e.g. the number of genes), but relatively little data: regularities that seem sound may be spurious, and vice versa.

Simply put, in cases like these there exists such a gigantic number of possible combinations of causes, that it is impossible for a human to gain enough overview to determine the factors that matter.

We can, however, apply pattern mining. We simply mine the patterns in the gathered data, and return those that pass certain criteria. In this case, such a pattern could be a combination of factors that have a strong relation with the disease. The doctor then selects the most promising patterns, e.g. those in accordance with present knowledge or those that contradict it, and builds a hypothesis from it.

So far, so good. However, in practice, the poor doctor will now be swamped in patterns. From being unable to overview the data, the problem now has become that it is impossible to overview the potentially interesting patterns.

Since the conception of pattern mining, one of the key goals has been completeness in discovery: the task is to find *all* patterns that satisfy certain conditions. In a way, this goal is very useful: for every returned pattern we know that it fulfils all conditions that we have set, e.g. it occurs frequently enough not to be considered 'noise'.

The drawback is that the number of patterns returned is typically prohibitively large. Generally, there are lots of patterns satisfying the conditions, and many patterns convey roughly the same information about the data: they are variations of the same theme.

Many proposals to reduce this redundancy exist. However, although stark reductions are attained by the proposed techniques, up to orders of magnitude,

the resulting numbers are still generally by far too large to be manhandable and considered by our expert, the doctor.

Making pattern mining useful

So, while pattern mining holds great promise, I dare say it collapses under its own weight: it finds patterns too easily. This is in particular so because the use of an interestingness measure, i.e. criteria to cherry pick individual patterns that should be interesting, proves to be very hard in practice: if we set such constraints tight, only few but commonly known patterns are returned, and when these criteria are set looser we are overwhelmed by the number of results.

While patterns can clearly provide useful insight, finding just those interesting patterns is a question not yet answered by pattern mining. The sheer amount of results makes it virtually impossible for the patterns to be interpreted by human experts such as our doctor. Further, it prohibits pattern mining, and the detail provided by the discovered patterns, to be practically applied more generally in data mining.

What this pattern explosion comes down to, is that we are asking the wrong question. While we ask for *all* patterns that satisfy the conditions, at the same time we actually only want to have a small set of the best patterns.

This thesis therefore proposes a different approach. We do not want to find all patterns in a database, or trying to summarise those collections of patterns. Instead, we want small, non-redundant, sets of high-quality patterns that summarise the data well, i.e. patterns that describe the data. The resulting groups should be small enough to be analysed by an expert such as our doctor and provide a detailed overview of the data.

In this thesis the problem of mining sets of patterns is approached through the Minimum Description Length principle, that is, by lossless compression. Intuitively, we can say that the better a set of patterns compress the data, the better it captures the regularities in the data. With MDL we define the best set of patterns as that set of patterns that compresses the data best.

One could ask, why would our doctor be interested in patterns that compress? Quite simply, because these are the patterns that matter. Because MDL takes the complexity of the selected patterns into account, we know that redundant patterns will be been eliminated, as well as those that model spurious information: such patterns only contribute to the complexity of the model, and therefore they will not be part of the set of patterns that compress best. In other words, the doctor will find that the best compressing set of patterns describes the data very well: without redundancy and noise.

Further, the resulting sets of patterns are small in size, thereby solving the problem of the pattern explosion. Second, by selection on describing data well, these sets contain detailed information on the most important patterns in the data. These two aspects make these sets of patterns *useful*, i.e. they cannot

only be presented for evaluation by an expert such as our doctor, but also naturally be applied to solve various data mining problems.

This thesis includes five such applications, including measuring and characterising differences between databases, finding blocks of data with similar characteristics, and estimating the missing values for data with incomplete records; all problems often faced by our doctor. These are all naturally approached through the MDL-principle. However, it is the level of detail captured in the pattern sets that makes the difference, allowing for both high performance *and* characterisation of the why.

As such, the research objective of this thesis is phrased by the title of this thesis and this subsection:

Making pattern mining useful

This goal includes to develop techniques for finding small groups of high-quality patterns, such that they can be presented to an expert, and showing how these pattern sets provide insight in the data and can be used to solve open data mining problems.

Outline of this thesis

This thesis is divided into nine chapters. Chapters 2 to 8 are edited versions of earlier published work. The references to the original publications are found on the first pages of those chapters. The topics treated in the following chapters are summarised as follows:

- In Chapter 2 we propose to use the Minimum Description Length principle to select small groups of frequent itemsets that describe the data well. To this end, we introduce Krimp; a heuristic algorithm for finding the optimal set of frequent itemsets. Through extensive evaluation, amongst which through the Krimp-classifier, we show the high quality of these *code tables*.

- In Chapter 3 we show how one can measure and characterise the difference between transaction databases. Difference is measured by calculating the relative Krimp-compressibility of the datasets. The code tables allow detailed insight in why data is deemed similar or not.

- In Chapter 4 we give two MDL-based algorithms through which one can identify and characterise the components of a database. Data is split into homogeneous blocks, such that the compression is optimised. The methods are orthogonal in approach: one is data-driven, while the second extracts the components from a Krimp model.

- In Chapter 5 we show how code tables, while mined as descriptive models, can be used as generative models. We introduce an algorithm that generates data that is virtually indistinguishable from the original. We show the use for this in privacy-preserving data mining, as our method provides anonymised data with all important patterns intact.
- In Chapter 6 we further onto the generative path, and introduce three algorithms for completion of data with missing values. All three follow the MDL principle, i.e. the completed database that can be compressed best, is the best completion. The methods attain high imputation accuracy and maintain all count statistics of the data.
- In Chapter 7 we present LESS, an algorithm to select patterns for describing data 0/1 symmetrically; not just items that are present. As patterns, it uses low-entropy sets [59], itemsets that identify strong interactions between attributes. It follows the MDL principle and selects groups of these patterns such that the data is described succinct.
- In Chapter 8 we introduce Pack, an algorithm for selecting good itemsets through refined MDL. It uses decision trees to compress data 0/1 symmetrically and attains high compression ratios. Besides selecting itemsets from a larger collection, we can also mine models directly from data.

Chapter 9 draws conclusions and summarises the main contributions of this thesis.

CHAPTER 2

Krimp: Mining Itemsets that Compress

One of the major problems in pattern mining is the explosion of the number of results. Tight constraints reveal only common knowledge, while loosening these leads to an explosion in the number of returned patterns. This is caused by large groups of patterns essentially describing the same set of transactions. In this chapter we approach this problem using the MDL principle: the best set of patterns is that set that compresses the database best. For this task we introduce the Krimp algorithm. Experimental evaluation shows that typically only hundreds of itemsets are returned; a dramatic reduction, up to 7 orders of magnitude, in the number of frequent item sets. These selections, called code tables, are of high quality. This is shown with compression ratios, swap-randomisation, and the accuracies of the code table-based Krimp classifier, all obtained on a wide range of datasets. Further, we extensively evaluate the heuristic choices made in the design of the algorithm.[1]

[1] This work has been accepted for publication as [123]:
J. Vreeken, M. van Leeuwen, and A. Siebes. Krimp: Mining itemsets that compress. *Data Mining and Knowledge Discovery*, Springer.
It is based on work originally published as [113] and [78]:
A. Siebes, J. Vreeken, and M. van Leeuwen (2006). Item sets that compress. In *Proceedings of the SDM'06*, pages 393-404.
M. van Leeuwen, J. Vreeken, and A. Siebes (2006). Compression picks the item sets that matter. In *Proceedings of the ECML PKDD'06*, pages 585-592.

2.1 Introduction

Patterns

Without a doubt, *pattern mining* is one of the most important concepts in data mining. In contrast to *models*, patterns describe only part of the data, see, e.g., [57,97]. In this chapter, we consider one class of pattern mining problems, viz., *theory mining* [90]. In this case, the patterns describe interesting subsets of the database.

Formally, this task has been described by [89] as follows. Given a database $\mathcal{D}$, a language $\mathcal{L}$ defining subsets of the data and a selection predicate q that determines whether an element $\phi \in \mathcal{L}$ describes an interesting subset of $\mathcal{D}$ or not, the task is to find

$$\mathcal{T}(\mathcal{L}, \mathcal{D}, q) = \{ \phi \in \mathcal{L} \mid q(\mathcal{D}, \phi) \text{ is true } \}.$$

That is, the task is to find *all* interesting subgroups.

The best known instance of theory mining is *frequent set mining* [5]; this is the problem we will consider throughout this chapter. The standard example for this is the analysis of shopping baskets in a supermarket. Let $\mathcal{I}$ be the set of items the store sells. The database $\mathcal{D}$ consists of a set of *transactions* in which each transaction t is a subset of $\mathcal{I}$. The pattern language $\mathcal{L}$ consists of *itemsets*, i.e. again sets of items. The *support* of an itemset X in $\mathcal{D}$ is defined as the number of transactions that contain X, i.e.

$$supp_{\mathcal{D}}(X) = |\{t \in \mathcal{D} \mid X \subseteq t\}|.$$

The 'interestingness' predicate is a threshold on the support of the itemsets, the *minimal support: minsup*. In other words, the task in frequent set mining is to compute

$$\{X \in \mathcal{L} \mid supp_{\mathcal{D}}(X) \geq \text{minsup}\}.$$

The itemsets in the result are called *frequent* itemsets. Since the support of an itemset decreases w.r.t. set inclusion, the *A Priori* property,

$$X \subseteq Y \Rightarrow supp_{\mathcal{D}}(Y) \leq supp_{\mathcal{D}}(X),$$

a simple level-wise search algorithm suffices to compute all the frequent itemsets. Many efficient algorithms for this task are known, see, e.g. [50]. Note, however, that since the size of the output can be exponential in the number of items, the term efficient is used w.r.t. the size of the output. Moreover, note that whenever $\mathcal{L}$ and q satisfy an A Priori like property, similarly efficient algorithms exist [89].

Sets of patterns

A major problem in frequent itemset mining, and pattern mining in general, is the so-called *pattern explosion*. For tight interestingness constraints, e.g. a high minsup threshold, only few, but well-known, patterns are returned. However, when the constraints are loosened, pattern discovery methods quickly return humongous amounts of patterns; the number of frequent itemsets is often many orders of magnitude larger than the number of transactions in the dataset.

This pattern explosion is caused by the locality of the minimal support constraint; each individual itemset that satisfies the constraint is added to the result set, independent of the already returned sets. Hence, we end up with a rather redundant set of patterns, in which many patterns essentially describe the same part of the database. One could impose additional constraints on the individual itemsets to reduce their number, such as closed frequent itemsets [103]. While this somewhat alleviates the problem, redundancy remains an issue.

We take a different approach: rather than focusing on the individual frequent itemsets, we focus on the resulting set of itemsets. That is, we want to find the *best set* of (frequent) itemsets. The question is, of course, what is the best set? Clearly, there is no single answer to this question. For example, one could search for small sets of patterns that yield good classifiers, or show maximal variety [69, 70].

We view finding the best set of itemsets as an *induction problem*. That is, we want to find the set of patterns that *describe the database best*. There are many different induction principles. So, again the question is which one to take?

The classical statistical approach [114] would be to basically test hypotheses, meaning that we would have to test every possible set of itemsets. Given the typically huge number of frequent itemsets and the exponentially larger number of sets of itemsets, testing all these pattern sets individually does not seem a computationally attractive approach.

Alternatively, the Bayesian approach boils down to updating the *a priori distribution* with the data [13]. This update is computed using Bayes' rule, which requires the computation of $P(\mathcal{D} \mid M)$. How to define this probability for sets of frequent itemsets is not immediately obvious. Moreover, our primary goal is to find a *descriptive* model, not a *generative* one[2]. For the same reason, principles that are geared towards predictive models, such as *Statistical Learning Theory* [121], are not suitable. Again, we are primarily interested in a descriptive model, not a predictive one.

The *Minimal Description Length Principle* (MDL) [52, 53, 108] on the other hand, is geared towards descriptions of the data. One could summarise this

[2] Although, in Chapter 5, we do build generative models from the small number of selected itemsets that generates data virtually indiscernible from the original.

approach by the slogan: *the best model compresses the data best*. By taking this approach we do not try to compress the set of frequent itemsets, rather, we want to find that set of frequent itemsets that yields the best *lossless* compression of the database.

The MDL principle provides us a fair way to balance the complexities of the compressed database and the encoding. Note that both need to be considered in the case of lossless compression. Intuitively, we can say that if the encoding is too simple, i.e. it consists of too few itemsets, the database will hardly be compressed. On the other hand, if we use too many, the code table for coding/decoding the database will become too complex.

Considering the combination of the complexities of the compressed data and the encoding is the cornerstone of the MDL principle; it ensures that the model will not be overly elaborate or simplistic w.r.t. the complexity of the data.

While MDL removes the need for user defined parameters, it comes with its own problems: only heuristics, no guaranteed algorithms. However, our experiments show that these heuristics give a dramatic reduction in the number of itemsets. Moreover, the set of patterns discovered is characteristic of the database as independent experiments verify; see Section 2.7.

We are not the first to address the pattern explosion, nor are we the first to use MDL. We are the first, however, to employ the MDL principle to select the best pattern set. For a discussion of related work, see Section 2.6.

A primary version of the Krimp algorithm (although not yet under that name) was published as [113] and a primary version of the Krimp classifier as [78]. Here, we thoroughly discuss the theory and choices, as well as providing extensive experimental validation of the methods on 27 datasets. In particular, we further evaluate the heuristic choices made in the Krimp algorithm, show that the selected itemsets model relevant structure in the data and that the method is robust w.r.t noise.

The chapter is organised as follows. First, we cover the theory of using MDL for selecting itemsets, after which we define our problem formally and analyse its complexity. We introduce the heuristic Krimp algorithm for solving the problem in Section 2.3. In a brief interlude we provide a small sample of the results. We continue with theory on using MDL for classification, and introduce the Krimp classifier in Section 2.5. Related work is discussed in Section 2.6. Section 2.7 provides extensive experimental validation of our method, as well as an evaluation of the heuristic choices made in the design of the Krimp algorithm. We round up with discussion in Section 2.8 and conclude in Section 2.9.

2.2 Theory

In this section we state our problem formally. First we briefly discuss the MDL principle. Next we introduce our models, code tables. We show how we can encode a database using such a code table, and what the total size of the coded database is. With these ingredients, we formally state the problems studied in this chapter. Throughout the chapter all logarithms have base 2.

MDL

MDL (Minimum Description Length) [52,108], like its close cousin MML (Minimum Message Length) [126], is a practical version of Kolmogorov Complexity [81]. All three embrace the slogan *Induction by Compression*. For MDL, this principle can be roughly described as follows.

Given a set of models[3] $\mathcal{H}$, the best model $H \in \mathcal{H}$ is the one that minimises

$$L(H) + L(\mathcal{D} \mid H)$$

in which

- $L(H)$ is the length, in bits, of the description of H, and
- $L(\mathcal{D} \mid H)$ is the length, in bits, of the description of the data when encoded with H.

This is called two-part MDL, or *crude* MDL. As opposed to *refined* MDL, where model and data are encoded together [53]. We use this particular version of MDL because we are specifically interested in the compressor: the set of frequent itemsets that yields the best compression. Further, although refined MDL has stronger theoretical foundations, it cannot be computed except in some special cases.

To use MDL, we have to define what our models $\mathcal{H}$ are, how a $H \in \mathcal{H}$ describes a database, and how all of this is encoded in bits.

MDL for itemsets

The key idea of our compression based approach is the *code table*. A code table is a simple two-column translation table that has itemsets on the left-hand side and a code for each itemset on its right-hand side. With such a code table we find, through MDL, the set of itemsets that together optimally describe the data.

[3] MDL-theorists tend to talk about *hypothesis* in this context, hence the $\mathcal{H}$; see [52] for the details.

Definition 1. *Let $\mathcal{I}$ be a set of items and $\mathcal{C}$ a set of code words. A code table CT over $\mathcal{I}$ and $\mathcal{C}$ is a two-column table such that:*

1. *The first column contains itemsets, that is, subsets over $\mathcal{I}$. This column contains at least all singleton itemsets.*
2. *The second column contains elements from $\mathcal{C}$, such that each element of $\mathcal{C}$ occurs at most once.*

An itemset X, drawn from the power set of $\mathcal{I}$, i.e. $X \in \mathcal{P}(\mathcal{I})$, occurs in CT, denoted by $X \in CT$ iff X occurs in the first column of CT, similarly for a code $C \in \mathcal{C}$. For $X \in CT$, $code_{CT}(X)$ denotes its code, i.e. the corresponding element in the second column. We call the set of itemsets $\{X \in CT\}$ the coding set CS. For the number of itemsets in the code table we write $|CT|$, i.e. we define $|CT| = |\{X \in CT\}|$. Likewise, $|CT \setminus \mathcal{I}|$ indicates the number of non-singleton itemsets in the code table.

To encode a transaction t from database $\mathcal{D}$ over $\mathcal{I}$ with code table CT, we require a cover function $cover(CT, t)$ that identifies which elements of CT are used to encode t. The parameters are a code table CT and a transaction t, the result is a disjoint set of elements of CT that cover t. Or, more formally, a cover function is defined as follows.

Definition 2. *Let $\mathcal{D}$ be a database over a set of items $\mathcal{I}$, t a transaction drawn from $\mathcal{D}$, let $\mathcal{CT}$ be the set of all possible code tables over $\mathcal{I}$, and CT a code table with $CT \in \mathcal{CT}$. Then, $cover : \mathcal{CT} \times \mathcal{P}(\mathcal{I}) \to \mathcal{P}(\mathcal{P}(\mathcal{I}))$ is a* cover function *iff it returns a set of itemsets such that*

1. *$cover(CT, t)$ is a subset of CS, the coding set of CT, i.e. $X \in cover(CT, t) \to X \in CT$*
2. *if $X, Y \in cover(CT, t)$, then either $X = Y$ or $X \cap Y = \emptyset$*
3. *the union of all $X \in cover(CT, t)$ equals t, i.e. $t = \bigcup_{X \in cover(CT,t)} X$*

We say that $cover(CT, t)$ covers t. Note that there exists at least one well-defined cover function on any code table CT over $\mathcal{I}$ and any transaction $t \in \mathcal{P}(\mathcal{I})$, since CT contains at least the singleton itemsets from $\mathcal{I}$.

To encode a database $\mathcal{D}$ using code table CT we simply replace each transaction $t \in \mathcal{D}$ by the codes of the itemsets in the cover of t,

$$t \to \{\ code_{CT}(X) \mid X \in cover(CT, t)\ \}.$$

Note that to ensure that we can decode an encoded database uniquely we assume that $\mathcal{C}$ is a *prefix code*, in which no code is the prefix of another code [35]. (Confusingly, such codes are also known as *prefix-free* codes [81].)

Since MDL is concerned with the best compression, the codes in CT should be chosen such that the most often used code has the shortest length. That is, we should use an optimal prefix code. Note that in MDL we are never interested in materialised codes, but only in the complexities of the model and the data. Therefore, we are only interested in the *lengths* of the codes of itemsets $X \in CT$. As there exists a nice correspondence between code lengths and probability distributions (see, e.g. [81]), we can calculate the optimal code lengths through the Shannon entropy. So, to determine the complexities we do not have to operate an actual prefix coding scheme such as Shannon-Fano or Huffman encoding.

Theorem 1. *Let P be a distribution on some finite set $\mathcal{D}$, there exists an optimal prefix code $\mathcal{C}$ on $\mathcal{D}$ such that the length of the code for $d \in \mathcal{D}$, denoted by $L(d)$ is given by*

$$L(d) = -\log(P(d)).$$

Moreover, this code is optimal in the sense that it gives the smallest expected code size for data sets drawn according to P. (For the proof, please refer to Theorem 5.4.1 in [35])

The optimality property means that we introduce no bias using this code length. The probability distribution induced by a cover function is, of course, simply given by the relative usage frequency of each of the item sets in the code table. To determine this, we need to know how often a certain code is used. We define the *usage* count of an itemset $X \in CT$ as the number of transactions t from $\mathcal{D}$ where X is used to cover. Normalised, this frequency represents the probability that this code is used in the encoding of an arbitrary $t \in \mathcal{D}$. The optimal code length then is $-log$ of this probability [81], and a code table is optimal if all its codes have their optimal length. Note that we use fractional lengths, not integer-valued lengths of materialised codes. This ensures that the length of a code accurately represents its usage probability, and since we are not interested in materialised codes, only relative lengths are of importance. After all, our ultimate goal is to score the optimal code table and not to actually compress the data. More formally, we have the following definition.

Definition 3. *Let $\mathcal{D}$ be a transaction database over a set of items $\mathcal{I}$, $\mathcal{C}$ a prefix code, cover a cover function, and CT a code table over $\mathcal{I}$ and $\mathcal{C}$. The* usage *count of an itemset $X \in CT$ is defined as*

$$usage_{\mathcal{D}}(X) = |\{\ t \in \mathcal{D} \mid X \in cover(CT, t)\ \}|.$$

The probability *of $X \in CT$ being used in the cover of an arbitrary transaction $t \in \mathcal{D}$ is thus given by*

$$P(X \mid \mathcal{D}) = \frac{usage_{\mathcal{D}}(X)}{\sum_{Y \in CT} usage_{\mathcal{D}}(Y)}.$$

The $code_{CT}(X)$ *for* $X \in CT$ *is* optimal *for* $\mathcal{D}$ *iff*

$$L(code_{CT}(X)) = |code_{CT}(X)| = -\log(P(X|\mathcal{D})).$$

A code table CT *is* code-optimal *for* $\mathcal{D}$ *iff all its codes,*

$$\{\ code_{CT}(X) \mid X \in CT\ \},$$

are optimal for $\mathcal{D}$*.*

From now onward we assume that code tables are code-optimal for the database they are induced on, unless we state differently.

Now, for any database $\mathcal{D}$ and a code table CT over the same set of items $\mathcal{I}$ we can compute $L(\mathcal{D} \mid CT)$. It is simply the summation of the encoded lengths of the transactions. The encoded size of a transaction is simply the sum of the sizes of the codes of the itemsets in its cover. In other words, we have the following trivial lemma.

Lemma 2. *Let* $\mathcal{D}$ *be a transaction database over* $\mathcal{I}$*,* CT *be a code table over* $\mathcal{I}$ *and code-optimal for* $\mathcal{D}$*,* cover *a cover function, and* usage *the usage function for* cover*.*

1. *For any* $t \in \mathcal{D}$ *its encoded length, in bits, denoted by* $L(t \mid CT)$*, is*

$$L(t \mid CT) \quad = \sum_{X \in cover(CT,t)} L(code_{CT}(X)).$$

2. *The encoded size of* $\mathcal{D}$*, in bits, when encoded by* CT*, denoted by* $L(\mathcal{D} \mid CT)$*, is*

$$L(\mathcal{D} \mid CT) \quad = \quad \sum_{t \in \mathcal{D}} L(t \mid CT).$$

With Lemma 2, we can compute $L(\mathcal{D} \mid H)$. To use the MDL principle, we still need to know what $L(H)$ is, i.e. the encoded size of a code table.

Recall that a code table is a two-column table consisting of itemsets and codes. As we know the size of each of the codes, the encoded size of the second column is easily determined: it is simply the sum of the lengths of the codes. For encoding the itemsets, the first column, we have to make a choice.

A naïve option would be to encode each item with a binary integer encoding, that is, using $\log(\mathcal{I})$ bits per item. Clearly, this is hardly optimal; there is no difference in encoded length between highly frequent and infrequent items.

A better choice is to encode the itemsets using the codes of the simplest code table, i.e. the code table that contains only the singleton itemsets $I \in \mathcal{I}$. This

code table, with optimal code lengths for database $\mathcal{D}$, is called the *standard code table* for $\mathcal{D}$, denoted by ST. It is the optimal encoding of $\mathcal{D}$ when nothing more is known than just the frequencies of the individual items; it assumes the items to be fully independent. As such, it provides a practical bound: ST provides the simplest, independent, description of the data. This encoding allows us to reconstruct the database up to the names of the individual items. With these choices, we have the following definition.

Definition 4. *Let $\mathcal{D}$ be a transaction database over $\mathcal{I}$ and CT a code table that is code-optimal for $\mathcal{D}$. The size of CT in bits, denoted by $L(CT \mid \mathcal{D})$, is given by*

$$L(CT \mid \mathcal{D}) = \sum_{X \in CT : usage_{\mathcal{D}}(X) \neq 0} |code_{ST}(X)| + |code_{CT}(X)|.$$

Note that we do not take itemsets with zero usage into account. Such itemsets are not used to code. We use $L(CT)$ wherever $\mathcal{D}$ is clear from context.

With these results we know the total size of our encoded database. It is simply the size of the encoded database plus the size of the code table. That is, we have the following result.

Definition 5. *Let $\mathcal{D}$ be a transaction database over $\mathcal{I}$, let CT be a code table that is code-optimal for $\mathcal{D}$ and* cover *a cover function. The* total compressed size *of the encoded database and the code table, in bits, denoted by $L(\mathcal{D}, CT)$ is given by*

$$L(\mathcal{D}, CT) = L(\mathcal{D} \mid CT) + L(CT \mid \mathcal{D}).$$

Now that we know how to compute $L(\mathcal{D}, CT)$, we can formalise our problem using MDL. Before that, we discuss three design choices we did not mention so far, because they do not influence the total compressed size of a database.

First, when encoding a database $\mathcal{D}$ with a code table CT, we do not mark the end of a transaction, i.e. we do not use stop-characters. Instead, we assume a given framework that needs to be filled out with the correct items upon decoding. Since such a framework adds the same additive constant to $L(\mathcal{D} \mid CT)$ for any CT over $\mathcal{I}$, it can be disregarded.

Second, for more detailed descriptions of the items in the decoded database, one could add an ASCII table giving the names of the individual items to a code table. Since such a table is the same for all code tables over $\mathcal{I}$, this is again an additive constant we can disregard for our purposes.

Last, since we are only interested in the complexity of the content of the code table, i.e. the itemsets, we disregard the complexity of its structure. That is, like for the database, we assume a static framework that fits any possible code table, consisting of up to $|\mathcal{P}(\mathcal{I})|$ itemsets, and is filled out using the above

encoding. The complexity of this framework is equal for any code table CT and dataset $\mathcal{D}$ over $\mathcal{I}$, and therefore we can also disregard this third additive constant when calculating $L(\mathcal{D}, CT)$.

The problem

Our goal is to find the set of itemsets that best describe the database $\mathcal{D}$. Recall that the set of itemsets of a code table, i.e. $\{X \in CT\}$, is called the coding set CS. Given a coding set, a cover function and a database, a (code-optimal) code table CT follows automatically.

Given a set of itemsets $\mathcal{F}$, the problem is to find a subset of $\mathcal{F}$ which leads to a minimal encoding; where minimal pertains to all possible subsets of $\mathcal{F}$. To make sure this is possible, $\mathcal{F}$ should contain at least the singleton item sets $X \in \mathcal{I}$. We will call such a set, a candidate set. By requiring the smallest coding set, we make sure the coding set contains no unused non-singleton elements, i.e. $usage_{CT}(X) > 0$ for any non-singleton itemset $X \in CT$.

Minimal Coding Set Problem *Let $\mathcal{I}$ be a set of items and let $\mathcal{D}$ be a dataset over $\mathcal{I}$,* cover *a cover function, and $\mathcal{F}$ a set of candidate itemsets. Find the smallest coding set $CS \subseteq \mathcal{F}$ such that for the corresponding code table CT the total compressed size, $L(\mathcal{D}, CT)$, is minimal.*

A solution for the Minimal Coding Set Problem allows us to find the 'best' coding set from a given collection of itemsets, e.g. (closed) frequent itemsets for a given minimal support. If $\mathcal{F} = \{X \in \mathcal{P}(\mathcal{I}) \mid supp_{\mathcal{D}}(X) > 0\}$, i.e. when $\mathcal{F}$ consists of all itemsets that occur in the data, there exists no candidate set $\mathcal{F}'$ that results in a smaller total encoded size. Hence, in this case the solution is truly the minimal coding set for $\mathcal{D}$ and *cover*.

In order to solve the Minimal Coding Set Problem, we have to find the optimal code table and cover function. To this end, we have to consider a humongous search space, as we will detail in the next subsection.

How hard is the problem?

The number of coding sets does not depend on the actual database, and nor does the number of possible cover functions. Because of this, we can compute the size of our search space rather easily.

A coding set contains the singleton itemsets plus an almost arbitrary subset of $\mathcal{P}(\mathcal{I})$. Almost, since we are not allowed to choose the $|\mathcal{I}|$ singleton itemsets.

In other words, there are

$$\sum_{k=0}^{2^{|\mathcal{I}|}-|\mathcal{I}|-1} \binom{2^{|\mathcal{I}|}-|\mathcal{I}|-1}{k}$$

Table 2.1: The number of cover possibilities for a database of one (1) transaction over $\mathcal{I}$.

$\lvert\mathcal{I}\rvert$	$NCP(\mathcal{I})$	$\lvert\mathcal{I}\rvert$	$NCP(\mathcal{I})$
1	1	4	2.70×10^{12}
2	8	5	1.90×10^{34}
3	8742	6	4.90×10^{87}

possible coding sets. In order to determine which one of these minimises the total encoded size, we have to consider all corresponding (code-optimal) code tables using every possible cover function. Since every itemset $X \in CT$ can occur only once in the cover of a transaction and no overlap between the itemsets is allowed, this translates to traversing the code table once for every transaction. However, as each possible code table order may result in a different cover, we have to test every possible code table order per transaction to cover. Since a set of n elements admits $n!$ orders, the total size of the search space is as follows.

Lemma 3. *For one transaction over a set of items $\mathcal{I}$, the number of cover possibilities, that is number of ordered coding sets, is given by $NCP(\mathcal{I})$.*

$$NCP(\mathcal{I}) = \sum_{k=0}^{2^{|\mathcal{I}|}-|\mathcal{I}|-1} \binom{2^{|\mathcal{I}|}-|\mathcal{I}|-1}{k} \times (k+|\mathcal{I}|)!$$

So, even for a rather small set $\mathcal{I}$ and a database of only *one* transaction, the search space we are facing is already huge. Table 2.1 gives an approximation of NCP for the first few sizes of $\mathcal{I}$. Clearly, the search space is far too large to consider exhaustively.

To make matters worse, there is no useable structure that allows us to prune level wise as the attained compression is not monotone w.r.t. the addition of itemsets. So, without calculating the *usage* of the itemsets in CT, it is generally impossible to call the effects (improvement or degrading) on the compression when an itemset is added to the code table. This can be seen as follows.

Suppose a database $\mathcal{D}$, itemsets X and Y such that $X \subset Y$, and a coding set CS, all over $\mathcal{I}$. The addition of X to CS, can lead to a degradation of the compression, first and foremost as X may add more complexity to the code table than is compensated for by using X in encoding $\mathcal{D}$. Second, X may get in 'the way' of itemsets already in CS, as such providing those itemsets with lower *usage*, longer codes and thus leading to massively worse compression. Instead, let us consider adding Y. While more complex, exactly those items $Y \setminus X$ may replace the hindered itemsets. As such Y may circumvent getting

Algorithm 1 The Standard Code Table Algorithm

Require: A transaction database $\mathcal{D}$ over a set of items $\mathcal{I}$.
Ensure: The standard code table CT for $\mathcal{D}$.

StandardCodeTable (t, CT) :

1: $CT \leftarrow \emptyset$
2: **for all** $X \in \mathcal{I}$ **do**
3: insert X into CT
4: $usage_{\mathcal{D}}(X) \leftarrow supp_{\mathcal{D}}(X)$
5: $code_{CT}(x) \leftarrow$ optimal code for X
6: **end for**
7: **return** CT

'in the way', and thus lead to an improved compression. However, this can just as well be the other way around, as exactly those items can also lead to low usage and/or overlap with other/more existing itemsets in CS.

2.3 Algorithms

In this section we present algorithms for solving the problem formulated in the previous section. As shown above, the search space one needs to consider for finding the optimal code table is far too large to be considered exhaustively. We therefore have to resort to heuristics.

Basic heuristic

To cut down a large part of the search space, we use the following simple greedy search strategy:

- Start with the standard code table ST, containing only the singleton itemsets $I \in \mathcal{I}$.
- Add the itemsets from $\mathcal{F}$ one by one. If the resulting codes lead to a better compression, keep it. Otherwise, discard the set.

To turn this sketch into an algorithm, some choices have to be made. Firstly, in which order are we going to encode a transaction? So, what cover function are we going to employ? Secondly, in which order do we add the itemsets? Finally, do we *prune* the newly constructed code table before we continue with the next candidate itemset or not?

Before we discuss each of these questions, we briefly describe the initial encoding. This is, of course, the encoding with the standard code table. For this, we need to construct a code table from the elements of $\mathcal{I}$. The algorithm

Algorithm 2 The StandardCover Algorithm

Require: Transaction $t \in \mathcal{D}$ and code table CT, with CT and $\mathcal{D}$ over a set of items $\mathcal{I}$.

Ensure: A cover of t using non-overlapping elements of CT.

StandardCover (t, CT) :

1: $S \leftarrow$ smallest element X of CT in **Standard Cover Order** for which $X \subseteq t$
2: **if** $t \setminus S = \emptyset$ **then**
3: $\quad Res \leftarrow \{S\}$
4: **else**
5: $\quad Res \leftarrow \{S\} \cup$ StandardCover$(t \setminus S, CT)$
6: **end if**
7: **return** Res

called **Standard Code Table**, given in pseudo-code as Algorithm 1, returns such a code table. It takes a set of items and a database as parameters and returns a code table. Note that for this code table all cover functions reduce to the same, namely the cover function that replaces each item in a transaction with its singleton itemset. As the singleton itemsets are mutually exclusive, all elements $X \in \mathcal{I}$ will be used $supp_{\mathcal{D}}(X)$ times by this cover function.

Standard cover function

From the problem complexity analysis in the previous section it is quite clear that finding an optimal cover of the database is practically impossible, even if we are given the optimal set of itemsets as the code table: examining all $|CT|!$ possible permutations is already virtually impossible for one transaction, let alone expanding this to all possible combinations of permutations for all transactions.

We therefore employ a heuristic and introduce a standard cover function which considers the code table in a fixed order. The pseudo-code for this **Standard Cover** function is given as Algorithm 2. For a given transaction t, the code table is traversed in a fixed order. An itemset $X \in CT$ is included in the cover of t iff $X \subseteq t$. Then, X is removed from t and the process continues to cover the uncovered remainder, i.e. $t \setminus X$. Using the same order for every transaction drastically reduces the complexity of the problem, but leaves the choice of the order.

Again, considering all possible orders would be best, but is impractical at best. A more prosaic reason is that our algorithm will need a definite order; random choice does not seem the wisest of ideas. When choosing an order, we should take into account that the order in which we consider the itemsets may

make it easier or more difficult to insert candidate itemsets into an already sorted code table.

We choose to sort the elements $X \in CT$ first decreasing on length, second decreasing on support in $\mathcal{D}$ and thirdly lexicographically increasing to make it a total order. To describe the order compactly, we introduce the following notation. We use $\downarrow$ to indicate an attribute is sorted descending, and $\uparrow$ to indicate it is sorted ascending:

$$|X| \downarrow \quad supp_{\mathcal{D}}(X) \downarrow \quad \text{lexicographically} \uparrow$$

We call this the **Standard Cover Order**. The rationale is as follows. To reach a good compression we need to replace as many individual items as possible, by as few and short as possible codes. The above order gives priority to long itemsets, as these can replace as many as possible items by just one code. Further, we prefer those itemsets that occur frequently in the database to be used as often as possible, resulting in high *usage* values and thus short codes. We rely on MDL not to select overly specific itemsets, as such sets can only be infrequently used and would thus receive relatively long codes.

Standard candidate order

Next, we address the order in which candidate itemsets will be regarded. Preferably, the candidate order should be in concord with the cover strategy detailed above. We therefore choose to sort the candidate itemsets such that long, frequently occurring itemsets are given priority. Again, to make it a total order we thirdly sort lexicographically. So, we sort the elements of $\mathcal{F}$ as follows:

$$supp_{\mathcal{D}}(X) \downarrow \quad |X| \downarrow \quad \text{lexicographically} \uparrow$$

We refer to this as the **Standard Candidate Order**. The rationale for it is as follows. Itemsets with the highest support, those with potentially the shortest codes, end up at the top of the list. Of those, we prefer the longest sets first, as these will be able to replace as many items as possible. This provides the search strategy with the most general itemsets first, providing ever more specific itemsets along the way.

A welcome advantage of the standard orders for both the cover function and the candidate order is that we can easily keep the code table sorted. First, the length of an itemset is readily available. Second, with this candidate order we know that any candidate itemset for a particular length will have to be inserted after any already present code table element with the same length. Together, this means that we can insert a candidate itemset at the right position in the code table in O(1) if we store the code table elements in an array (over itemset length) of lists.

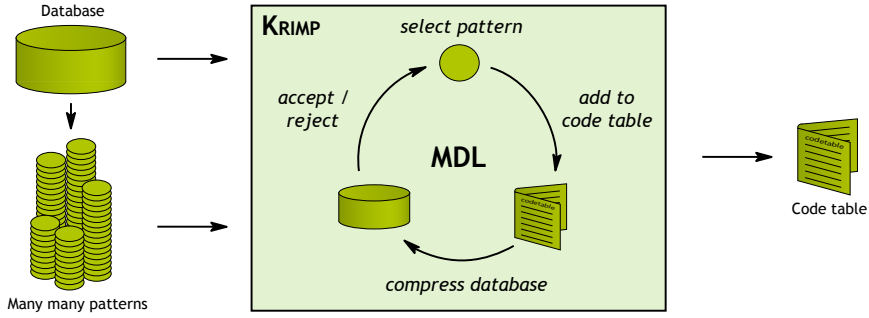


Figure 2.1: KRIMP in action

The Krimp algorithm

We now have the ingredients for the basic version of our compression algorithm:

- Start with the standard code table ST.
- Add the candidate itemsets from $\mathcal{F}$ one by one. Each time, take the itemset that is maximal w.r.t. the standard candidate order. Cover the database using the standard cover algorithm. If the resulting encoding provides a smaller compressed size, keep it. Otherwise, discard it permanently.

This basic scheme is formalised as the KRIMP algorithm given as Algorithm 3. For the choice of the name: 'krimp' is Dutch for 'to shrink'. The KRIMP pattern selection process is illustrated in Figure 2.1.

KRIMP takes as input a database $\mathcal{D}$ and a candidate set $\mathcal{F}$. The result is the best code table the algorithm has seen, w.r.t. the Minimal Coding Set Problem.

Now, it may seem that each iteration of KRIMP can only lessen the *usage* of an itemset in CT. For, if $F_1 \cap F_2 \neq \emptyset$ and F_2 is used before F_1 by the standard cover function, the usage of F_1 will go down (provided the support of F_2 does not equal zero). While this is true, it is not the whole story. Because, what happens if we now add an itemset F_3, which is used before F_2 such that:

$$F_1 \cap F_3 = \emptyset \quad \text{and} \quad F_2 \cap F_3 \neq \emptyset$$

The usage of F_2 will go down, while the usage of F_1 will go up again; by the same amount, actually. So, taking this into consideration, even code table elements with zero usage cannot be removed without consequence. However, since they are not used in the actual encoding, they are not taken into account while calculating the total compressed size for the current solution.

Algorithm 3 The Krimp Algorithm

Require: A transaction database $\mathcal{D}$ and a candidate set $\mathcal{F}$, both over a set of items $\mathcal{I}$

Ensure: A solution to the Minimal Coding Set problem, code table CT

Krimp $(\mathcal{D}, \mathcal{F})$:

1: $CT \leftarrow$ **Standard Code Table**$(\mathcal{D})$
2: $\mathcal{F}_o \leftarrow \mathcal{F}$ in **Standard Candidate Order**
3: **for all** $F \in \mathcal{F}_o \setminus \mathcal{I}$ **do**
4: $\quad CT_c \leftarrow (CT \cup F)$ in **Standard Cover Order**
5: $\quad$ **if** $L(\mathcal{D}, CT_c) < L(\mathcal{D}, CT)$ **then**
6: $\quad\quad CT \leftarrow CT_c$
7: $\quad$ **end if**
8: **end for**
9: **return** CT

In the end, itemsets with zero usage can be safely removed though. After all, they do not code, so they are not part of the optimal answer that should consist of the smallest coding set. Since the singletons are required in a code table by definition, these remain.

Pruning

That said, we can't be sure that leaving itemsets with a very low usage count in CT is the best way to go. As these have a very small probability, their respective codes will be very long. Such long codes may make better code tables unreachable for the greedy algorithm; it may get stuck in a local optimum. As an example, consider the following three code tables:

$$\begin{aligned} CT_1 &= \{\{X_1, X_2\}, \{X_1\}, \{X_2\}, \{X_3\}\} \\ CT_2 &= \{\{X_1, X_2, X_3\}, \{X_1, X_2\}, \{X_1\}, \{X_2\}, \{X_3\}\} \\ CT_3 &= \{\{X_1, X_2, X_3\}, \{X_1\}, \{X_2\}, \{X_3\}\} \end{aligned}$$

Assume that $supp_{\mathcal{D}}(\{X_1, X_2, X_3\}) = supp_{\mathcal{D}}(\{X_1, X_2\}) - 1$. Given these assumptions, standard Krimp will never consider CT_3, but it is very well possible that $L(\mathcal{D}, CT_3) < L(\mathcal{D}, CT_2)$ and that CT_3 provides access to a branch of the search space that is otherwise left unvisited. To allow for searching in this direction, we can *prune* the code table that Krimp is considering.

There are many possibilities to this end. The most obvious strategy is to check the attained compression of all valid subsets of CT including the candidate itemset F, i.e. $\{\ CT_p \subseteq CT \mid F \in CT_p \ \wedge \ \mathcal{I} \subset CT_p\ \}$, and choose CT_p with minimal $L(\mathcal{D}, CT_p)$. In other words, prune when a candidate itemset is added to CT, but before the acceptance decision. Clearly, such a

pre-acceptance pruning approach implies a huge amount of extra computation. Since we are after a fast and well-performing heuristic we do not consider this strategy.

A more efficient alternative is post-acceptance pruning. That is, we only prune when F is accepted: when candidate code table $CT_c = CT \cup F$ is better than CT, i.e. $L(\mathcal{D}, CT_c) < L(\mathcal{D}, CT)$, we consider its valid subsets. This effectively reduces the pruning search space, as only few candidate itemsets will be accepted.

To cut the pruning search space further, we do not consider all valid subsets of CT, but iteratively consider for removal those itemsets $X \in CT$ for which $usage_{\mathcal{D}}(X)$ has decreased. The rationale is that for these itemsets we know that their code lengths have increased; therefore, it is possible that these sets now harm the compression.

In line with the standard order philosophy, we first consider the itemset with the smallest usage and thus the longest code. If by pruning an itemset the total encoded size decreases, we permanently remove it from the code table. Further, we then update the list of prune candidates with those item sets whose usage consequently decreased. This post-acceptance pruning strategy is formalised in Algorithm 4. We refer to the version of Krimp that employs this pruning strategy (which would be on line 6 of Algorithm 3) as Krimp with pruning. In Section 2.7 we will show that employing pruning improves the performance of Krimp.

Algorithm 4 Code Table Post-Acceptance Pruning

Require: Code tables CT_c and CT, for a transaction database $\mathcal{D}$ over a set of items $\mathcal{I}$, where $\{X \in CT\} \subset \{Y \in CT_c\}$ and $L(\mathcal{D}, CT_c) < L(\mathcal{D}, CT)$.
Ensure: Pruned code table CT_p, with $L(\mathcal{D}, CT_p) \leq L(\mathcal{D}, CT_c)$, $CT_p \subseteq CT_c$.

PruneCodeTable $(CT_c, CT, \mathcal{D})$:

1: $CT_p \leftarrow CT_c$
2: $PruneSet \leftarrow \{\ X \in CT \mid usage_{CT_c}(X) < usage_{CT}(X)\ \}$
3: **while** $PruneSet \neq \emptyset$ **do**
4: $\quad PruneCand \leftarrow X \in PruneSet$ with lowest $usage_{CT_p}(X)$
5: $\quad PruneSet \leftarrow PruneSet \setminus PruneCand$
6: $\quad CT_t \leftarrow CT_p \setminus PruneCand$
7: $\quad$ **if** $L(\mathcal{D}, CT_t) < L(\mathcal{D}, CT_p)$ **then**
8: $\quad\quad PruneSet \leftarrow PruneSet \cup \{\ X \in CT_t \mid usage_{CT_t}(X) < usage_{CT_p}(X)\ \}$
9: $\quad\quad CT_p \leftarrow CT_t$
10: $\quad$ **end if**
11: **end while**
12: **return** CT_p

Complexity

Here we analyse the complexity of the Krimp algorithms step–by–step. We start with time-complexity, after which we cover memory-complexity.

Given a set of (frequent) itemsets $\mathcal{F}$, we first order this set, requiring $\mathrm{O}(|\mathcal{F}| \log |\mathcal{F}|)$ time. Then, every element $F \in \mathcal{F}$ is considered once. Using a hash-table implementation we need only $\mathrm{O}(1)$ to insert an element at the right position in CT, keeping CT ordered. To calculate the total encoded size $L(\mathcal{D}, CT)$, the *cover* function is applied to each $t \in \mathcal{D}$. For this, the standard cover function considers each $X \in CT$ once for a t. Checking whether X is an (uncovered) subset of t takes at most $\mathrm{O}(|\mathcal{I}|)$. Therefore, covering the full database takes $\mathrm{O}(|\mathcal{D}| \times |CT| \times |\mathcal{I}|)$ time. Then, optimal code lengths and the total compressed size can be computed in $\mathrm{O}(|CT|)$.

Note that we know the code table will grow to at most $|\mathcal{F}|$ elements. So, given a set of (frequent) itemsets $\mathcal{F}$ and a *cover* function that considers the elements of the code table in a static order, the worst-case time-complexity of the Krimp algorithm without pruning is

$$\mathrm{O}(\ |\mathcal{F}| \log |\mathcal{F}| + |\mathcal{F}| \times (|\mathcal{D}| \times |\mathcal{F}| \times |\mathcal{I}| + |\mathcal{F}|)\ ).$$

When we do employ pruning, in the worst-case we have to reconsider each element in CT after accepting each $F \in \mathcal{F}$,

$$\mathrm{O}(\ |\mathcal{F}| \log |\mathcal{F}| + |\mathcal{F}|^2 \times (|\mathcal{D}| \times |\mathcal{F}| \times |\mathcal{I}| + |\mathcal{F}|)\ ).$$

This looks quite horrendous. However, it is not as bad as it seems.

First of all, due to MDL, the number of elements in the code table is very small, $|CT| \ll |\mathcal{D}| \ll |\mathcal{F}|$, in particular when pruning is enabled. In fact, this number (typically 100 to 1000) can be regarded as a constant, removing it from the big-O notation. Therefore,

$$\mathrm{O}(\ |\mathcal{F}| \log |\mathcal{F}| + |\mathcal{D}| \times |\mathcal{F}| \times |\mathcal{I}|\ )$$

is a better estimate for the time-complexity for Krimp with or without pruning enabled.

Next, for $\mathcal{I}$ of reasonable size (say, up to 1000), bitmaps can be used to represent the itemsets. This allows for subset checking in $\mathrm{O}(1)$, again removing a term from the complexity. Further, for any new candidate code table itemset $F \in \mathcal{F}$, the database needs only to be covered partially; so instead of all $|\mathcal{D}|$ transactions only those d transactions in which F occurs need to be covered. If $\mathcal{D}$ is large and the *minsup* threshold is low, d is generally very small ($d \ll |\mathcal{D}|$) and can be regarded as a constant. So, in the end we have

$$\mathrm{O}(\ |\mathcal{F}| \log |\mathcal{F}| + |\mathcal{F}|\ ).$$

Now, we consider the order of the memory requirements of KRIMP. The worst-case memory requirements of the KRIMP algorithms are

$$\mathrm{O}(\ |\mathcal{F}| + |\mathcal{D}| + |\mathcal{F}|\ ).$$

Again, as the code table is dwarfed by the size of the database, it can be regarded a (small) constant. The major part is the storage of the candidate code table elements. Sorting these can be done in place. As it is iterated in order, it can be handled from the hard drive without much performance loss. Preferably, the database is kept resident, as it is covered many many times.

2.4 Interlude

Before we continue with more theory, we will first present some results on a small number of datasets to provide the reader with some measure and intuition on the performance of KRIMP. To this end, we ran KRIMP with post-acceptance pruning on six datasets, using all frequent itemsets mined at $minsup = 1$ as candidates. The results of these experiments are shown in Table 2.2. Per dataset, we show the number of transactions and the number of candidate itemsets. From these latter figures, the problem of the pattern explosion becomes clear: up to 5.5 billion itemsets can be mined from the *Mushroom* database, which consists of only 8124 transactions. It also shows that KRIMP successfully battles this explosion, by selecting only hundreds of itemsets from millions up to billions. For example, from the 5.5 billion for *Mushroom*, only 442 itemsets are chosen; a reduction of 7 orders of magnitude.

For the other datasets, we observe the same trend. In each case, fewer than 2000 itemsets are selected, and reductions of many orders of magnitude are attained. The number of selected itemsets depends mainly on the characteristics of the data. These itemsets, or the code tables they form, compress the data to a fraction of its original size. This indicates that very characteristic itemsets are chosen, and that the selections are non-redundant. Further, the timings for these experiments show that the compression-based selection process, although computationally complex, is a viable approach in practice. The selection of the above mentioned 442 itemsets from 5.5 billion itemsets takes under 4 hours. For the *Adult* database, KRIMP considers over 400.000 itemsets per second, and is limited not by the CPUs but by the rate with which the itemsets can be read from the hard disk.

Given this small sample of results, we now know that indeed few, characteristic and non-redundant itemsets are selected by KRIMP, in number many orders smaller than the complete frequent itemset collections. However, this leaves the question of how *good* are the returned pattern sets?

Table 2.2: Results of running Krimp on a few datasets.

			Krimp		
Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{F}\rvert$	$\lvert CT \setminus \mathcal{I}\rvert$	$\frac{L(\mathcal{D},CT)}{L(\mathcal{D},ST)}\%$	*time*
Adult	48842	58461763	1303	24.4	2m25
Chess (kr–k)	28056	373421	1684	61.6	13s
Led7	3200	15250	152	28.6	0.05s
Letter recognition	20000	580968767	1780	35.7	52m33
Mushroom	8124	5574930437	442	20.6	3h40
Pen digits	10992	459191636	1247	42.3	31m33

For all datasets the candidate set $\mathcal{F}$ was mined with $minsup = 1$, and Krimp with post-acceptance pruning was used. For Krimp, the size of the resulting code table (minus the singletons), the compression ratio and the run time is given. The compression ratio is the encoded size of the database with the obtained code table divided by the encoded size with the standard code table. Timings recorded on quad-core 3.0 Ghz Xeon machines.

2.5 Classification by Compression

In this section, we describe a method to verify the quality of the Krimp selection in an independent way. To be more precise, we introduce a simple classification scheme based on code tables, previously published as [78]. We answer the quality question by answering the question: how well do Krimp code tables classify? For this, classification performance is compared to that of state-of-the-art classifiers in the Section 2.7.

Classification through MDL

If we assume that our database of transactions is an i.i.d. sample from some underlying data distribution, we expect the optimal code table for this database to compress an arbitrary transaction sampled from this distribution well. We make this intuition more formal in Lemma 4.

We say that the itemsets in CT are *independent* if any co-occurrence of two itemsets $X, Y \in CT$ in the cover of a transaction is independent. That is, $P(XY) = P(X)P(Y)$. Clearly, this is a Naïve Bayes [131] like assumption.

Lemma 4. *Let $\mathcal{D}$ be a bag of transactions over $\mathcal{I}$,* cover *a cover function, CT the optimal code table for $\mathcal{D}$ and t an arbitrary transaction over $\mathcal{I}$. Then, if*

the itemsets $X \in cover(CT, t)$ are independent,

$$L(t \mid CT) = -\log\left(P(t \mid \mathcal{D})\right).$$

Proof.

$$\begin{aligned} L(t \mid CT) &= \sum_{X \in cover(CT,t)} L(code_{CT}(X)) \\ &= \sum_{X \in cover(CT,t)} -\log\left(P(X \mid \mathcal{D})\right) \\ &= -\log\left(\prod_{X \in cover(CT,t)} P(X \mid \mathcal{D})\right) \\ &= -\log\left(P(t \mid \mathcal{D})\right). \end{aligned}$$

□

The last equation is only valid under the Naïve Bayes like assumption, which might be violated. However, if there are itemsets $X, Y \in CT$ such that $P(XY) > P(X)P(Y)$, we would expect an itemset $Z \in CT$ such that $X, Y \subset Z$. Therefore, we do not expect this assumption to be overly optimistic.

Now, assume that we have two databases generated from two different underlying distributions, with corresponding optimal code tables. For a new transaction that is generated under one of the two distributions, we can now decide to which distribution it most likely belongs. That is, under the Naïve Bayes assumption, we have the following lemma.

Lemma 5. *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two bags of transactions over $\mathcal{I}$, sampled from two different distributions,* cover *a cover function, and t an arbitrary transaction over $\mathcal{I}$. Let CT_1 and CT_2 be the optimal code tables for resp. $\mathcal{D}_1$ and $\mathcal{D}_2$. Then, from Lemma 4 it follows that*

$$L(t \mid CT_1) > L(t \mid CT_2) \;\Rightarrow\; P(t \mid \mathcal{D}_1) < P(t \mid \mathcal{D}_2).$$

Hence, the Bayes optimal choice is to assign t to the distribution that leads to the shortest code length.

The Krimp classifier

The previous subsection, with Lemma 5 in particular, suggests a straightforward classification algorithm based on Krimp code tables. This provides an independent way to assess the quality of the resulting code tables. The Krimp Classifier is given in Algorithm 5. The Krimp classification process is illustrated in Figure 2.2.

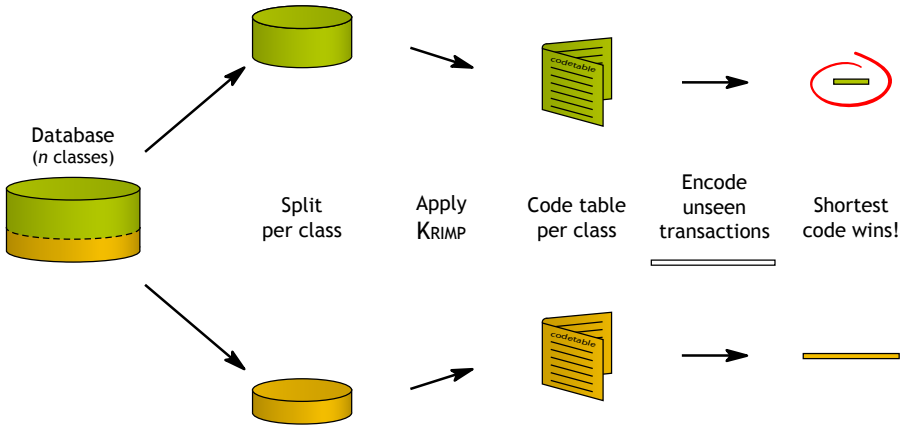


Figure 2.2: The Krimp Classifier in action

The classifier consists of a code table per class. To build it, a database with class labels is needed. This database is split according to class, after which the class labels are removed from all transactions. Krimp is applied to each of the single-class databases, resulting in a code table per class. At the very end, after any pruning has been applied, each code table is Laplace corrected: the *usage* of each itemset in CT_k is increased by one. This ensures that all itemsets in CT_k have non-zero usage, therefore have a code, i.e. their code length can be calculated, and thus, that any arbitrary transaction $t \subseteq \mathcal{I}$ can be encoded.

Algorithm 5 The Krimp Classifier

Require: A database $\mathcal{D}$ with class labels, and transaction t, both over set of items $\mathcal{I}$

Ensure: The class label assigned to t

KrimpClassify $(t, \mathcal{D})$:

1: $K \leftarrow \{$ class labels of $\mathcal{D}$ $\}$
2: $\{\mathcal{D}_k\} \leftarrow$ split $\mathcal{D}$ on K, remove each $k \in K$ from each $t \in \mathcal{D}$
3: **for all** $\mathcal{D}_k$ **do**
4: $\quad \mathcal{F}_k \leftarrow$ MineCandidates$(\mathcal{D}_k)$
5: $\quad CT_k \leftarrow$ Krimp$(\mathcal{D}_k, \mathcal{F}_k)$
6: $\quad$ **for** $X \in CT_k$ **do** $usage_{CT_k}(X) \leftarrow usage_{CT_k}(X) + 1$
7: **end for**
8: **return** $\arg\min_{k \in K} L(t \mid CT_k)$

(Recall that we require a code table to always contain all singleton itemsets.)

When the compressors have been constructed, classifying a transaction is trivial. Simply assign the class label belonging to the code table that provides the minimal encoded length for the transaction.

2.6 Related Work

MDL in data mining

MDL was introduced by [108] as a noise-robust model selection technique. In the limit, refined MDL is asymptotically the same as the Bayes Information Criterion (BIC), but the two may differ (strongly) on finite data samples [53]. We are not the first to use MDL, nor are we the first to use MDL in data mining or machine learning. Many, if not all, data mining problems can be related to Kolmogorov Complexity, which means they can be practically solved through compression [42], e.g. clustering (unsupervised learning), classification (supervised learning), distance measurement. Other examples include feature selection [105], defining a parameter-free distance measure on sequential data [67,68], discovering communities in matrices [26], and evolving graphs [115].

Pattern selection

Most, if not all pattern mining approaches suffer from the pattern explosion. As discussed before, its cause lies primarily in the large redundancy in the returned pattern sets. This has long since been recognised as a problem and has received ample attention.

To address this problem, closed [103] and non-derivable [25] itemsets have been proposed, which both provide a concise lossless representation of the original itemsets. However, these methods deteriorate under small amounts of noise. Similar, but providing a partial (i.e. lossy) representation, are maximal itemsets [12] and δ-free sets [36]. Along these lines, Yan et al [135] proposed a method that selects k representative patterns that together summarize the frequent pattern set.

Recently, the approach of finding small subsets of informative patterns that describe the database has attracted a significant amount of research [20,70,95]. First, there are the methods that provide a lossy description of the data. These strive to describe just part of the data, and as such may overlook important interactions. Summarization as proposed by [28] is a compression-based approach that identifies a group of itemsets such that each transaction is summarised by one itemset with as little loss of information as possible. [129] find summary sets, sets of itemsets that contain the largest frequent itemset covering each transaction.

Pattern Teams [69] are groups of most-informative length-k itemsets [70]. These are exhaustively selected through an external criterion, e.g. joint entropy or classification accuracy. As this approach is computationally intensive, the number of team members is typically < 10. [20] proposed a similar selection method that can consider larger pattern sets. However, it also requires the user to choose a quality measure to which the pattern set has to be optimized, unlike our parameter-free and lossless method.

Second, in the category of lossless data description, we recently [113] introduced the MDL-based Krimp algorithm. In this chapter we extend the theory, tune the pruning techniques and thoroughly verify the validity of the chosen heuristics, as well as provide extensive experimental evaluation of the quality of the returned code tables.

Tiling [48] is closely related to our approach. A tiling is a cover of the database by a group of (overlapping) item sets. Itemsets with maximal uncovered area are selected, i.e. as few as possible itemsets cover the data. Unlike our approach, model complexity is not explicitly taken into account. Another major difference in the outcome is that Krimp selects more specific (longer) itemsets. [134] proposed a slight reformulation of Tiling that allows tiles to also cover transactions in which not all its items are present.

Two approaches inspired by Krimp are Pack [118] and LESS [60]. Both approaches consider the data 0/1 symmetric, unlike here, where we only regard items that are present (1s). LESS employs a generalised Krimp encoding to select only tens of low-entropy sets [59] as lossless data descriptions, but attains worse compression ratios than Krimp. Pack does provide a significant improvement in that regard. It employs decision trees to succinctly transmit individual attributes, and these models can be built from data or candidate sets. Typically, Pack selects many more itemsets than Krimp.

Our approach seems related to the set cover problem [65], as both try to cover the data with sets. Although NP-complete, fast approximation algorithms exist for set cover. These are not applicable for our setup though, as in set cover the complexity of the model is not taken into account. Another difference is that we do not allow overlap between itemsets. As optimal compression is the goal, it makes intuitive sense that overlapping elements may lead to shorter encodings, but it is not immediately clear how to achieve this in a fast heuristic.

Classification

A lot of classification algorithms have been proposed, many of which fall into either the class of rule-induction-based or that of association-rule-based methods. Because we use classification as a quality measure for the patterns that Krimp picks, we will compare our results with those obtained by some of the best existing classifiers. Such comparison can be done with rule-induction-

based methods such as C4.5 [106], FOIL [107] and CPAR [136]. [91] use MDL to prune decision trees for classification. However, we are more interested in the comparison to association-rule-based algorithms like iCAEP [137], HARMONY [128], CBA [84] and LB [93] as these also employ a collection of itemsets for classification. Because we argued that our method is strongly linked to the principle of Naïve Bayes (NB) [40] it is imperative we compare to it. Because, opposed to KRIMP, these methods were devised with the goal of classification in mind, we would expect them to (slightly) outperform the KRIMP classifier.

2.7 Experiments

In this section we experimentally evaluate the KRIMP algorithms, and the underlying heuristics, and assess the quality of the resulting code tables.

We first describe our setup in the next subsection and second the datasets we use in the experiments. Then, we start our evaluation of KRIMP by looking at how many itemsets are selected and what compression ratios are attained. The stability of these results, and whether these rely on specific itemsets is explored next. Afterwards, we test through swap-randomisation whether the code tables model relevant structure. In the next subsection the quality of the code tables is independently validated through classification. Last, we evaluate the cover and candidate order heuristics of KRIMP.

Setup

We use the shorthand notation $L\%$ to denote the relative total compressed size of $\mathcal{D}$,

$$\frac{L(\mathcal{D}, CT)}{L(\mathcal{D}, ST)}\%,$$

wherever $\mathcal{D}$ is clear from context. As candidates, $\mathcal{F}$, we typically use all frequent itemsets mined at $minsup = 1$, unless indicated otherwise. We use the AFOPT miner [85], taken from the FIMI repository [50], to mine (closed) frequent itemsets. The reported KRIMP timings are of the selection process only and do not include the mining and sorting of the candidate itemset collections. All experiments were conducted on quad-core Xeon 3.0 GHz systems (in black casing) running Windows Server 2003. Reported timings are based on four-threaded runs, again, unless stated otherwise.

Data

For the experimental validation of our methods we use a wide range of freely available datasets. From the LUCS/KDD data set repository [33] we take

Table 2.3: Statistics of the datasets used in the experiments.

Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert$	*density*	# *classes*	$L(\mathcal{D}\lvert ST)$
Accidents	340183	468	7.22	-	74592568
Adult	48842	97	15.33	2	3569724
Anneal	898	71	20.15	5	62827
BMS-pos	515597	1657	0.39	-	25321966
BMS-webview 1	59602	497	0.51	-	1173962
BMS-webview 2	77512	3340	0.14	-	3747293
Breast	699	16	62.36	2	27112
Chess (k–k)	3196	75	49.33	2	687120
Chess (kr–k)	28056	58	12.07	18	1083046
Connect–4	67557	129	33.33	3	17774814
DNA amplification	4590	392	1.47	-	212640
Heart	303	50	27.96	5	20543
Ionosphere	351	157	22.29	2	81630
Iris	150	19	26.32	3	3058
Led7	3200	24	33.33	10	107091
Letter recognition	20000	102	16.67	26	1980244
Mammals	2183	121	20.5	-	320094
Mushroom	8124	119	19.33	2	1111287
Nursery	12960	32	28.13	5	569042
Page blocks	5473	44	25	5	216552
Pen digits	10992	86	19.77	10	1140795
Pima	768	38	23.68	2	26250
Pumsbstar	49046	2088	2.42	-	19209514
Retail	88162	16470	0.06	-	10237244
Tic–tac–toe	958	29	34.48	2	45977
Waveform	5000	101	21.78	3	656084
Wine	178	68	20.59	3	14101

Per dataset the number of transactions, the number of attributes, the density (percentage of 1's) and the number of bits required by Krimp to compress the data using the singleton-only standard code table ST.

some of the largest and most dense databases. We transformed the *Connect-4* dataset to a slightly less dense format by removing all 'empty-field' items. From the FIMI repository [50] we use the BMS[4] datasets [71]. Further, we use the *Mammals* presence and *DNA Amplification* databases. The former consists of presence records of European mammals[5] within geographical areas of 50×50 kilometers [96]. The latter is data on DNA copy number amplifications. Such copies activate oncogenes and are hallmarks of nearly all advanced tumours [101]. Amplified genes represent attractive targets for therapy, diagnostics and prognostics.

The details for these datasets are depicted in Table 2.3. For each database we show the number of attributes, the number of transactions and the density: the percentage of 'present' attributes. Last, we provide the total compressed size in bits as encoded by the singleton-only standard code tables ST.

Selection

We first evaluate the question whether Krimp provides an answer to the pattern explosion. To this end, we ran Krimp on 27 datasets, and analysed the outcome code tables, with and without post-acceptance pruning. The results of these experiments are shown as Table 2.4. As candidates itemset collections we mined frequent itemsets of the indicated *minsup* thresholds. These were chosen as low as possible, either storage-wise or computationally feasible.

The main result shown in the table is the reduction attained by the selection process: up to 7 orders of magnitude. While the candidate sets contain millions up to billions of itemsets, the resulting code tables typically contain hundreds to thousands of non-singleton itemsets. These selected itemsets compress the data well, typically requiring only a quarter to half of the bits of the independent ST encoding. Dense datasets are compressed very well. For *Adult* and *Mushroom*, ratios of resp. 24% and 21% are noted. Sparse data, on the other hand, typically contains little structure. We see that such datasets (e.g. the *Retail* and *BMS* datasets) indeed prove difficult to compress; relatively many itemsets are required to provide a meagre compression.

Comparing between Krimp with and without post-acceptance pruning, we see that enabling pruning provides the best: fewer itemsets (~ 1000, on average) are returned, which provide better compression (avg. 2% improvement). For *Accidents* and *BMS-pos* the difference in the number of selected itemsets is a factor of 10. The average length of the itemsets in the code tables is about the same, with resp. 5.9 and 5.7 with and without pruning. However, the average *usage* of these itemsets differs more, with averages of resp. 80.7 and 48.2.

[4] We wish to thank Blue Martini Software for contributing the KDD Cup 2000 data.

[5] The full version of the mammal dataset is available for research purposes upon request from the Societas Europaea Mammalogica. `http://www.european-mammals.org`

Table 2.4: Results of Krimp with and without post-acceptance pruning

			Krimp			
			w/o pruning		w/ pruning	
Dataset	*minsup*	$\lvert\mathcal{F}\rvert$	$\lvert CT \setminus \mathcal{I}\rvert$	$L\%$	$\lvert CT \setminus \mathcal{I}\rvert$	$L\%$
Accidents	50000	2881487	4046	55.4	467	55.1
Adult	1	58461763	1914	24.9	1303	24.4
Anneal	1	4223999	133	37.5	102	35.3
BMS-pos	100	5711447	14628	82.7	1657	81.8
BMS-wv1	32	1531980297	960	86.6	736	86.2
BMS-wv2	10	4440334	5475	84.4	4585	84.0
Breast	1	9919	35	17.4	30	17.0
Chess (k–k)	319	4603732933	691	30.9	280	27.3
Chess (kr–k)	1	373421	2203	62.9	1684	61.6
Connect–4	1	233142539	4525	11.5	2036	10.9
DNA amp	9	312073710	417	38.6	326	37.9
Heart	1	1922983	108	61.4	79	57.7
Ionosphere	35	225577741	235	63.4	164	61.3
Iris	1	543	13	48.2	13	48.2
Led7	1	15250	194	29.5	152	28.6
Letter	1	580968767	3758	43.3	1780	35.7
Mammals	200	93808243	597	50.4	316	48.4
Mushroom	1	5574930437	689	22.2	442	20.6
Nursery	1	307591	356	45.9	260	45.5
Page blocks	1	63599	56	5.1	53	5.0
Pen digits	1	459191636	2794	48.8	1247	42.3
Pima	1	28845	72	36.3	58	34.8
Pumsbstar	11120	272580786	734	51.0	389	50.9
Retail	4	4106008	7786	98.1	6264	97.7
Tic–tac–toe	1	250985	232	65.0	160	62.8
Waveform	5	465620240	1820	55.6	921	44.7
Wine	1	2276446	76	80.9	63	77.4

Per dataset, the *minsup* for mining frequent itemsets, and the size of the resulting candidate set $\mathcal{F}$. For Krimp without and with post-acceptance pruning enabled, the number of non-singleton elements in the returned code tables and the attained compression ratios.

As post-acceptance pruning provides improved performance, from now onward we employ KRIMP with post-acceptance pruning, unless indicated otherwise. Further, due to the differences in code table size, experiments with pruning typically execute faster than those without pruning.

Next, we examine the development in number of selected itemsets w.r.t. the number of candidate itemsets. For the *Mushroom* database, the top graph of Figure 2.3 shows the size of the candidate set and size of the corresponding code table for varying *minsup* thresholds. While we see that the number of candidate itemsets grows exponentially, to 5.5 billion for $minsup = 1$, the number of selected itemsets stabilises at around 400. This stabilisation is typical for all datasets, with the actual number being dependent on the characteristics of the data.

This raises the question whether the total compressed size also stabilises. In the bottom graph of Figure 2.3, we plot the total compressed size of the database for the same range of *minsup*. From the graph it is clear that this is not the case: the compressed size decreases continuously, it does not stabilise. Again, this is typical behaviour; especially for sparse data we have recorded steep descents at low *minsup* values. As the number of itemsets in the code table is stable, we thus know that itemsets are being replaced by better ones. Further, note that the compressed size of the code table is dwarfed by the compressed size of the database. This is especially the case for large datasets.

Back to the top graph of the figure, we see a linear correlation between the run time and the number of candidate sets. The correlation factor depends heavily on the data characteristics; its density, the number of items and the number of transactions. For this experiment, we observed 200,000 to 400,000 candidates considered per second, the performance being limited by IO.

In the top graph of Figure 2.4 we provide an overview of the differences in the sizes of the candidate sets and code tables, and in the bottom graph the run times recorded for these experiments. Those experiments for which the run time bars are missing finished within one second. The bottom graph shows that the run times are low.

Letting KRIMP consider the largest candidate sets, billions of itemsets, takes up to a couple of hours. The actual speed (candidates per second) mainly depends on the support of the itemsets (the number of transactions that have to be covered). The speed at which our current implementation considers candidate itemsets typically ramps up to thousands, even hundreds of thousands, per second.

Stability

Here, we verify the stability of KRIMP w.r.t. different candidate sets. First we investigate whether good results can be attained without the itemsets normally chosen in the code table. Given the large redundancy in the frequent pattern

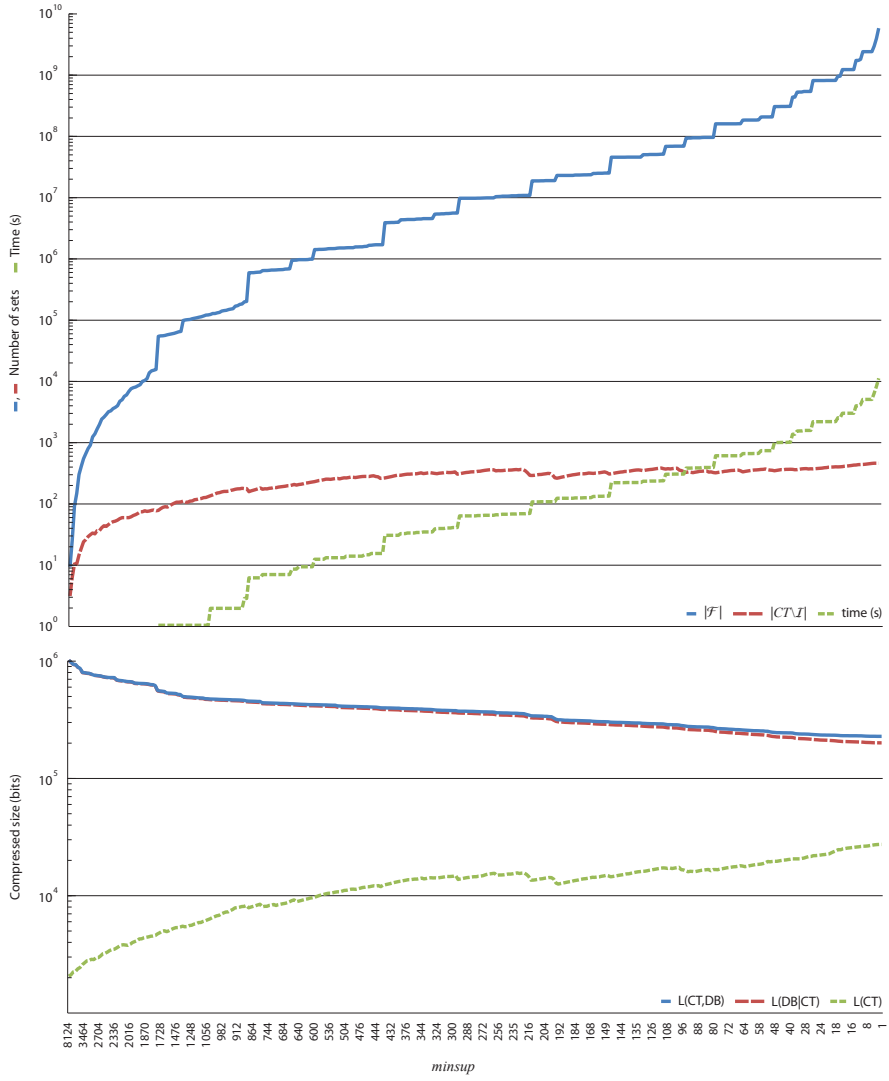


Figure 2.3: Running Krimp with post-acceptance pruning on the *Mushroom* dataset using 5.5 billion frequent itemsets as candidates ($minsup = 1$). (top) Per $minsup$, the size of the candidate set, $|\mathcal{F}|$, the size of the code table, $|CT \backslash \mathcal{I}|$, and the runtime in seconds. (bottom) Per $minsup$, the size in bits of the code table, the data, and the total compressed size (resp. $L(CT)$, $L(\mathcal{D} \mid CT)$ and $L(\mathcal{D}, CT)$).

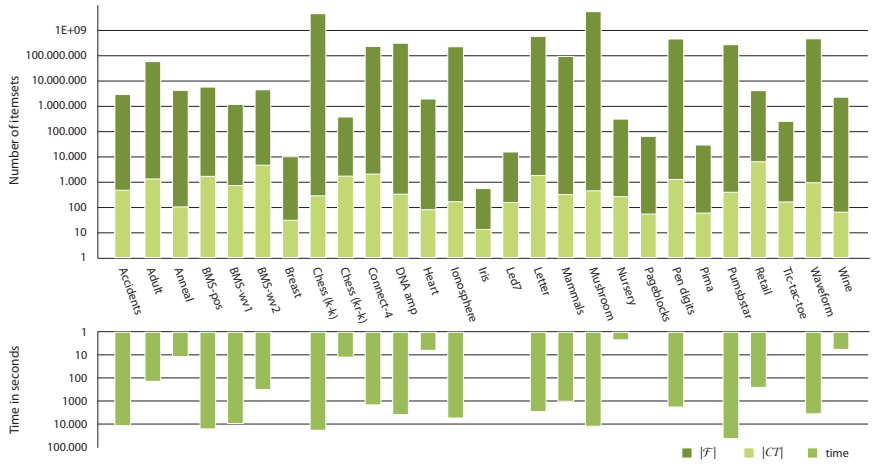


Figure 2.4: The results of KRIMP with post-acceptance pruning on the 27 datasets using the *minsup* thresholds of Table 2.4. The dark coloured bars show the number of itemsets in the candidate set, $|\mathcal{F}|$, the lighter coloured bars the number of non-singleton itemsets selected in the code table, $|CT \setminus \mathcal{I}|$, and the bottom graph the associated run times.

set, one would expect so.

To this end, we first ran KRIMP using candidate set $\mathcal{F}$ to obtain CT. Then, for each $X \in CS \setminus \mathcal{I}$ we ran KRIMP using the candidate set $\mathcal{F} \setminus \{X\}$. In addition, we also ran KRIMP using $\mathcal{F} \setminus \{X \in CS \setminus \mathcal{I}\}$.

As a code table typically consists of about 100 to 1000 elements, a considerable set of experiments is required per database. Therefore, we use five of the datasets. The results of these experiments are presented in Table 2.5. The minute differences in compression ratios show that the performance of KRIMP does not rely on just those specific itemsets in the original code tables. Excluding all elements from the original code table results in compression ratios up till only .5% worse than normal. This is expected, as in this setting all structure in the data captured by the original code table has to be described differently. The results of the individual itemset exclusion experiments, on the other hand, are virtually equal to the original. In fact, sometimes shorter (better) descriptions are found this way: the removed itemsets were in the way of ones that offer a better description.

Next, we consider excluding far more itemsets as candidates. That is, we use closed, as opposed to all, frequent itemsets as candidates for KRIMP. For

datasets with little noise the closed frequent pattern set can be much smaller and faster to mine and process. For the *minsup* thresholds depicted in Table 2.4, we mined both the complete and closed frequent itemset collections, and used these as candidate sets $\mathcal{F}$ for running Krimp. Due to crashes of the closed frequent itemset miner, we have no results for the *Chess (k–k)* dataset. Figure 2.5 shows the comparison between the results of either candidate set in terms of the relative compression ratio. The differences in performance are slight, with an average increase in $L\%$ of about 1%. The only exception seems *Ionosphere*, where a 12% difference is noted. Further, the resulting code tables are of approximately the same size; the 'closed' code tables consist of fewer itemsets: 10 on average. From these experiments we can conclude that closed frequent itemsets are a good alternative to be used as input for Krimp, especially if otherwise too many frequent itemsets are mined.

Table 2.5: Stability of the Krimp given candidate sets with exclusions

		$L\%$ *given candidates*		
Dataset	*minsup*	$\mathcal{F}$	$\mathcal{F} \setminus X$	$\mathcal{F} \setminus CT$
Chess (kr–k)	1	61.6	61.7 ± 0.21	61.6
Mushroom	1	24.7	24.7 ± 0.01	25.0
Nursery	1	45.5	45.4 ± 0.36	46.0
Pen digits	50	46.7	46.7 ± 0.12	47.2
Wine	1	77.4	77.4 ± 0.26	78.0

Per dataset, the *minsup* threshold at which frequent itemsets were mined as candidates $\mathcal{F}$ for Krimp. Further, the relative compression $L\%$ for running Krimp with $\mathcal{F}$, the average relative compression attained by excluding single original code table elements from $\mathcal{F}$, and the relative compression attained by excluding all itemsets normally chosen from $\mathcal{F}$, i.e. using $\mathcal{F} \setminus \{X \in CT \mid X \notin \mathcal{I}\}$ as candidates for Krimp. For *Mushroom* and *Pen digits* the closed frequent itemset collections were used as candidates.

Relevance

To evaluate whether Krimp code tables model relevant structure, we employ swap randomisation [49]. Swap randomisation is the process of randomising data to obscure the internal dependencies, while preserving the row and column margins of the data. The idea is that if the true structure of the data is

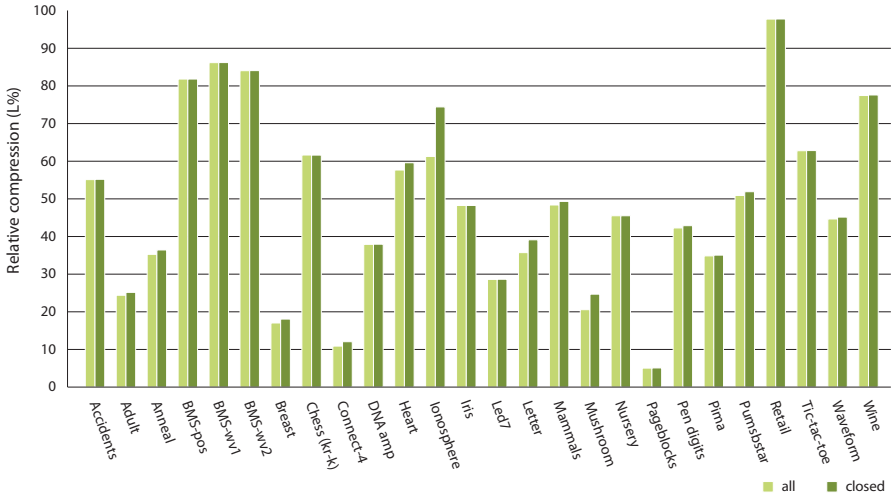


Figure 2.5: The results of KRIMP with post-acceptance pruning on 26 datasets using the *minsup* thresholds of Table 2.4. Per dataset, the upward bars indicated the relative total compressed size ($L\%$). As candidates, $\mathcal{F}$, all (left bars), and closed (right bars) frequent itemsets were used.

captured, there should be significant differences between the models found in the original and randomised datasets.

To this end, we compare the total compressed size of the actual data to those of 1000 swap randomised versions of the data. As this implies a large number of experiments, we have to restrict ourselves to a small selection of datasets. To this end, we chose three datasets with very different characteristics: *BMS-wv1*, *Nursery* and *Wine*. To get reasonable numbers of candidate itemsets (i.e. a few million) from the randomised data we use *minsup* thresholds of 1 for the latter two datasets and 35 for *BMS-wv1*. We apply as many swaps as there are 1's in the data.

Figure 2.6 shows the histogram of the total compressed sizes of these 1000 randomisations. The total compressed sizes of the original databases are indicated by arrows. The standard encoded size for these three databases, $L(\mathcal{D}, ST)$, are 1173962 bits, 569042 bits and 14100 bits, respectively.

The graphs show that the original data can be compressed significantly better than the randomised datasets (p-value of 0). Further quantitative results are shown in Table 2.6. Besides much better compression, we see that for *Nursery* and *Wine* the code tables induced on the original data contain fewer, but more specific (i.e. longer) itemsets. For *BMS-wv1* the randomised data

Table 2.6: Swap randomisation experiments

	Original data			*Swap randomised*		
Dataset	$\lvert CT \setminus \mathcal{I}\rvert$	$\overline{\lvert X\rvert}$	$L\%$	$\lvert CT \setminus \mathcal{I}\rvert$	$\overline{\lvert X\rvert}$	$L\%$
BMS-wv1	718	2.8	86.7	277.9 ± 7.3	2.7 ± 0.0	97.8 ± 0.1
Nursery	260	5.0	45.5	849.2 ± 19.9	4.0 ± 0.0	77.5 ± 0.1
Wine	67	3.8	77.4	76.8 ± 3.4	3.1 ± 0.1	93.1 ± 0.4

Results of 1000 independent swap randomisation experiments per dataset. As many swaps were applied as there are 1's in the data. By $\overline{\lvert X\rvert}$ we denote the average cardinality of itemsets $X \in CT \setminus \mathcal{I}$. The results for the swap randomisations are averaged over the 1000 experiments per dataset. As candidates for Krimp with post-acceptance pruning we used all frequent itemsets, $minsup = 1$, except for *BMS-wv1* for which we set $minsup = 35$.

is virtually incompressible with Krimp ($L\% \approx 98\%$), and as such much fewer itemsets are selected.

Classification

As an independent evaluation of the quality of the itemsets picked by Krimp, we compare the performance of the Krimp classifier (detailed in Section 2.5) to the performance of a wide range of well-known classifiers. Our hypothesis is that, if the code table-based classifier performs on-par, Krimp selects itemsets that are characteristic for the data.

We use the same *minsup* thresholds as we used for the compression experiments, listed in Table 2.4. Although the databases are now split on class before they are compressed by Krimp, these thresholds still provide large numbers of candidate itemsets and result in code tables that compress well.

It is not always beneficial to use the code tables obtained for the lowest *minsup*, as class sizes are often unbalanced or more structure is present in one class than in another. Also, over-fitting may occur for very low values of *minsup*. Therefore, we store code tables at fixed support intervals during the pattern selection process. During classification, we need to select one code table for each class. To this end, we 'pair' the code tables using two methods: *absolute* and *relative*. In absolute pairing, code tables that have been generated at the same support levels are matched. Relative pairing matches code tables of the same relative support between 100% and 1% of the maximum support value (per class, equals the number of transactions in a class). We evaluate all pairings and choose that one that maximises accuracy.

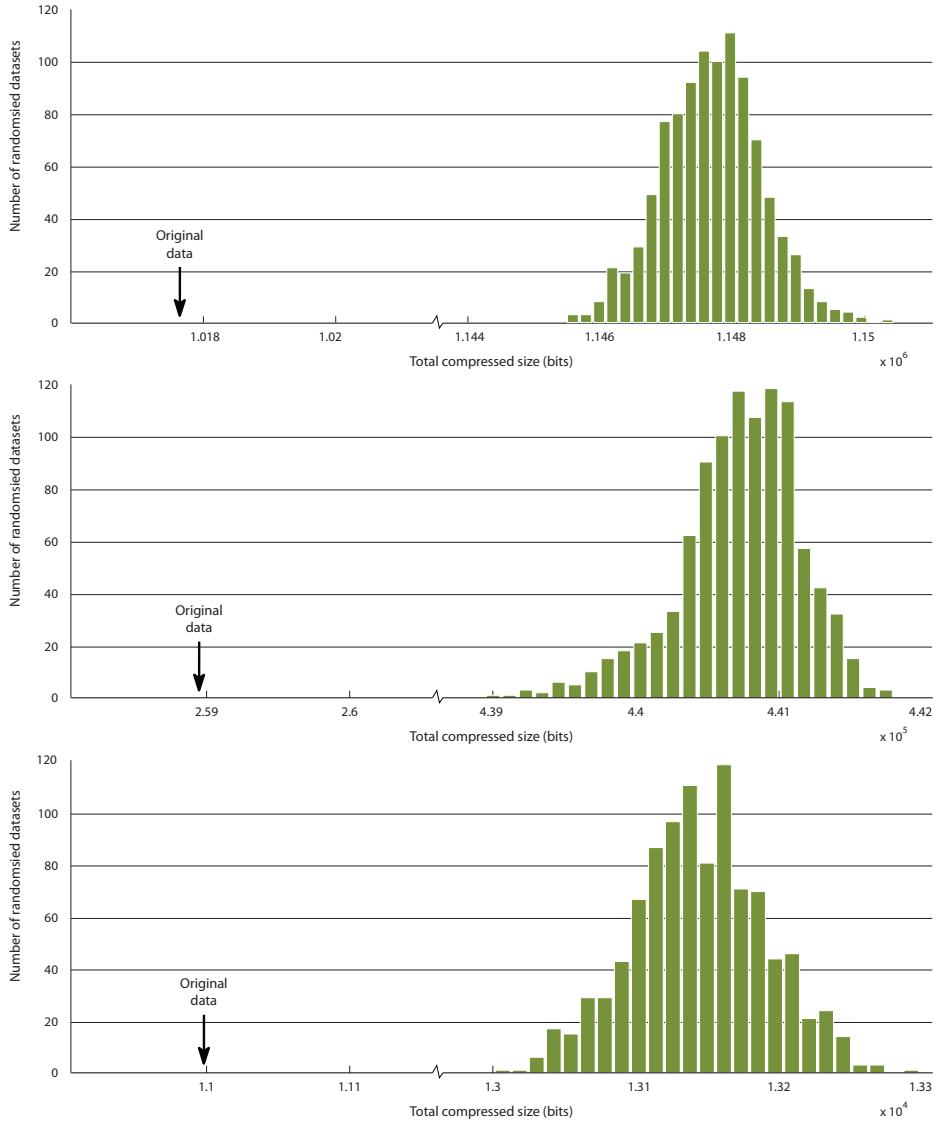


Figure 2.6: Histograms of the total compressed sizes of 1000 swap randomised *BMS-wv1* (top), *Nursery* (middle) and *Wine* (bottom) datasets, using all frequent itemsets as candidates for KRIMP with post-acceptance pruning. Total compressed sizes of the original datasets are indicated by the arrows. Note the jumps in compressed size on the x-axes.

Table 2.7: Results of Krimp classification, for all/closed frequent itemsets and without/with pruning

	Krimp	
Candidates	*w/o pruning*	*w/ pruning*
all	84.3 ± 14.2	84.5 ± 13.1
closed	84.0 ± 13.9	83.7 ± 14.2

For each combination of candidates and pruning, the average accuracy (%) over the 19 datasets from Table 2.8 is given. Standard deviation is high as a result of the diverse range of datasets used.

All results reported in this section have been obtained using 10-fold cross-validation. As performance measure we use accuracy, the percentage of true positives on the test data. We compare to results obtained with 6 state-of-the-art classifiers. These scores are taken from [16, 128, 137], which all used the same discretised datasets. Missing scores for C4.5, Naïve Bayes and SVM have been acquired using Weka [133]. Note that, in contrast to others, we use the sparse version of *Connect–4* described in Section 2.7.

Before we compare the Krimp classification performance to other methods, we verify whether there is a qualitative difference between using all or only closed frequent itemsets as candidates and using post-acceptance pruning or not. Table 2.7 shows that the variation in average accuracy is very small, but using all frequent itemsets as candidates with pruning gives the highest accuracy, making it an obvious choice for inspection of more detailed results in the rest of this subsection.

Classification results with all frequent itemsets as candidates and post-acceptance pruning are presented in Table 2.8, together with accuracies obtained with 6 competitive classifiers. Baseline accuracy is the (relative) size of the largest class in a dataset. For about half of the datasets, absolute pairing gives the maximum score, relative pairing gives the best results for the other cases. As expected, relative pairing performs well especially for datasets with small classes or unbalanced class sizes. In general though, the difference in maximum accuracy between the two types of pairings is very small, i.e. $< 1\%$. For a few datasets, the difference is more notable, $2 - 10\%$.

Looking at the scores, we observe that performance on most datasets is very similar for most algorithms. For *Pima*, for example, all accuracies are within a very small range. Because of this, it is important to note that performance may vary up to a few percent depending on the (random) partitioning used for cross-validation, especially for datasets having smaller classes. Since we took

Table 2.8: Results of KRIMP classification, compared to 6 state-of-the-art classifiers

Dataset	*base*	KRIMP	NB	C4.5	CBA	HRM	iCAEP	SVM
Adult	76.1	84.3	83.8	**85.4**	75.2	81.9	80.9	84.1
Anneal	76.2	96.6	97.0	90.4	**98.1**		95.1	97.4
Breast	65.5	94.1	97.1	95.4	95.3		**97.4**	69.6
Chess (k–k)	52.2	90.0	87.9	**99.5**	98.1		94.6	95.4
Chess (kr–k)	16.2	**57.9**	36.1	56.6		44.9		29.8
Connect–4	65.8	69.4	72.1	**81.0**		68.1	69.9	72.5
Heart	54.1	61.7	83.5	78.4	81.9		80.3	**84.2**
Ionosphere	64.1	91.0	89.5	92.0	**92.1**		90.6	88.6
Iris	33.3	**96.0**	93.3	94.7	92.9	94.7	93.3	**96.0**
Led7	11.0	**75.3**	74.0	71.5	71.9	74.6		73.8
Letter	4.1	70.9	64.1	**87.9**		76.8		67.8
Mushroom	51.8	**100**	99.7	**100**		99.9	99.8	99.7
Nursery	33.3	92.3	90.3	**97.1**	88.8	92.8	84.7	91.4
Page blocks	89.8	**92.6**	90.9	92.4	89.0	91.6		91.2
Pen digits	10.4	95.0	85.8	**96.6**		96.2		93.2
Pima	65.1	72.7	74.7	72.5	73.1	73.0	72.3	**77.3**
Tic–tac–toe	65.3	88.7	70.2	86.3	**100**	81.0	92.1	98.3
Waveform	33.9	77.1	80.8	70.4	75.3	80.5	81.7	**83.2**
Wine	39.9	**100**	89.9	87.9	91.6	63.0	98.9	98.3

For each dataset, baseline accuracy and accuracy (%) obtained with KRIMP classification is given, as well as accuracies obtained with 6 other classifiers are given. We use HRM as abbreviation for HARMONY. The highest accuracy per dataset is displayed in boldface. For KRIMP with post-acceptance pruning, per class, frequent itemsets mined at thresholds found in Table 2.4 were used as candidates. All results are 10-fold cross-validated.

Table 2.9: Confusion matrices

Mushroom

	1	2
1	4208	0
2	0	3916

Iris

	1	2	3
1	47	2	0
2	2	48	1
3	1	0	40

Heart

	1	2	3	4	5
1	142	22	9	6	2
2	17	23	8	9	5
3	3	2	12	4	2
4	0	7	5	10	4
5	2	1	2	6	0

The values denote how many transactions with class *column* are classified as class *row*

the accuracies from different sources, we cannot conclude that a particular classifier is better on a certain dataset if the difference is not larger than 2-3%.

The Krimp classifier scores 6 wins, indicated in boldface, which is only beaten by C4.5 with 7 wins. Additionally, the achieved accuracy is close to the winning score in 5 cases, not so good for *Connect–4*, *Heart* and *Tic–tac–toe*, and average for the remaining 5 datasets.

Compared to Naïve Bayes, to which our method is closely related, we observe that the obtained scores are indeed quite similar. On average, however, Krimp achieves 2.4% higher accuracy, with 84.5% for Krimp and 82.1% for Naïve Bayes. The average accuracy for SVM is 83.8%, while C4.5 outperforms all with 86.3%. We also looked at the performance of FOIL, PRM and CPAR [107, 136] reported in [34]. These classifiers perform sub-par in comparison to those in Table 2.8. A comparison to LB and/or LB-chi2 [93] is problematic, as only few accuracies are available for the (large) datasets we use and for those that are available, the majority of the LB results is based on train/test, not 10-fold cross-validated.

To get more insight in the classification results, confusion matrices for 3 datasets are given in Table 2.9. The confusion matrix for *Heart* shows us why the Krimp classifier is unable to perform well on this dataset: it contains 4 very small classes. For such small databases, the size of the code table is dominant, precluding the discovery of the important frequent itemsets. This is obviously the case for some *Heart* classes. If we consider *Mushroom* and *Iris*, then the bigger the classes, the better the results. In other words, if the classes are big enough, the Krimp classifier performs very well.

We can zoom in even further to show how the classification principle works in practice. Figure 2.7 illustrates the effect of coding a transaction with the 'wrong' code. The rounded boxes in this figure visualise the itemsets that make up the cover of the transaction. Each of the itemsets is linked to its code by

the dashed line. The widths of the black and white encodings represent the actual computed code lengths. From this figure, it is clear that code tables can be used to both characterise and recognise data distributions.

Order

Next, we investigate the order heuristics of the KRIMP algorithm. Both the standard cover order and standard candidate order are rationally made choices, but choices nevertheless. Here, we consider a number of alternatives for either and evaluate the quality of possible combinations through compression ratios and classification accuracies. The outcome of these experiments are shown in Table 2.10. Before we cover these results, we discuss the orders. As before, $\downarrow$ indicates the property to be sorted descending, and $\uparrow$ ascending.

For the **Standard Cover Algorithm**, we experimented with the following orders of the coding set.

- **Standard Cover Order**:

$$|X| \downarrow \quad supp_{\mathcal{D}}(X) \downarrow \quad \text{lexicographically} \uparrow$$

- **Entry**:

$$\textit{index of } X \text{ in } \mathcal{F} \downarrow$$

- **Area ascending**:

$$|X| \times supp_{\mathcal{D}}(X) \uparrow \quad \textit{index of } X \text{ in } \mathcal{F} \downarrow$$

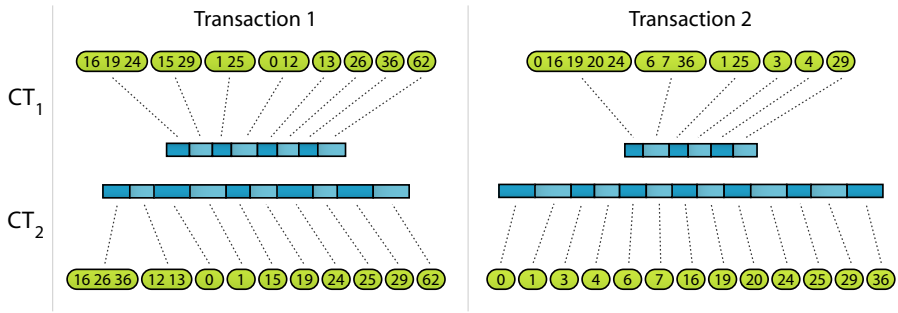


Figure 2.7: *Wine*; two transactions from class 1, $\mathcal{D}_1$, encoded by the code tables for class 1, CT_1 (top), and class 2, CT_2 (bottom)

We also consider the **Random** cover order, where new itemsets are entered at a random position in the code table. For all the above orders, we sort the singleton itemsets below itemsets of longer length. In Table 2.10 we refer to these orders, respectively, as Standard, Entry, Area and Random. We further experimented with a number of alternatives of the above, but the measured performance did not warrant their inclusion in the overall comparison.

As for the lineup in which the itemsets are considered by the Krimp algorithm, the candidate order, we experimented with the following options.

- **Standard Candidate Order**:

$$supp_{\mathcal{D}}(X) \downarrow \quad |X| \downarrow \quad \text{lexicographically} \uparrow$$

- **Standard, but length ascending**:

$$supp_{\mathcal{D}}(X) \downarrow \quad |X| \uparrow \quad \text{lexicographically} \uparrow$$

- **Length ascending, support descending**:

$$|X| \uparrow \quad supp_{\mathcal{D}}(X) \downarrow \quad \text{lexicographically} \uparrow$$

- **Area descending, support descending**:

$$|X| \times supp_{\mathcal{D}}(X) \downarrow \quad supp_{\mathcal{D}}(X) \downarrow \quad \text{lexicographically} \uparrow$$

A **Random** candidate order is also considered, which is simply a random permutation of the candidate set. In Table 2.10 we refer to these orders as, respectively, Standard, Standard', Length, Area and Random. Again, we considered a number of variants of the above, for none of which the performance was good enough to be included here.

For all 20 combinations of the above cover and candidate orders we ran compression experiments on 16 datasets and classification experiments for 11 datasets. As candidate itemsets, we ran experiments with both the complete and closed frequent itemset collections. Due to the amount of experiments required by this evaluation we only used single samples for the random orders. Classification scores are 10-fold cross-validated, as usual. Per dataset, the score of the best performing pairing (absolute or relative) was chosen. The details on which datasets and what *minsup* thresholds were used are displayed in Table 2.11.

The results of these experiments are depicted in Table 2.10. Shown are, per combination of orders, the total compression ratio $L\%$ and the classification accuracy, both of which are averaged over all databases and both all and closed frequent itemsets as candidate sets.

Between the orders, we measure considerable differences in compression ratio, up to 15%. For the candidate orders, the standard cover order is the best choice. The difference between the two variants Standard and Standard', is negligible, while the other options perform significantly worse for cover orders Standard and Entry. The same can be said for classification. We note that, although intuitively it seems a good choice, all variants of the area-descending order we further tested perform equally to each other, but sub-par when compared to the other orders.

For the standard cover algorithm, order-of-entry performs best, with the standard cover order second at a slight margin of half a percent. Covering in order of area shows outright bad performance, loosing even to random. As order-of-entry shows the best compression ratios, it is preferred from a MDL point of view. However, the standard order has the practical benefit of being (much) faster in practice. As it does not always insert new elements at the 'top' of the code table, partially covered transactions can be cached, speeding up the cover process significantly. The differences between the two, both in terms of compression and classification accuracies, are small, warranting the choice of the 'suboptimal' standard cover order for the practical reason of speed.

The scores for the random orders show that the greedy covering and MDL-based selection are most important for attaining good compression ratios. With

Table 2.10: Evaluation of candidate and cover orders

	Cover order							
	Standard		Entry		Area		Random	
$\mathcal{F} \downarrow$	$L\%$	*acc.*	$L\%$	*acc.*	$L\%$	*acc.*	$L\%$	*acc.*
Standard	44.2	88.6	43.7	88.7	51.1	88.1	49.5	88.1
Standard'	44.2	88.5	43.7	88.8	51.6	88.1	49.5	88.3
Length	45.2	88.0	43.9	87.9	55.4	87.1	49.1	78.8
Area	48.6	88.0	64.5	88.4	64.4	88.1	65.0	88.0
Random	49.4	86.8	51.2	87.0	57.5	86.8	50.8	87.0

Results for 20 combinations of candidate and cover orders for Krimp with post-acceptance pruning. Shown are average relative Krimp compression, $L\%$, and average classification accuracy (%) on a number of datasets. Results for compression and classification are averaged over 16 resp. 11 datasets. Table 2.11 shows which datasets were used and at what *minsup* thresholds the candidate sets, $\mathcal{F}$, were mined for these experiments.

either the candidate and/or cover order being random, Krimp still typically attains ratios of 50% and average accuracies are far above the baseline of 47.7%. This is due to the redundancy in the candidate sets and the cover order being fixed, even when the insertion position for a candidate is random. This allows the selection process to still pick sets of high-quality itemsets, albeit sub-optimal.

Table 2.11: Datasets and settings for the candidate and cover order experiments

	minsup		*Used for*	
Dataset	all	closed	Compression	Classification
Adult	20	1	✓	✓
Anneal	1	1	✓	✓
Breast	1	1	✓	✓
Chess (kr–k)	1	1	✓	
DNA amplification	10	1	✓	
Ionosphere	50	1	✓	✓
Iris	1	1	✓	✓
Led7	1	1	✓	✓
Letter recognition	50	20	✓	
Mammals	545	545	✓	
Mushroom		1	✓	✓
Nursery	1	1	✓	
Pen digits	20	1	✓	✓
Pima	1	1	✓	✓
Waveform	50	5	✓	✓
Wine	1	1	✓	✓

Details for which datasets and which settings were used for the experiments for Table 2.10. Per dataset, shown are the *minsup* thresholds at which candidate sets, $\mathcal{F}$, were mined for all and closed frequent itemsets. Ticks indicate which datasets were used for what experiments. In total, tens of thousands of individual Krimp compression runs were required for these order experiments.

2.8 Discussion

The experimental evaluation shows that KRIMP provides a practical solution to the well-known explosion in pattern mining. It reduces the highly redundant frequent itemset collections with many orders of magnitude to sets of only hundreds of high-quality itemsets. High compression ratios indicate that these itemsets are characteristic for the data and non-redundant in-between. Swap randomisation experiments show that the selections model relevant structure, and exclusion of itemsets shows that the method is stable with regard to noise. The quality of the itemsets is independently validated through classification, for which we introduced theory to classify by code-table based compression. While the patterns are chosen to compress well, the KRIMP classifier performs on par with state-of-the-art classifiers.

KRIMP is a heuristic algorithm, as is usual with MDL: the search space is by far too large to consider fully, especially since it is unstructured. The empirical evaluation of the choices made in the design of the algorithm show that the standard candidate order is the best, both from a compression and a classification perspective. The standard order in which itemsets are considered for covering a transaction is near-optimal; the order-of-entry approach, where new itemsets are used maximally, achieves slightly better compression ratios and classification accuracies. However, the standard order allows for efficient caching, speeding up the cover process considerably while hardly giving in on quality. Post-acceptance pruning is shown to improve the results: fewer itemsets are selected, providing better compression ratios and higher classification accuracies. Although pruning requires itemsets in the code table to be reconsidered regularly, its net result is a speed-up as code tables are kept smaller and the cover process thus needs to consider fewer itemsets to cover a transaction.

The timings reported in this study show that compression is not only a good, but also a realistic approach, even for large databases and huge candidate collections; the single-threaded implementation already considers up to hundreds of thousands of itemsets per second. While highly efficient frequent itemset miners were used, we observed that mining the candidates sometimes takes (much) longer than the actual KRIMP selection. Also, the algorithm can be easily parallelised, both in terms of covering parts of the database and of checking the candidate itemsets. The implementation[6] we used for the experiments in this chapter does the latter, as the performance of the former deteriorates rapidly for candidate itemsets with low support.

In general, the larger the candidate set, the better the compression ratio. The total compressed size decreases continuously, even for low *minsup* values, i.e. it never converges. Hence, $\mathcal{F}$ should be mined at a *minsup* threshold as low

[6] Our implementation of KRIMP is freely available for research purposes from `http://www.cs.uu.nl/groups/ADA/krimp/`

as possible. Given a suited frequent itemset miner, experiments could be done iteratively, continuing from the so-far optimal code table and corresponding previous *minsup*. For many datasets, it is computationally feasible to set $minsup = 1$.

When mining all frequent itemsets for a low *minsup* is infeasible, using closed frequent itemsets as candidate set is a good alternative instead. For most datasets, results obtained with closed are almost as good as with all frequent itemsets, while for some datasets this makes the candidate set much smaller and thus computationally attractive.

Krimp can be regarded a parameter-free algorithm and used as such in practice. The candidate set is a parameter, but since larger candidate sets give better results this can always be set to all frequent itemsets with $minsup = 1$. Only when this default candidate set turns out to be too large to handle for the available implementation and hardware, this needs to be tuned. Additionally, using post-acceptance pruning always improves the results and even results in a speed-up in computation time, so there is no reason not to use this.

Although code tables are made to just compress well, it turns out they can easily be used for classification. Because other classifiers have been designed with classification in mind, we expected these to outperform the Krimp classifier. We have shown this is not the case: Krimp performs on par with the best classifiers available. We draw two conclusions from this observation. Firstly, Krimp selects itemsets that are very characteristic for the data. Secondly, the compression-based classification scheme works very well.

While this chapter covers a large body of work done, there remains plenty of future work left to do. For example, Krimp could be further improved by directly generating candidate itemsets from the data and its current cover. Or, all frequent itemsets could be generated on-the-fly from a closed candidate set. Both extensions would address the problems that occur with extremely large candidate sets, i.e. crashing itemset miners and IO being the bottleneck instead of CPU time.

2.9 Conclusions

In this chapter we have shown how MDL gives a dramatic reduction in the number of frequent itemsets that one needs to consider. For twenty-seven data sets, the reductions reached by the Krimp algorithm ranges up to seven orders of magnitude; only hundreds of itemsets are required to succinctly describe the data. The algorithm shows a high stability w.r.t. different candidate sets. It is parameter-free for all practical purposes; for the best result, use as large as possible candidate sets and enable pruning.

Moreover, by designing a simple classifier we have shown that Krimp picks itemsets that matter. This is all the more telling since the selection of code

table elements does not take predictions into account. The small sets that are selected characterise the database accurately, as is also indicated by small compressed sizes and swap randomisation experiments.

In this chapter, we verified the heuristic choices we made for the KRIMP algorithm in [113]. We extensively evaluated different possible orders for both the candidate set and code table. The outcome is that the standard orders are very good: no combination of orders was found that performs significantly better, while the standard orders offer good opportunities for optimisation.

Because we set the frequent pattern explosion, the original problem, in a wide context but discussed only frequent itemsets, the reader might wonder: does this also work for other types of patterns? The answer is affirmative, in [11] we have shown that our MDL-based approach also works for pattern-types such as *frequent episodes* for sequence data and *frequent subgraphs* for graph data. In [74, 75], we extended the approach to multi-relational databases, i.e. to select patterns over multiple tables. Also, the LESS algorithm [60] (see also Section 2.6) introduces an extension of the encoding such that it can be used to select more generic patterns, e.g. low-entropy sets.

Like detailed in [42], there are many data mining tasks for which compression, and thus the foundations presented in this chapter, can be used. E.g. we have independently shown that compression (or, more specifically, KRIMP) can be successfully employed for characterising differences [124], generating data and preserving privacy [125], and identifying the components in a database [79]. The next chapters will detail these and further approaches.

CHAPTER 3

Characterising the Difference

Characterising the differences between two databases is an often occurring problem in Data Mining. Detection of change over time is a prime example, comparing databases from two branches is another one. The key problem is to discover the patterns that describe the difference. Emerging patterns provide only a partial answer to this question.

In the previous chapter, we showed that the data distribution can be captured in a pattern-based model using compression. Here, we extend this approach to define a generic dissimilarity measure on databases. Moreover, we show that this approach can identify those patterns that characterise the differences between two distributions. Experimental results show that our method provides a well-founded way to independently measure database dissimilarity that allows for thorough inspection of the actual differences. This illustrates the use of our approach in real world data mining.[1]

[1] This work was originally published as [124]:
Vreeken, J., van Leeuwen, M., Siebes, A. (2007). Characterising the Difference. In *Proceedings of the KDD'07.* pages 765-774. ACM.

3.1 Introduction

Comparing databases to find and explain differences is a frequent task in many organisations. The two databases can, e.g. be from different branches of the same organisations, such as sales records from different stores of a chain or the 'same' database at different points in time. In the first case, the goal of the analysis could be to understand why one store has a much higher turnover than the other. In the second case, the goal of the analysis could be to detect changes or drift over time.

The problem of this kind of 'difference detection' has received ample attention, both in the database and in the data mining community. In the database community, OLAP [32] is the prime example. Using roll-up and drill-down operations, a user can (manually) investigate, e.g. the difference in sales between the two stores. Emerging pattern mining [38] is a good example from the data mining community. It discovers those patterns whose support increase significantly from one database to the other.

Emerging patterns, though, are often redundant, giving many similar patterns. Also, the growth rate that determines the minimal increase in support has a large impact on the number of resulting patterns. Lower growth rates give large amounts of patterns, of which only some are useful. To discover only 'interesting' differences would require the data miner to test with multiple growth rate settings and, manually, trace what setting gives the most useful results and filter those from the complete set of emerging patterns.

In this chapter we propose a new approach to 'difference detection' that identifies those patterns that characterise the differences between the two databases. In fact, the approach just as easily identifies the characteristic differences between multiple databases. The approach extends our earlier work employing Minimum Description Length (MDL) for frequent pattern mining. As in most of the chapters of this thesis, we here restrict ourselves to frequent itemset mining, although the methodology easily extends to other kinds of patterns and data types, see [11].

In the previous chapter we attacked the well-known frequent itemset explosion at low support thresholds using MDL. The MDL philosophy is that the selected subset gives the best approximation of the underlying data distribution. We independently verified this claim by using the compression schemes for classification. Say, we have two classes, C_1 and C_2. Select the MDL-best set F_1 of frequent itemsets for the sub-database for class C_1 and F_2 for class C_2. As explained above, this gives us two compression algorithms, configured by code table CT_1 based on F_1 and code table CT_2 based on F_2. A new, unseen, example t can now be compressed by both CT_1 and CT_2. In the chapter we argued that the Bayes optimal choice is to assign t to the class whose compressor compresses t best. This simple classification algorithm scores on-par with state-of-the-art classification algorithms; please refer to Chapter 2 for details.

The approach towards difference detection introduced in this chapter is again based on compression. First, we use compression to define a dissimilarity measure on databases. Then we introduce three ways to characterise the differences between two (dis)similar databases.

Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be the two databases, with transactions concerning the same sets of items, of which we need to analyse the differences. In Section 3.3, we first consider the difference in compressed length for the transactions in $\mathcal{D}_1$ when compressed by the MDL-compression schemes. The MDL-principle as well as our results in classification imply that the compression scheme induced from $\mathcal{D}_2$ should in general do worse than the scheme induced from $\mathcal{D}_1$. This is verified by some simple experiments.

Next, we aggregate these differences per transaction by summing over all transactions in $\mathcal{D}_1$ and normalising this sum by the optimal code length for $\mathcal{D}_1$. This aggregation measures how different a database is from $\mathcal{D}_1$. This is verified by experiments that show the correlation between this similarity measure and the confusion matrix of our classification algorithm briefly introduced above and in Section 3.2. Finally, this simple measure is turned into a dissimilarity measure for any pair of databases by taking the maximum of how different $\mathcal{D}_1$ is from $\mathcal{D}_2$ and vice versa. Again, the MDL-principle implies that this is a dissimilarity measure. Experiments verify this claim by showing the correlation between this dissimilarity measure and the accuracy of our classification algorithm.

The result of Section 3.3 is a dissimilarity measure for a pair of databases, based on code tables. If the dissimilarity is small, the two databases are more or less the same and a further analysis of the differences will not show anything interesting. The topic of Section 3.4 is on how to proceed if the dissimilarity is large. In that section, we introduce three ways to characterise these differences. The first approach focuses on the usage-patterns of the code table elements, while the second focuses on how (sets of) transactions are compressed by the two different schemes. The third and last approach focuses on differences in the code tables themselves. All three approaches highlight complementary, characteristic, differences between the two databases.

In Section 3.5 we discuss related work and describe the differences with our work. We round up with conclusions and future research in Section 3.6.

3.2 Preliminaries

Foundation of all data discussed in this chapter is a set of items $\mathcal{I}$, e.g. the items for sale in a shop. A transaction $t \in \mathcal{P}(\mathcal{I})$ is a set of items, e.g. representing the items a client bought at that store. A database $\mathcal{D}$ over $\mathcal{I}$ is a bag of transactions, e.g. the different sale transactions on a given day. An itemset $X \subseteq \mathcal{I}$ occurs in a transaction $t \in \mathcal{D}$ iff $X \subseteq t$. The support of X in $\mathcal{D}$ is the

number of transactions in the database in which X occurs.

In this work we build on the Krimp compressor and classifier introduced in Chapter 2. For details, please refer to that chapter, or see [123].

During the classification experiments, we made some interesting observations in the distributions of the code lengths (not shown previously). Figure 3.1 shows the encoded lengths for transactions of a single class, encoded by code tables constructed for each of the three classes. Not only gives the code table constructed for these transactions shorter encodings, the standard deviation is also much smaller (compare the histogram on the left to the other two). This means that a better fit of the code table to the distribution of the compressed data results in a smaller standard deviation.

Experimental setup

Although a lot of time series data is being gathered for analysis, no good benchmark datasets with this type of data currently exist. We therefore decided to use a selection from the UCI repository [33], which has been commonly used for emerging patterns [38] and related topics before.

As these are all datasets containing multiple classes, we look at the differences between classes. Hence, we split each dataset on classlabel C and remove this label from each transaction, resulting in a database $\mathcal{D}_i$ per class C_i. A code table induced from $\mathcal{D}_i$ using Krimp is written as CT_i.

For many steps in Sections 3.3 and 3.4, we show results obtained with the datasets *Heart* and *Wine* because of their properties: they are interesting because they consist of more than 2 classes, but don't have too many classes. Please note this selection is only for purpose of presentation; results we obtained with other (larger) datasets are similar. In fact, Krimp is better at approximating data distributions of larger databases, providing more reliable results.

Characteristics of all datasets used are summarised in Table 3.8, together with the minimum support levels we use for mining the frequent itemsets that function as candidates for Krimp. All experiments in this chapter are done with all frequent itemsets.

3.3 Database Dissimilarity

In this section, we introduce a dissimilarity measure for transaction databases. This measure indicates whether or not it is worthwhile to analyse the differences between two such databases. If the dissimilarity is low, the differences between the two databases are small. If the measure is high, it is worthwhile to investigate the differences. Rather than defining the similarity measure upfront followed by a discussion and illustration of its properties, we 'develop' the

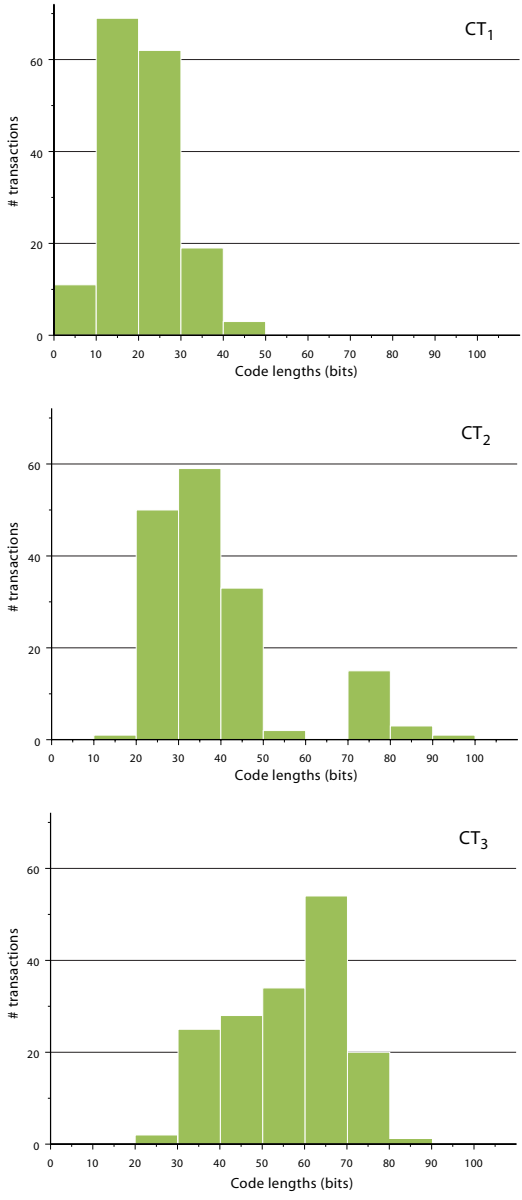


Figure 3.1: *Heart*; encoded transaction lengths for all transactions belonging to one class ($\mathcal{D}_1$), encoded with the code tables constructed for each of the three classes (top to bottom: CT_1, CT_2, CT_3).

measure in a few steps as that allows us to discuss the intuition that underlies the definition far easier.

Differences in code lengths

The MDL principle implies that the optimal compressor induced from a database $\mathcal{D}_1$ will generally provide shorter encodings for its transactions than the optimal compressor induced from another database $\mathcal{D}_2$. Our earlier experiments on classification verify that this is also true for the code table compressors Krimp discovers heuristically; see Section 3.2.

More in particular, denote by H_i the optimal compressor induced from database $\mathcal{D}_i$ and let t be a transaction in $\mathcal{D}_1$. Then, the MDL principle implies that:

$$|H_1(t) - H_2(t)|$$

- is small if t is equally likely generated by the underlying distributions of $\mathcal{D}_1$ and $\mathcal{D}_2$
- is large if t is more likely generated by the distribution underlying one database than that it is generated by the distribution underlying the other.

In fact the MDL principle implies that if the code length differences are large (the second case), then on average the smallest code length will be $H_1(t)$. Our classification results suggest that something similar should hold for the code table compressors discovered by Krimp. In other words, we expect that

$$CT_2(t) - CT_1(t)$$

measures how characteristic t is for $\mathcal{D}_1$. That is, we expect that this difference is most often positive and large for those transactions that are characteristic for $\mathcal{D}_1$.

In Figures 3.2 and 3.3 code length differences are shown for two datasets, respectively for transactions of the $Wine_1$ and $Heart_1$ databases. As we expected, virtually all code length differences are positive. This means that in practice the native code table does indeed provide the shortest encoding.

In the case of the $Wine_1$ database depicted in Figure 3.2, we see a whopping average difference of 45bits per transaction. The shapes of the two histograms also show a nice clustering of the differences between the encoded lengths. No negative differences occur, each single transaction is compressed better by its native code table. This confirms that MDL creates code tables that are truly specific for the data.

We see the same general effect with $Heart_1$ in Figure 3.3, as again the peaks of the distribution lay within safe distance from the origin. From the histograms

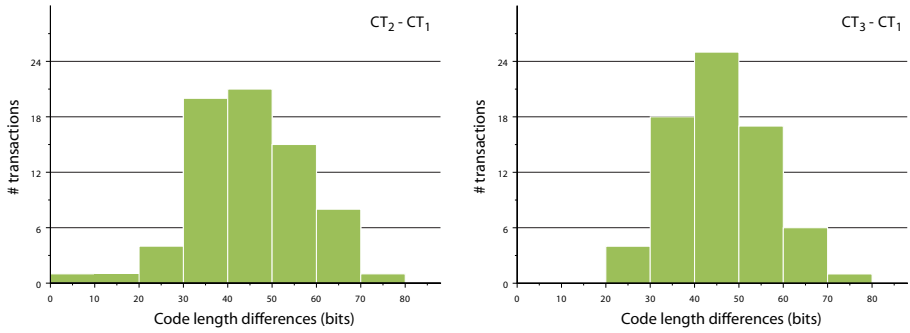


Figure 3.2: *Wine*; code length difference histograms for transactions in $\mathcal{D}_1$: encoded length differences between CT_2 and CT_1 (left) and between CT_3 and CT_1 (right).

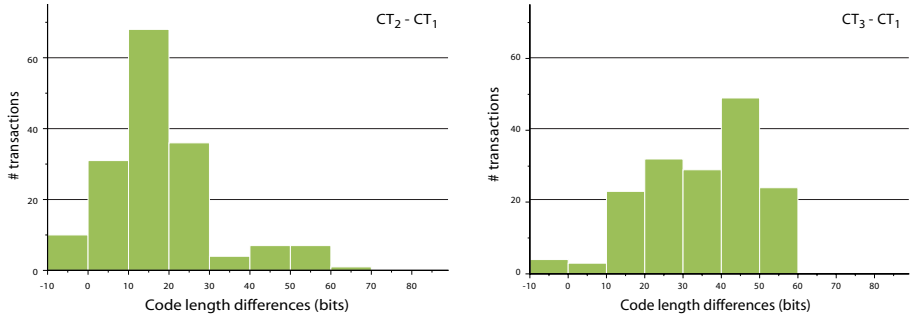


Figure 3.3: *Heart*; code length difference histograms for transactions in $\mathcal{D}_1$: encoded length differences between CT_2 and CT_1 (left) and between CT_3 and CT_1 (right).

there is little doubt that code tables CT_2 and CT_3 are encoding data from a different distribution than they have been induced from. More importantly, comparing these diagrams unambiguously shows that it is possible to use the differences in encoded lengths to measure the amount of change between data. For example, as the differences on the left histogram are clearly smaller than in the situation on the right, this seems to imply that *Heart* classes 1 and 2 are more alike than classes 1 and 3. How to investigate this hypothesis further will be discussed in the next section. First we continue the development of our dissimilarity measure.

Aggregating code length differences

In the previous subsection we have seen that the histograms of code length differences give good insight in the differences between two databases. The next logical step towards the definition of a dissimilarity measure is to aggregate these differences over the database. That is, to sum the individual code length differences over the complete database.

Straightforward aggregation, however, might give misleading results for two reasons:

- code length differences can be negative, so even if $\mathcal{D}_1$ and $\mathcal{D}_2$ are rather different, the aggregated total might be small.
- if $\mathcal{D}_1$ is a large database, the aggregated total might be large even if $\mathcal{D}_2$ is very similar to $\mathcal{D}_1$.

As already mentioned in the previous subsection, the MDL principle implies that for the MDL-optimal compressors H_1 and H_2, the expected average value of $H_2(t) - H_1(t)$ is positive. In other words, negative code length differences will be relatively rare and won't unduly influence the aggregated sum.

Our results in classification and, more importantly, the results of the previous subsection indicate that the same observation holds for the code table compressors CT_1 and CT_2 induced by Krimp. Clearly, only experiments can verify this claim.

The second problem indicated above is, however, already a problem for the MDL-optimal compressors H_1 and H_2. For, the expected value of the sum of the code length differences is simply the number of transactions times the expected average code length difference. Since the latter number is positive according to the MDL principle, the expected value of the sum depends linearly on the number of transactions on the database.

Clearly, the 'native' encoded size of the database, $CT_1(\mathcal{D}_1)$, also depends on the size of the database. Therefore, we choose to counterbalance this problem by dividing the sum of code length differences by this size. Doing this, we end up with the Aggregated Code Length Difference:

$$ACLD(\mathcal{D}_1, CT_2) = \frac{CT_2(\mathcal{D}_1) - CT_1(\mathcal{D}_1)}{CT_1(\mathcal{D}_1)}$$

Note that ACLD is an asymmetric measure: it measures how different $\mathcal{D}_2$ is from $\mathcal{D}_1$, not vice versa! While one would expect both to be in the same ballpark, this is by no means given. The asymmetry is further addressed in the next subsection. To clearly indicate the asymmetry, the parameters are asymmetric: the first parameter is a database, while the second is a code table.

Given this definition, we can now verify experimentally whether it works or not. That is, do greater dissimilarities imply larger differences and vice versa?

Table 3.1: *Heart*; aggregated code length differences.

	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$	$\mathcal{D}_5$
CT_1	0.00	0.36	0.71	0.88	1.58
CT_2	0.85	0.00	0.60	0.65	1.03
CT_3	1.65	0.78	0.00	0.60	1.25
CT_4	1.85	0.65	0.61	0.00	1.09
CT_5	2.18	1.07	0.72	0.87	0.00

Table 3.2: *Wine*; aggregated code length differences.

	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$
CT_1	0.00	1.27	1.32
CT_2	1.13	0.00	1.73
CT_3	1.14	1.68	0.00

In Table 3.1 we read the aggregated code length differences for all possible combinations of code tables and class databases for the *Heart* dataset. It is immediately clear there are distinct differences between the class distributions, as measurements of 1.00 imply code lengths averaging twice as long (or, 100% more bits required) as that of the actual class. We also notice that while the data distributions of databases 1 and 5 are quite distinct, the lower measurements between the other three classes indicate that their distributions are more alike.

For the *Wine* database the class distributions are more adrift than those in the Heart database, for all cross-compressions result in encodings more than twice as long as the native ones. This is completely in line with what we have seen before in Figure 3.2, in which we showed there is no uncertainty in keeping transactions of the *Wine* databases apart based on encoded lengths.

If this technique truly quantifies the likeliness of the distributions belonging to some data, intuition tells us there has to be a close relation with the classification quality based on encoded transaction lengths. We can easily check this by comparing the aggregated code length differences with the confusion matrices for these databases. We therefore ran 10-fold cross validated classification experiments for these databases, as we did in the previous chapter.

The confusion matrix for the *Heart* database, in Table 3.3, clearly shows the intuition to be correct, as the number of misclassified instances drops completely according to ACLD. While 24 transactions of class 2 are misclassified as belonging to class 1, we see in Table 3.1 that these two classes are measured as

Table 3.3: *Heart*: classification confusion matrix.

Classified as	*Class* 1	2	3	4	5
1	137	24	9	6	3
2	12	11	11	7	5
3	6	8	7	8	1
4	8	10	7	9	4
5	1	2	2	5	0

Table 3.4: *Wine*: classification confusion matrix.

Classified as	*Class* 1	2	3
1	65	3	6
2	5	55	0
3	1	1	42

rather similar. In fact, if we sort the measurements in Table 3.1 per class, we find the same order as when we sort Table 3.3 on the number of misclassifications. The measured difference thus directly relates to the ability to distinguish classes.

In Table 3.4 we see the same pattern with the *Wine* database as with the *Heart* database before: the lowest dissimilarities relate to the most misclassifications. We also observe that while analysis of individual code length differences, like Figure 3.2, suggests there should be no confusion in classification, a number of transactions are misclassified. These can be tracked back as being artefacts of the 10-fold cross validation on a small database.

The database dissimilarity measure

The experiments presented above verified that the aggregated differences of database encodings provide a reliable means to measure the similarity of one database to another. To make it into a true dissimilarity measure, we would like it to be symmetric. Since the measure should indicate whether or not we should investigate the differences between two databases, we do this by taking the maximum value of two Aggregated Code Length Differences:

$$\max\{ACLD(\mathcal{D}_1, CT_2), ACLD(\mathcal{D}_2, CT_1)\}$$

This can easily be rewritten in terms of compressed database sizes, without using the ACLD function.

Definition 6. *For all databases x and y, define the code table dissimilarity measure DS between x and y as*

$$DS(x,y) = \max\left\{ \frac{CT_y(\mathcal{D}_x) - CT_x(\mathcal{D}_x)}{CT_x(\mathcal{D}_x)}, \frac{CT_x(\mathcal{D}_y) - CT_y(\mathcal{D}_y)}{CT_y(\mathcal{D}_y)} \right\}.$$

The databases are deemed very similar (possibly identical) iff the score is 0, higher scores indicate higher levels of dissimilarity. Although at first glance this method comes close to being a distance metric for databases, this is not entirely the case. A distance metric D must be a function with non-negative real values defined on the Cartesian product $K \times K$ of a set K. Furthermore, it must obey the following requirements for every $k, l, m \in K$:

1. $D(k,l) = 0$ iff $k = l$ (identity)
2. $D(k,l) = D(l,k)$ (symmetry)
3. $D(k,l) + D(l,m) \geq D(k,m)$ (triangle inequality)

For the MDL optimal compressors, we can prove that DS will be positive. For our code table compressors, we can not. However, the experiments in the previous two subsections as well as those in this one indicate that DS is unlikely to be negative. As we can not even guarantee that DS is always positive, we can certainly not prove the identity axiom. The second axiom, the symmetry axiom holds, of course, by definition. For the triangle inequality axiom we again have no proof. However, in the experiments reported on this subsection the axioms hold. In other words, for all practical purposes our measure acts as a distance measure. However, to clearly indicate that our measure is not a proven distance metric we call it a dissimilarity measure.

The dissimilarity measurements for the *Heart*, *Nursery* and *Wine* database are given in respectively Tables 3.5, 3.6 and 3.7. One of the most striking observations is that many of the measurements are greater than 1.0, meaning that the cross-compressed databases are more than twice as large as the natively-compressed databases. The differences between the $Nursery_3$ and $Nursery_5$ datasets are such that a dissimilarity measurement of 10.12 is the result: a difference of a factor 11 of the average encoded length of a transaction.

In Table 3.8 a summary of datasets, their characteristics and dissimilarity results is given. For each dataset, the lowest and the highest observed dissimilarity is listed. A full results overview would obviously require too much space; datasets with many classes have squared as many database pairs of which the dissimilarity can be measured.

Table 3.5: *Heart*: dissimilarity.

	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$
$\mathcal{D}_2$	0.85			
$\mathcal{D}_3$	1.65	0.78		
$\mathcal{D}_4$	1.85	0.65	0.61	
$\mathcal{D}_5$	2.18	1.07	1.25	1.09

Table 3.6: *Nursery*: dissimilarity.

	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$
$\mathcal{D}_2$	2.62			
$\mathcal{D}_3$	2.83	2.04		
$\mathcal{D}_4$	3.10	1.91	4.05	
$\mathcal{D}_5$	7.38	1.26	10.12	1.54

Table 3.7: *Wine*: dissimilarity.

	$\mathcal{D}_1$	$\mathcal{D}_2$
$\mathcal{D}_1$	1.27	
$\mathcal{D}_2$	1.32	1.73

Overall, we see that the dissimilarities between the classes of the UCI datasets vary quite a bit. Some datasets seem to have very little difference between classes (*Connect–4*, *Adult*, *Tic–tac–toe*), others contain rather large dissimilarity (*Mushroom*, *Iris*, *Led7*).

Another interesting comparison is between the dissimilarities and the classification results also reported in that table, taken from Chapter 2. There is a clear correlation between the two. The larger the dissimilarity, the better the classification results. This pattern is less clear for datasets containing small classes, which is caused by the fact that MDL doesn't work well for small data sets.

This observation is interesting because classification errors are made on individual transactions, whereas DS is an aggregated measure. In other words, the observation verifies that this aggregated measure reflects what happens at the level of individual transactions. This is exactly the property our dissimilarity measure should hold.

Table 3.8: Database characteristics.

			Krimp		Dissimilarity (DS)	
Dataset	$\lvert\mathcal{D}\rvert$	*#classes*	*minsup*	*Acc. (%)*	*Min*	*Max*
Adult	48842	2	20	84.6	0.60	0.60
Chess (kr–k)	28056	18	10	58.0	0.29	2.69
Connect–4	67557	3	50	69.9	0.18	0.28
Heart	303	5	1	52.5	0.61	2.18
Iris	150	3	1	96.0	2.06	13.00
Led7	3200	10	1	75.3	1.27	11.29
Letter	20000	26	50	68.1	0.43	2.83
Mushroom	8124	2	50	100	8.24	8.24
Nursery	12960	5	1	92.4	1.26	10.12
PenDigits	10992	10	20	88.6	1.33	4.43
Tic–tac–toe	958	2	1	87.1	0.62	0.62
Wine	178	3	1	97.7	1.27	1.73

Candidate minsup and class dissimilarity measurements for a range of UCI datasets. As candidates, all frequent itemsets were used up to the given minimum support level.

3.4 Characterising Differences

The first benefit of our dissimilarity measure is that it quantifies the difference between databases, the second advantage is the ability to characterise those differences.

There are three methods available for difference analysis, which zoom in to separate levels of difference between the distributions. First, we can compare the code table covers of the databases. This directly informs us which patterns that are important in one database are either over or under-expressed in another database. The second approach is to zoom in on how specific transactions are covered by the different code tables. This reveals in detail where differences are identified by the code tables. Thirdly, we can extract knowledge about the specific differences and similarities between the distributions from the code tables.

Comparing database covers

The most straightforward, but rather informative method for difference analysis is the direct comparison of database covers. Such evaluation immediately identifies which patterns are over and under-expressed, showing us the charac-

teristics of the differences in structure between the two databases.

To run this analysis, we first use Krimp to obtain a code table for database $\mathcal{D}_2$ and use it to cover database $\mathcal{D}_1$. Because the itemsets and their frequencies in the code table capture the data distribution of database $\mathcal{D}_2$, the frequencies found by covering database $\mathcal{D}_1$ are expected to be different if the two databases are different.

Identification of these differences is done by finding those patterns in the code table that have a large shift in frequency between the two database covers. The same process can be applied vice versa for even better insight of the differences.

If the distribution is really different, we would expect to see a dramatic increase in use of the singletons caused by a decrease in use of the larger, more specific, sets. Slighter differences will lead to more specific shifts in patterns usage, with less of a shift towards singleton usage.

An example visualisation can be seen in Figure 3.4. A code table for *Wine* $\mathcal{D}_1$ has been constructed and used to cover all three databases. A quick glance shows that our hypothesis on the use of singletons is correct: $\mathcal{D}_1$ is covered by quite some sets of 2 or more items, but both $\mathcal{D}_2$ and $\mathcal{D}_3$ are covered largely by singletons.

Of special interest is the contrast in peaks between the plots, indicating (strong) shifts in pattern usage. A rather strong difference in pattern usage is visible for the lower indexes in the code table, corresponding to the longest, most specific, patterns. However, in this figure the high peaks are also indicative; we marked the peaks of an interesting case A1 and A2. These peaks are at exactly the same code table element, meaning that this pattern is used quite often in the covers of both $\mathcal{D}_1$ and $\mathcal{D}_2$. Note that it is not used at all in the cover of $\mathcal{D}_3$; hence this pattern could really give us a clue as to what differentiates $\mathcal{D}_1$ and $\mathcal{D}_2$ from $\mathcal{D}_3$. Another interesting peak is the one indicated with B: although it is also applied in the other covers, this pattern is clearly used much more often to cover $\mathcal{D}_3$.

Comparing transaction covers

A second approach for difference characterisation zooms in on individual database rows, and is thus especially useful when you are interested in specific transactions: why does a certain transaction belong to one database and not to another? Again, we use our code tables to inspect this.

Suppose we have two databases and their respective code tables. After computing the individual code length differences (as described in the first subsection of 3.3), it is easy to pick out those transactions that fit well in one database and not in another. After selecting a transaction, we can cover it with both code tables separately and visualise which patterns are used for this. In general, it will be covered by longer and more frequent patterns if it belongs

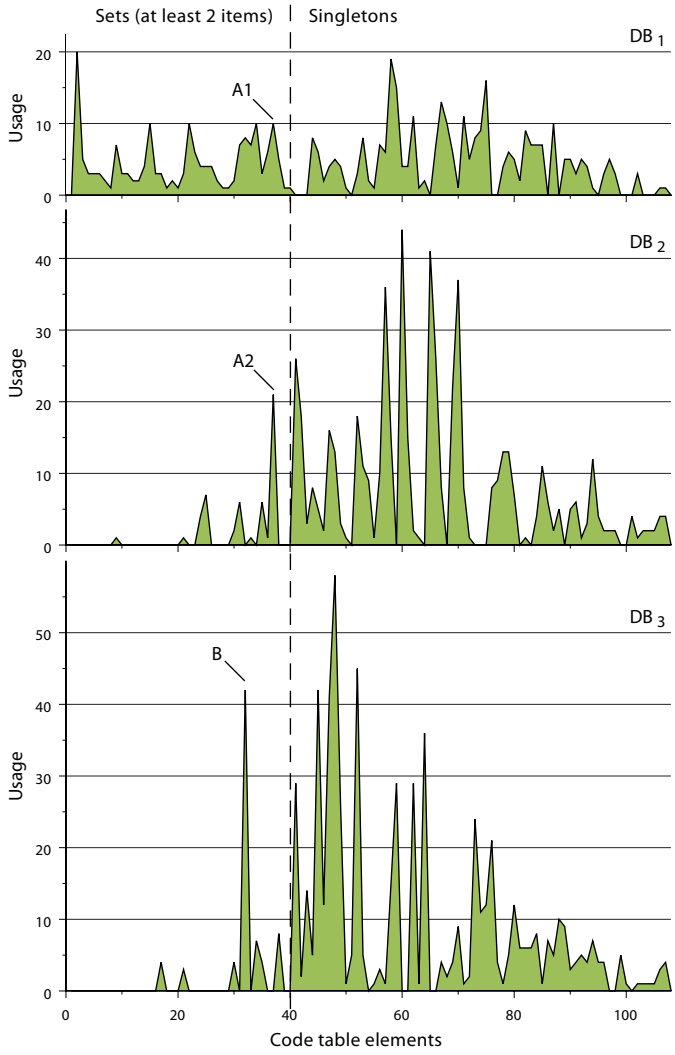


Figure 3.4: Comparing database covers. Each database of *Wine* has been covered by code table CT_1. Visualised is the absolute frequency for each of the code table elements.

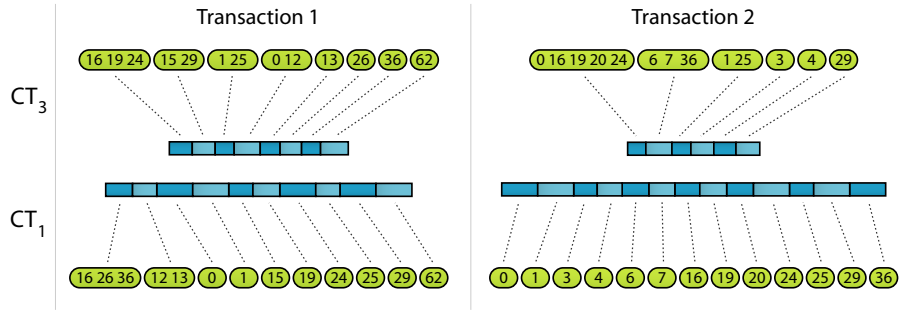


Figure 3.5: *Wine*; two transactions from $\mathcal{D}_3$ encoded by CT_3 (above) and CT_1 (below). The rounded boxes visualise the itemsets making up the cover of the transaction. Each of the itemsets is linked to its code by the dashed line. The widths of the black and white encodings represent the actual computed code lengths.

to a certain distribution than if it does not. Manual inspection of the individual transaction covers can reveal valuable knowledge.

As an example, have a look at another Wine example in Figure 3.5. The encodings by CT_1 and CT_3 of two sets from $\mathcal{D}_3$ are shown. Left and right show the same transactions, but they are covered by different itemsets (depicted by the rounded boxes). The itemsets are linked to their codes with the dashed lines. The width of each black or white code represents the length of that particular code; together the sum of these widths makes up the total length of the encoded transaction.

Looking at the upper transaction, we observe that both code tables cover the transaction with itemsets of intermediate length. However, CT_3 uses less and different patterns in its cover than CT_1. Moreover, the code lengths are obviously shorter, relating to high occurrence in the distribution from which CT_3 was induced. For further inspection of how important such patterns are, we zoom in to the pattern level in the third approach.

The covers of the second transaction give an even larger contrast than the previous one. The native code table covers the transaction with few and large patterns, while the other one uses only singletons. We may therefore conclude this transaction fits very well in its native distribution and very bad in the other. This also shows in the lengths of the encodings. Both examples show again that more singletons are used in a cover when data doesn't belong to a distribution.

Comparing code tables

The final third method for difference inspection focuses on the individual patterns in a data distribution. In order to pinpoint the differences in this respect, we have to directly compare the patterns in two code tables.

The weight and importance of patterns in the code tables cannot be compared naively, as for many of the patterns in a code table there does not have to be a direct equivalent in the other code table. However, the set of patterns in a code table can also be regarded as a database; in that fashion we can actually apply code tables to each other to find out what the alternative encoded length for each pattern is.

For each pattern in a code table we can compare its own encoded length to that of the alternative provided by the other code table, similarly to what we did for transactions in Section 3.3. Likewise, if the distributions are similar, we expect the encoded lengths to be comparable; even if the code tables use rather different patterns to encode it. In contrast, exactly those patterns for which the encoded lengths differ significantly mark the difference between the distributions.

We analysed the CT_2 and CT_3 code tables of the *Wine* dataset, and found further evidence for what puts these databases apart. The first peak in the topmost plot of Figure 3.4 corresponds to the pattern (0 16 19 20 24) from CT_3, which due to its high relative usage is encoded natively using only 1.4bits. From the same figure we already know this pattern is not used when covering the other databases; suggesting that perhaps neither this pattern, nor anything like it exists in the other code tables. Confirmation comes from an encoded length of 12.6bits that CT_2 assigns to this pattern; making it one of the patterns for which the encoded lengths differ most. As CT_2 cannot use any of the more frequently occurring code table patterns, it has to resort to low-frequency singleton encoding; arguably the least efficient method for encoding a pattern.

From the definition of the *Wine* database and analysis above we conclude that the main difference between the two classes lies in the combination of certain levels of malic acid (element 0) and a corresponding colour intensity (16). While CT_3 has a number of patterns that give these short encodings, CT_2 has virtually none: this pattern does not occur in this data distribution.

The above example evidently shows that the differences between the data distributions can be directly analysed, and that through comparison of the code table encodings key differences can be extracted. Similarities as well as the differences between distributions are pinpointed.

3.5 Related Work

Our dissimilarity measure DS is clearly related to the Normalised Information Distance (NID) and its compression-based instantiation NCD [80]. With

the NCD, general compressors like gzip are used as Kolmogorov complexity approximators and as such compressed sizes are used to measure distance between strings. As a generic distance, the NID has been successfully applied in a plethora of clustering tasks including small snippet based language and evolutionary tree rebuilding [31]. An adaptation was developed that has some practical data mining applications, among which compression-based anomaly detection [67].

However, the aim of the NID is different from ours: compression is only used as a means to quantify differences, not to qualitatively find what these differences are. In contrast, this is the main goal of our line of research. This is illustrated by the results of both the previous chapter and here. By considering transactional databases instead of individual strings and building code tables that can be analysed, Krimp provides a very natural way to gain insight in the differences between data distributions.

Our dissimilarity measure is also related to Emerging Patterns [38], although there are major differences. First of all, here we only consider patterns that are MDL-wise important with respect to the data distribution of a single database. The code table built allows to investigate other data sets (or transactions) from that particular database's perspective. This in contrast to Emerging Patterns, which are by definition identified as differences between pairs of databases, without regarding individual data distributions. Although we here focus on identifying differences, Krimp also reveals similarities between databases; arguably equally important when inspecting two databases. Also, when a large number n of databases is to be compared, constructing n code tables is computationally less intensive than mining n^2 sets of Emerging Patterns.

Secondly, Emerging Patterns are defined as patterns having a large difference in support (growth rate) between two databases. However, the frequencies used in our approach depend on the database cover, thus taking into account other patterns (and their order) in the code table. Through these dependencies, important changes in the structure of the data are enlarged and therefore easier to spot.

Thirdly, Krimp only selects small numbers of patterns. This allows for manual inspection at all stages, from data distribution approximation to difference detection and characterisation. Emerging Patterns suffer from the same combinatory explosion problem as frequent patterns: in order to capture all differences, a low (zero) growth rate has to be used, resulting in obstructively many patterns. Shorter descriptions have been defined for EPs, for example using borders [39], but as these only give a shorter description for the same set of patterns, manual inspection remains impossible. The set of Emerging Patterns cannot straightforwardly be reduced by Krimp. First, because it operates on individual databases, not on pairs. Second, to satisfy the MDL assumption,

the candidate pattern set should enable the algorithm to grasp full data distributions, not just differences. This is guaranteed by the frequent pattern set, but not by a set solely consisting of EPs.

3.6 Conclusions

In previous work, the MDL-principle and its implementation in the Krimp algorithm have proven themselves to be a reliable way for approximating the data distributions of databases. Here, we used this principle to develop a database dissimilarity measure with which characteristic differences between databases can be discovered.

Histograms for encoded transaction lengths, and the differences thereof, show differences between data distributions straightforwardly. From the MDL principle, code tables with a good fit on the distribution of some data provide shorter codes and smaller standard deviations than code tables less suited for the data at hand. The code length difference is shown to be a good indication to how well a transaction fits a distribution.

We show the informative quality of the aggregation of the code length differences. The measured likenesses show close relation to the confusion matrices of earlier classification experiments; the number of misclassified instances drops according to this measure.

We define a generic dissimilarity measure on databases as the maximum of two mirrored aggregated code length difference measurements; it is symmetric and well suited to detect and characterise the differences between two databases. While we cannot prove it to fulfil the distance metric axioms, we argued that these hold for all practical purposes.

A large advantage of our method is that it allows for thorough inspection of the actual differences between data distributions. Based on the dissimilarity, three methods for detailed inspection are proposed. The most detailed method zooms in onto and compares the patterns that describe the data distribution in the code tables. Individual transactions that do not fit the current distribution well can be identified. Further, it can be analysed why they do not fit that distribution well. Last but not least is the possibility to take a more global stance and pinpoint under or over expressed patterns in the respective databases.

Dissimilarity measures are key to many different data mining algorithms. For instance, it would be possible to apply our measure in a number of bioinformatics applications using these algorithms. For example, in those cases where classification appears to be hard; deeper insight in the causes of these problems might suggest promising research directions.

CHAPTER 4

Identifying the Components

Most, if not all, databases are mixtures of samples from different distributions. Transactional data is no exception. For the prototypical example, supermarket basket analysis, one also expects a mixture of different buying patterns. Households of retired people buy different collections of items than households with young children.

Models that take such underlying distributions into account are in general superior to those that do not. In this chapter we introduce two MDL-based algorithms that follow orthogonal approaches to identify the components in a transaction database. The first follows a model-based approach, while the second is data-driven. Both are parameter-free: the number of components and the components themselves are chosen such that the combined complexity of data and models is minimised. Further, neither prior knowledge on the distributions nor a distance metric on the data is required. Experiments with both methods show that highly characteristic components are identified.[1]

[1] This work was originally published as [79]:
Van Leeuwen, M., Vreeken, J. and Siebes, A. (2009) Identifying the Components. *Data Mining and Knowledge Discovery.* 19(2):173-192. Springer. (ECML PKDD'09 Special Issue) (Best student paper award)

4.1 Introduction

Most, if not all, databases are mixtures of samples from different distributions. In many cases, nothing is known about the source components of these mixtures and therefore many methods that induce models regard a database as sampled from a single data distribution. While this greatly simplifies matters, it has the disadvantage that it results in suboptimal models.

Models that do take into account that databases actually are sampled from mixtures of distributions are often superior to those that do not, independent of whether this is modelled explicitly or implicitly. A well-known example of explicit modelling is mixture modelling [120]. This statistical approach models data by finding combinations of several well-known distributions (e.g. Gaussian or Bernoulli) that are assumed to underlie the data.

Boosting algorithms [43] are a good example of implicit modelling of multiple underlying distributions. The principle is that a set of weak learners can together form a single strong learner. Each separate learner can adapt to a specific part of the data, implicitly allowing for modelling of multiple distributions.

Transaction databases are no different with regard to data distribution. As an illustrative example, consider supermarket basket analysis. One also expects a mixture of different buying patterns: different groups of people buy different collections of items, although overlap may exist. By extracting both the groups of people and their corresponding buying patterns, a company can learn a lot about its customers.

Part of this problem is addressed by clustering algorithms, as these group data points together, often based on some distance metric. However, since we do not know upfront what distinguishes the different groups, it is hard to define the appropriate distance metric. Furthermore, clustering algorithms such as k-means [88] only find the groups of people and do not give any insight in their corresponding buying behaviours. The only exception is bi-clustering [104], however this approach imposes strong restrictions on the possible clusters.

Frequent itemset mining, on the other hand, does give insight into what items customers tend to buy together, e.g. the (in)famous Beer and Nappies example. But, not all customers tend to buy Nappies with their Beer and standard frequent set mining does not distinguish groups of people. Clearly, knowing several groups that collectively form the client base as well as their buying patterns would provide more insight. So, the question is: can we find these groups and their buying behaviour? That is, we want to partition the database $\mathcal{D}$ in sub-databases $\mathcal{D}_1, \cdots, \mathcal{D}_k$ such that

- the buying behaviour of each $\mathcal{D}_i$ is different from all $\mathcal{D}_j$ (with $j \neq i$), and
- each $\mathcal{D}_i$ itself is homogeneous with regard to buying behaviour.

But, what does 'different buying behaviour' mean? It certainly does not mean that the different groups should buy completely different sets of items. Also, it does not mean that these groups cannot have frequent itemsets in common. Rather, it means that the characteristics of the sampled distributions are different. This may seem like a play of words, but it is not. Sampled distributions of transaction data can be characterised precisely through the use of code tables.

In Chapter 2 we introduced the Krimp algorithm. We use code tables and MDL to formalise our problem statement. That is: find a partitioning of the database such that the total compressed size of the components is minimised. This problem statement agrees with the compression perspective on data mining as recently positioned by Faloutsos et al. [42].

We propose two orthogonal methods to solve the problem at hand. The first method is based on the assumption that Krimp implicitly models all underlying distributions in a single code table. If this is the case, it should be possible to extract these and model them explicitly. We propose an algorithm to this end that optimises the compressed total size by specialising copies of the original compressor to the different partial distributions of the data.

Throughout our research we observed (see Chapters 2.5 and 3) that by partitioning a database, e.g. on class label, compressors are obtained that encode transactions from their 'native' distribution shortest. This observation suggests an iterative algorithm that resembles k-means: without any prior knowledge, randomly split the data in a fixed number of parts, induce a compressor for each and re-assign each transaction to the compressor that encodes it shortest, etcetera. This scheme is very generic and could also be easily used with other types of data and compressors. Here, as we are concerned with transaction data, we employ Krimp as compressor.

Both algorithms are implemented and evaluated on basis of total compressed sizes and component purity, but we also look at (dis)similarities between the components and the code tables. The results show that both our orthogonal methods identify the components of the database, without parameters: the optimal number of components is determined by MDL. Visual inspection confirms that characteristic decompositions are identified.

4.2 Problem Statement

Preliminaries

In this chapter we use the same notions and notations regarding databases, itemsets as in Chapter 2. Also, we will use MDL, and the Krimp algorithm, as introduced in that chapter. Further, we will use the database dissimilarity measure introduced in the previous chapter to determine how different the identified database components are. Two databases are deemed very similar

(possibly identical) iff the score is 0, higher scores indicate greater dissimilarity.

As discussed in Chapter 2, code tables characterise the sampled distributions of the data. Hence, as we want a partitioning for which the different components have different characteristics, they should have different code tables. In terms of MDL, this is stated as follows.

The problem

Our goal is to discover an optimal partitioning of the database; optimal, in the sense that the characteristics of the different components are different, while the individual components are homogeneous.

As discussed before, code tables characterise the sampled distributions of the data. Hence, we want a partitioning for which the different components have different characteristics, and thus code tables. In terms of MDL, this is stated as follows.

Problem 1. *Let $\mathcal{I}$ be a set of items and let $\mathcal{D}$ be a bag of transactions over $\mathcal{I}$. Find a partitioning $\mathcal{D}_1, \cdots, \mathcal{D}_k$ of $\mathcal{D}$ and a set of associated code tables $CT_1, \cdots, CT_k$, such that* the total encoded size *of $\mathcal{D}$,*

$$\sum_{i \in \{1, \cdots, k\}} L(CT_i, \mathcal{D}_i),$$

is minimised.

There are a few things one should note about this statement. First, we let MDL determine the optimal number of components for us, by picking the smallest encoded size over every possible partitioning and all possible code tables. Note that this also ensures that we will not end up with two components that have the same or highly similar code tables. It would be far cheaper to combine these.

Secondly, asking for both the database partitioning and the code tables is in a sense redundant. For any partitioning, the best associated code tables are, of course, the optimal code tables. The other way around, given a set of code tables, a database partitions naturally. Each transaction goes to the code table that compresses it best. This suggests two ways to design an algorithm to solve the problem. Either one tries to find an optimal partitioning or one tries to find an optimal set of code tables.

Thirdly, the size of the search space is gigantic. The number of possible partitions of a set of n elements is the well-known Bell number B_n. Similarly, the number of possible code tables is enormous. It consists of the set of all sets of subsets of that contain at least the singletons. Further, for each of these code tables we would have to consider all possible combinations of covers per transaction. Moreover, there is no structure in this search space that we can

use to prune. Hence, we will have to introduce heuristic algorithms to solve our problem.

Datasets

We use a number of UCI datasets [33], the *Retail* dataset [17] and the *Mammals* dataset[2] [58] for experimental validation of our methods. The latter consists of presence/absence records of 121 European mammals in geographical areas of 50×50 kilometres . The properties of these datasets are listed in Table 4.1. For fair comparison between the found components and the original classes, the class labels are removed from the databases in our experiments. In addition, KRIMP candidates (all or closed frequent itemsets up to minsup), the regular (single-component) KRIMP compressed size of the database and class dissimilarities are given. Average class dissimilarities are computed by splitting the database on class label, computing pair-wise code table dissimilarities as defined above and taking the average over these. Both *Retail* and *Mammals* datasets do not contain class labels.

4.3 Model-Driven Component Identification

In this section we present an algorithm that identifies components by finding an optimal set of code tables.

Motivation

A code table induced for a complete database captures the entire distribution, so the multiple underlying component distributions are modelled implicitly. This suggests that we should be able to extract code tables for specific components from the code table induced on the whole database, the original code table.

If the data in a database is a mixture of several underlying data distributions, the original code table can be regarded as a mixture of code tables. This implies that all patterns required for the components are in the code table and we only need to 'extract' the individual components. In this context, each possible subset of the original code table is a candidate component and thus each set of subsets a possible decomposition. So, given an original code table CT induced for a database $\mathcal{D}$, the optimal model driven decomposition is the set of subsets of CT that minimises the total encoded size of $\mathcal{D}$.

Obviously, the optimal decomposition could be found by simply trying all possible decompositions. However, although code tables consist of only few

[2]The full dataset [96] is available for research purposes from the Societas Europaea Mammalogica,`http://www.european-mammals.org`

Table 4.1: Basic statistics and Krimp statistics of the datasets used in the experiments.

	Basic Statistics			
Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert$	$\lvert C\rvert$	*pure*
Adult	48842	95	2	76.1
Anneal	898	65	6	76.2
Chess (kr–k)	28056	40	18	16.2
Mammals	2183	121	-	-
Mushroom	8124	117	2	51.8
Nursery	12960	21	2	33.3
Pageblocks	5473	33	11	89.8
Retail	88162	16470	-	-
	Krimp			
Dataset	*Cand's*	*ms*	$L(CT, \mathcal{D})$	DS
Adult	Closed	20	841604	0.8
Anneal	All	1	21559	6.3
Chess (kr–k)	All	10	516623	1.3
Mammals	Closed	150	164912	-
Mushroom	Closed	1	272600	8.4
Nursery	Closed	1	240537	4.2
Pageblocks	All	1	7160	10.6
Retail	All	16	10101135	-

Statistics on the datasets used in the experiments. As basic statistics, provided are the number of transactions, number of items, number of classes and purity (pure, the accuracy by majority class voting). Krimp-specific are the candidates used (all or closed frequent itemsets), the minsup(ms) at which the candidates are mined, the total compressed size in bits and the average code table dissimilarity between the classes.

Algorithm 6 Model-Driven Component Identification

IdentifyTheComponentsByModel $(\mathcal{D}, minsup)$:

1: $CT_{orig} \leftarrow$ Krimp($\mathcal{D}$, MineFreqSets($\mathcal{D}, minsup$))
2: $b \leftarrow \arg\min_{k \in [1, |\mathcal{D}|]}$ CalcEncSize(IdKComponentsByModel($\mathcal{D}, CT_{orig}, k$))
3: **return** IdentifyKComponents($\mathcal{D}, CT_{orig}, b$)

IdKComponentsByModel $(\mathcal{D}, CT_{orig}, k)$:

4: $C \leftarrow$ { k Laplace corrected copies of CT_{orig} }
5: **do** $e \leftarrow$ FindBestElimination $(\mathcal{D}, C, k)$
6: $C \leftarrow$ ApplyElimination(C, e)
7: **while** *compressed size decreases*
8: **return** C

FindBestElimination $(\mathcal{D}, C, k)$:

9: $(C_b, B) \leftarrow \arg\min_{C_i \in C, X \in C_i}$ CalcEncSize$(\mathcal{D}, (C \setminus C_i) \cup (C_i \setminus X), k)$
10: **return** (C_b, B)

CalcEncSize $(\mathcal{D}, C, k)$:

11: **for each** $t \in \mathcal{D}$: assign t to $CT_i \in C$ that gives the shortest code
12: **for each** $CT_i \in C$: ComputeOptimalCodes$(\mathcal{D}_i, CT_i)$
13: **return** $\sum_i L(C_i, \mathcal{D}_i)$

patterns (typically hundreds), the search space for such approach is enormous. Allowing only partitions of the code table would strongly reduce the size of the search space. But, as different distributions may have overlapping common elements, this is not a good idea. The solution is therefore to apply a heuristic.

Algorithm

The algorithm, presented in detail in Algorithm 6, works as follows: first, obtain the overall code table by applying Krimp to the entire database (line 1). Then, for all possible values of k, i.e. $[1, |\mathcal{D}|]$, identify the best k components. We return the solution that minimises the total encoded size (2-3).

To identify k components, start with k copies of the original code table (6). A Laplace correction of 1 is applied to each copy, meaning that the frequencies of all itemsets in the code table are increased by one. This correction ensures that each code table can encode any transaction, as no itemsets with zero frequency (and therefore no code) can occur. Now, iteratively eliminate that code table element that reduces the total compressed size most; until compression cannot be improved any further (5-7). Iteratively, each element in each code table is temporarily removed to determine the best possible elimination (9). To

Table 4.2: Experimental results for Model-Driven Component Identification

	Model-Driven Component Identification				
Dataset	*Gain L%*	*Purity (%)*	*avg. DS*	*opt. k*	*max. k*
Adult	8.7	76.1	6.44	2	2
Anneal	18.1	80.8	5.96	19	30
Chess (kr–k)	14.5	18.2	2.86	6	7
Mushroom	25.7	88.2	11.32	12	15
Nursery	11.7	45.0	1.77	14	20
PageBlocks	46.4	91.5	804.9	6	40

Experimental results for Model-Driven Component Identification. Given are the gain in compression over the single component Krimp compression, component purity by majority class voting, average dissimilarity between the components, the optimal value of k and the maximum value of k.

compute the total compressed size, each transaction is assigned to that code table that compresses it best (11). This can be translated as being the Bayes optimal choice (see Chapter 2.5). After this, re-compute optimal code lengths for each component (frequencies may have changed) and compute the total encoded size by summing the sizes of the individual components (12-13).

Experiments

The results of the experiments with the algorithm just described are summarised in Table 4.2. By running IdentifyTheComponentsByModel for all values of k and choosing the smallest decomposition, MDL identifies the optimal number of components. However, the search space this algorithm considers grows enormously for large values of k, so in our experiments we imposed a maximum value for k.

The compression gain is the reduction in compressed size relative to that attained by the original code table; the regular Krimp compressed size as given in Table 4.1. The compression gains in Table 4.2 show that the algorithm ably finds decompositions that allow for a much better compression than when the data is considered as a single component. By partitioning the data, reductions from 9% up to 46% are obtained. In other words, for all datasets compression improves by using multiple components.

We define purity as the weighted sum of individual component purities, a measure commonly used in clustering. Hence, the baseline purity is the percentage of transactions belonging to the majority class. If we look at the obtained purities listed in Table 4.2 and compare these to the baseline values

in Table 4.1, we notice that these values range from baseline to very good. Especially the classes of the *Mushroom* and *PageBlocks* datasets get separated well. The average (pairwise) dissimilarity between the Optimal k components, Average DS, shows how much the components differ from each other. We obtain average dissimilarities ranging from 1.7 to 804.9, which are huge considering the values measured between the actual classes (as given in Table 4.1). Without any prior knowledge the algorithm identifies components that are at least as different as the actual classes. While for the *Adult* database the obtained purity is at baseline, the data is very well partitioned: the dissimilarity between the components measures 6.4, opposed to just 0.8 between the actual classes.

Arguably, the most important part of the results is the code tables that form the end product of the algorithm: these provide the characterisation of the components. Inspection of these provides two main observations. First, a number of the original elements occur in multiple components, others occur only in a single component and some lost their use. Secondly, the code lengths of the elements are adapted to the specific components: codes become shorter, but lengths also change relative to each other. Example original and resulting code tables are depicted in Figure 4.1.

Discussion

The significantly smaller total encoded sizes after decomposition show that the proposed algorithm extracts different underlying distributions from the mixture. The purities and component dissimilarities show that components are different from each other but homogeneous within themselves.

One of the properties of the method is that the number of (unique) patterns required to define the components is never higher than the number of itemsets in the original code table, which consists of only few patterns (see Chapter 2.5). As the total number of patterns actually used in defining the components is often even smaller, the resulting code tables can realistically be manually inspected and interpreted by domain experts.

The running times for the reported experiments ranged from a few minutes for *Anneal* up to 80 hours for *Adult*. Obviously, more computation time is needed for very dense and/or very large databases, which is why in this section no results are presented for *Retail* and *Mammals*. While parallelisation of the algorithm is trivial and should provide a significant speedup, there is another approach to handle large datasets: the data driven method presented in the next section is inherently much faster.

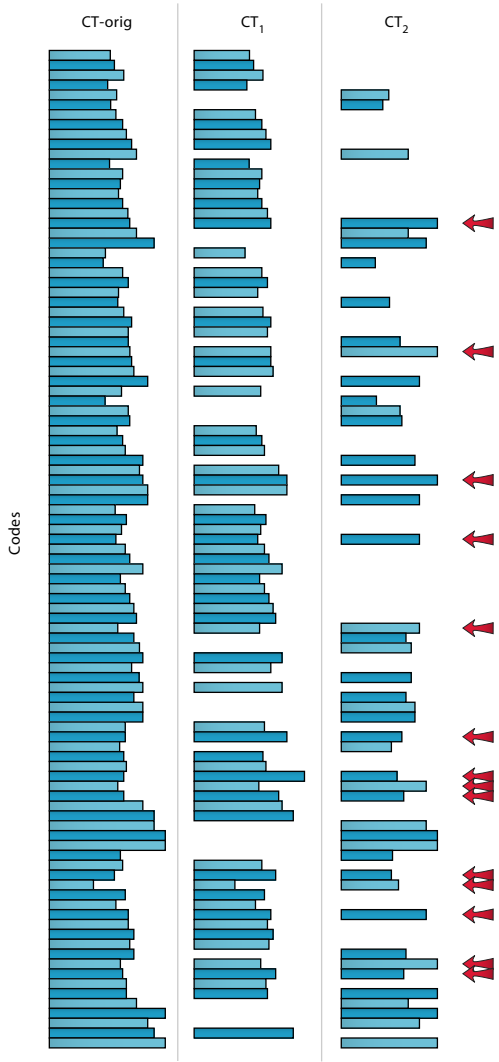


Figure 4.1: The components of *Anneal*. Codes and their lengths (in bits, represented by width) for original and component code tables, $k = 2$. Common elements are marked by arrows.

4.4 Data-Driven Component Identification

In this section, we present an approach to identify the components of a dataset by finding the MDL-optimal partitioning of the data.

Motivation

Suppose we have two equally sized databases, $\mathcal{D}_1$ and $\mathcal{D}_2$, both drawn from a mixture of distributions D_A and D_B. Further, suppose that $\mathcal{D}_1$ has more transactions from D_A than from D_B and vice versa for $\mathcal{D}_2$. Now, we induce compressors C_1 and C_2 for $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively. Assuming that D_A and D_B are different, C_1 will encode transactions from distribution D_A shorter than C_2, as C_1 has seen more samples from this distribution than C_2. This bias will be stronger if more transactions come from a single distribution - and the strongest when the data consists solely of transactions from one distribution. So, provided a particular source distribution has more transactions in one database than in another, transactions of that distribution will be encoded shorter by a compressor induced on that data.

We can exploit this property to find the components of a database. Given a partitioning of the data, we can induce a code table for each part. Now, we simply reassign each transaction to that part whose corresponding code table encodes it shortest. We thus re-employ the Bayes optimal choice to group those transactions that belong to similar distribution(s) (see Chapter 2.5. By doing this iteratively until all transactions remain in the same part, we can

Algorithm 7 Data-Driven Component Identification

IdentifyTheComponentsByData $(\mathcal{D}, minsup)$

1: **for** $k = 2$ **to** $|\mathcal{D}|$ **do** $r_k \leftarrow$ IdentifyKComponentsByData$(\mathcal{D}, k, minsup)$
2: $best \leftarrow \arg\min_{k \in [1,|\mathcal{D}|]}$ CalcEncodedSize$(\mathcal{D}, r_k, k)$
3: **return** r_{best}

IdentifyKComponentsByData $(\mathcal{D}, k, minsup)$

4: $parts \leftarrow$ Partition(RandomizeOrder$(\mathcal{D})$, k)
5: **do**
6: **for each** $p_i \in parts$ **do**
7: $CT_i \leftarrow$ Krimp(p_i, MineFreqSets$(p_i, minsup)$)
8: **for each** $X \in CT_i$ **do** $usage_{\mathcal{D}}(X) \leftarrow usage_{\mathcal{D}}(X) + 1$
9: **end for**
10: **for each** $t \in \mathcal{D}$ **do** assign t to CT_i that gives the shortest code
11: **while** *transactions were swapped*
12: **return** *parts*

Table 4.3: Experimental results for Data-Driven Component Identification

	Data-Driven Component Identification			
Dataset	*Gain L%*	*Purity* (%)	*avg. DS*	*opt. k*
Adult	40.3	82.2	31.7	177
Anneal	4.8	76.2	3.7	2
Chess (kr–k)	18.2	17.8	2.9	13
Mammals	46.2	-	1.7	6
Mushroom	14.7	75.6	7.8	20
Nursery	15.0	43.4	3.6	8
PageBlocks	70.3	92.5	19.6	30
Retail	25.5	-	0.10	2

Experimental results for Data-Driven Compnent Identification. Per database, the gain in compression over regular Krimp compression, component purity by majority class voting, average dissimilarity between the components and the optimal value of k are shown.

identify the components of the database. This method can be seen as a form of k-means; without the need for a distance metric and providing insight in the characteristics of the found groupings through the code tables.

Algorithm

From the previous subsection it is clear what the algorithm will look like, but its initialisation remains an open question. Without prior background knowledge, we start with a random initial partitioning. Because of this, it is required to do a series of runs for each experiment and let MDL pick the best result. However, as transactions are reassigned to better fitting components already in the first iteration, it is expected that the algorithm will be robust despite this initial non-deterministic step. The algorithm is presented in pseudo-code as Algorithm 7.

Experiments

Each experiment was repeated 10 times, each run randomly initialised. The different runs resulted in almost the same output, indicating that random initialisation does not harm robustness. In Table 4.3 we report the main results of these experiments. The reported results are those of the run with the shortest global encoded length.

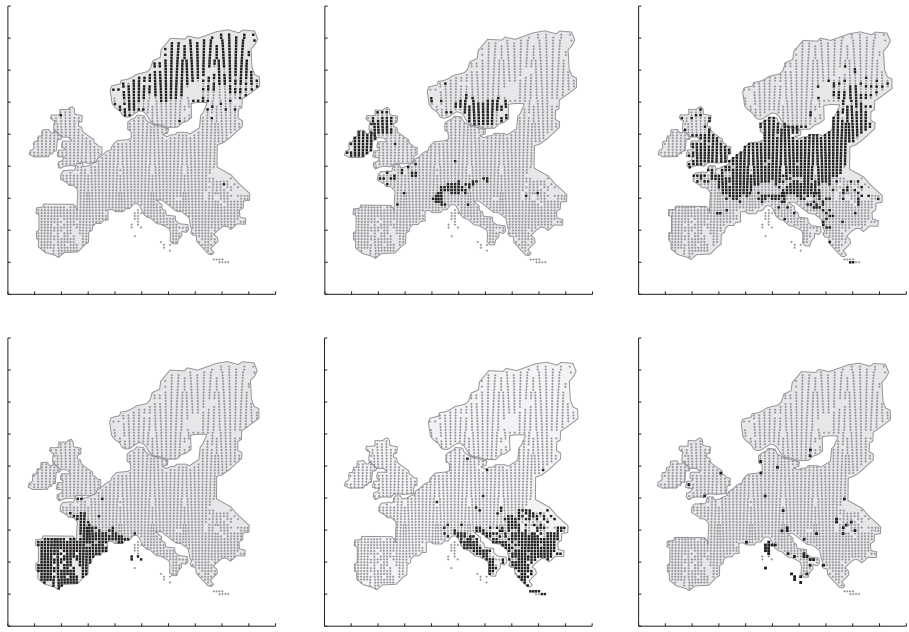


Figure 4.2: The components of *Mammals* ($k = 6$, optimal).

For all datasets decompositions are found that allow for much better compression of the whole. The gains show that very homogeneous blocks of data are identified; otherwise the total encodings would only have become longer, as many code tables have to be included instead of one. The components identified in the *Adult*, *Mammals* and *Pageblocks* datasets are so specific that between 40% and 70% fewer bits are required than when the data is compressed as a single block.

In between, however, the components are very heterogeneous. This is shown by the average dissimilarity measurements. For example, the 19.6 measured for *Pageblocks* means that on average 1960% more bits would be required for a random transaction, if it were encoded by a 'wrong' component. Also for the other datasets partitions are created that are at least as different as the original classes (see Table 4.1). Further, the component purities are in the same league as those of the model driven algorithm.

The geographic *Mammals* dataset allows us to meaningfully visually inspect the found components. Figure 4.2 shows the best found decomposition. Each of the rows (i.e. grid location of 50×50 kilometres) has been assigned to one of the six components based on its items (i.e. the mammals that have been recorded at that location). This is a typical example where normally ad-hoc

solutions are used [58] as it is difficult to define a meaningful distance measure. Our method only regards the characteristics of the components, the patterns in which the items occur. As can be seen in Figure 4.2, this results in clear continuous and geographically sound groupings - even though the algorithm was completely uninformed in this regard. For example, the top row components cover respectively the 'polar' region, highlands and more temperate areas of Europe.

Discussion

Much shorter data descriptions and high dissimilarities found show that highly specific database components are identified: internally homogeneous and heterogeneous in between. Moreover, the improvements in purity over the baseline show that many components have a strong relation to a particular class. Although mainly a property of the data, the optimal number of components we identify is low, which enables experts to analyse and interpret the components by hand.

By definition, each randomly initialised run of the algorithm may result in a different outcome. In all our experiments, however, we found the outcomes to be stable - indicating the robustness of a data compression approach. Nevertheless, to ensure that one finds a very good solution a couple of runs are required. However, as only partitions of the data have to be compressed, the algorithm runs very fast (a run typically takes seconds to minutes) so this poses no practical problems. For example, for the largest dataset, *Retail*, it took only six hours in total to run ten independent runs over all k.

4.5 Discussion

The components of a database can be identified using both model and data driven compression approaches. The experimental results show that both methods return characteristic partitions with much shorter total encoded size than regular single–component Krimp compression. The distributions of the components are shown to be very different from each other using dissimilarity measurements. These dissimilarities show that the code tables, and thus the patterns present in the data, are very unalike - i.e. the characteristics of the data components are different.

The optimal number of components is determined automatically by MDL by either method: no parameter k has to be set by hand. Each of the two proposed component identification methods has its own merit and depending on the data and computational power available one may choose for either the data or model driven algorithm. When dealing with (very) large databases, the data driven method is bound to provide good results quickly. For analysis

of reasonable amounts of data the model driven method has the advantage of characterising the components very well with only modest numbers of patterns. Although we here focus on transaction data and the KRIMP encoding, note that without much effort the framework can be generalised to a generic solution based on MDL: given a data type and suited encoding scheme, components can be identified by minimising the total compressed size. Especially the data driven algorithm is very generic and can be easily applied to other data types or adapted to use different compressors. This is trivial if a compressor can encode single transactions, otherwise one should assign each transaction to the component of which the encoded size of transaction and component is minimal. The model driven algorithm is more specific to our code table approach, but can also be translated to other data types.

4.6 Related Work

Clustering is clearly related to our work, as it addresses part of the problem we consider. The best-known clustering algorithm is k-means [88], to which our data driven component identifier is related as both iteratively reassign data points to the currently closest component. For this, k-means requires a distance metric, but it is often hard to define one as this requires prior knowledge on what distinguishes the different components. Our method does not require any a priori knowledge. In addition, clustering only aims at finding components, while here we simultaneously model these partitions explicitly with small pattern sets.

Local frequency of items has been used by Wang et al. [130] to form clusters, avoiding pair-wise comparisons but ignoring higher order statistics. Aggarwal et al. [1] describe a form of k-means clustering with market basket data, but for this a similarity measure on transactions is required. This measure is based on pair-wise item correlations; more complex (dis)similarities between transactions may be missed. Moreover, many parameters (eight) have to be set manually, including the number of clusters k.

Bi-clustering algorithms [104] look for linked clusters of both objects and attribute-value pairs called bi-clusters. These methods impose more restrictions on the possible clusters; for instance, they do not per se allow for overlap of the attribute-value pairs in the clusters. Further, the number of clusters k often has to be fixed in advance.

Koyotürk et al. [77] regarded item data as a decomposable matrix, of which the approximation vectors were used to build hierarchical cluster trees. However, rather many vectors are required for decomposition. Cadez et al. [24] do probabilistic modelling of transaction data to characterise (profile) what we would call the components within the data, which they assume to be known beforehand.

Recently, a number of information theoretic approaches to clustering have been proposed. The Information Bottleneck [119] can be used to find clusters by minimising information loss.

In particular, LIMBO [9] focuses on categorical databases. Main differences with our approach are that the number of clusters k has to be specified in advance, and as a lossy compression scheme is used MDL is not applicable.

Entropy measures allow for non-linearly defined clusters to be found in image segmentation tasks [51]. The MDL principle was used for vector quantization [14], where superfluous vectors were detected via MDL. Böhm et al. [15] used MDL to optimise a given partitioning by choosing specific models for each of the parts. These model-classes need to be pre-defined, requiring premonition of the component models in the data. Cilibrasi and Vitányi [31] used pairwise compression to construct hierarchical cluster trees with high accuracy. The method works well for a broad range of data, but performance deteriorates for larger (>40 rows) datasets. Kontkanen et al. [73] proposed a theoretical framework for data clustering based on MDL which shares the same global code length criterion with our approach, but does not focus on transaction databases.

4.7 Conclusion

Transaction databases are mixtures of samples from different distributions; identifying the components that characterise the data, without any prior knowledge, is an important problem. We formalise this problem in terms of total compressed size using MDL: the optimal decomposition is that set of code tables and partitioning of the data that minimises the total compressed size. No prior knowledge on the distributions, distance metric or the number of components has to be known or specified.

We presented two approaches to solve the problem and provide parameter-free algorithms for both. The first algorithm provides solutions by finding optimal sets of code tables, while the second does this by finding the optimal data partitioning. Both methods result in significantly improved compression when compared to compression of the database as a whole. Component purity and dissimilarity results confirm our hypothesis that characteristic components can be identified by minimising total compressed size. Visual inspection shows that MDL indeed selects sound groupings.

The approach we present can easily be adopted for other data types and compression schemes. The Krimp-specific instantiation for categorical data in this chapter shows that the MDL principle can be successfully applied to this problem. The data driven method especially is very generic and only requires a compression scheme that approximates the Kolmogorov Complexity of the data.

CHAPTER 5

Data Generation for Privacy Preservation

Many databases will not or can not be disclosed without strong guarantees that no sensitive information can be extracted. To address this concern several data perturbation techniques have been proposed. However, it has been shown that either sensitive information can still be extracted from the perturbed data with little prior knowledge, or that many patterns are lost.

In this chapter we show that generating new data is an inherently safer alternative. We present a data generator based on the models obtained by the MDL-based KRIMP algorithm. These are accurate representations of the data distributions and can thus be used to generate data with the same characteristics as the original data.

Experimental results show a very large pattern-similarity between the generated and the original data, ensuring that viable conclusions can be drawn from the anonymised data. Furthermore, anonymity is guaranteed for suited databases and the quality-privacy trade-off can be balanced explicitly.[1]

[1] This is an extended and edited version of work originally published as [125]: Vreeken, J., Van Leeuwen, M. and Siebes, A. (2007) Preserving Privacy through Data Generation. In *Proceedings of the ICDM' 07*, pages 685-690.

5.1 Introduction

Many databases will not or can not be disclosed without strong guarantees that no sensitive information can be extracted from it. The rationale for this ranges from keeping competitors from obtaining vital business information to the legally required protection of privacy of individuals in census data. However, it is often desirable or even required to publish data, leaving the question how to do this without disclosing information that would compromise privacy.

To address these valid concerns, the field of privacy-preserving data mining (PPDM) has rapidly become a major research topic. In recent years ample attention is being given to both defender and attacker stances, leading to a multitude of methods for keeping sensitive information from prying eyes. Most of these techniques rely on perturbation of the original data: altering it in such a way that given some external information it should be impossible to recover individual records within certainty bounds.

Data perturbation comes in a variety of forms, of which adding noise [7], data transformation [3] and rotation [30] are the most commonly used. At the heart of the PPDM problem is the balance between the quality of the released data and the amount of privacy it provides. While privacy is easily ensured by strongly perturbing the data, the quality of conclusions that can be drawn from it diminishes quickly. This is inherent of perturbation techniques: sensitive information cannot be fully masked without destroying non-sensitive information as well [64]. This is especially so if no special attention is given to correlations within the data by means of multidimensional perturbation [61], something which has hardly been investigated so far [86].

An alternative approach to the PPDM problem is to generate new data instead of perturbing the original. This has the advantage that the original data can be kept safe as the generated data is published instead, which renders data recovery attacks useless. To achieve this, the expectation that a data point in the generated database identifies a data point in the original database should be very low, whilst all generated data adhere to the characteristics of the original database. Data generation as a means to cover-up sensitivities has been explored in the context of statistical databases [82], but that method ignores correlations as each dimension is sampled separately.

We propose a novel method that uses data generation to guarantee privacy while taking important correlations into account. For this we use Krimp , for which we have shown that it provides accurate pattern-based approximations of data distributions. Using the patterns picked by MDL, we can construct a model that generates data very similar (but not equal) to the original data. Experiments show that the generative model is well suited for producing data that conserves the characteristics of the original data while preserving privacy.

Using our generative method, it is easy to ensure that generated data points cannot reliably be traced to individual data points in the original data. We

can thus easily obtain data that is in accordance with the well-known privacy measure k-anonymity [111]. Also, we can mimic the effects that can be obtained with l-diversity [87].

Although preserving intrinsic correlations is an important feat, in some applications preservation of particular patterns might be highly undesirable from a privacy point of view. Fortunately, this can easily be taken care of in our scheme by influencing model construction.

5.2 The Problem

Data perturbation

Since Agrawal & Srikant [7] initiated the privacy-preserving data mining field, researchers have been trying to protect and reconstruct sensitive data. Most techniques use data perturbation and these can be divided into three main approaches, of which we will give an overview here.

The addition of random noise to the original data, obfuscating without completely distorting it, was among the first proposals for PPDM [7]. However, it was quickly shown that additive randomisation is not good enough [4]. The original data can often be reconstructed with little error using noise filtering techniques [64] - in particular when the distortion does not take correlations between dimensions into account [61].

The second class is that of condensation-based perturbation [3]. Here, after clustering the original data, new data points are constructed such that cluster characteristics remain the same. However, it has been observed that the perturbed data is often too close to the original, thereby compromising privacy [30]. A third major data perturbation approach is based on rotation of the data [30]. While this method seemed sturdy, it has recently been shown that with sufficient prior knowledge of the original data the rotation matrix can be recovered, thereby allowing full reconstruction of the original data [86]. In general, perturbation approaches suffer from the fact that original data is used as starting point. Little perturbation can be undone, while stronger perturbation destroys both correlations and non-sensitive information. In other words, there is a privacy-quality trade-off that can not be balanced well.

In the effort to define measures on privacy, a few models have been proposed that can be used to obtain a definable amount of privacy. An example is the well-known k-anonymity model that ensures that no private information can be related to fewer than k individuals [111]. A lack of diversity in such masses can thwart privacy though and in some situations it is well possible to link private information to individuals. Improving on k-anonymity, the required critical diversity can be ensured using the l-diversity model. However, currently the available method can only ensure diversity for one sensitive attribute [87].

Data generation

The second category of PPDM solutions consists of methods using data generation, generating new (privacy preserving) data instead of altering the original. This approach is inherently safer then data perturbation, as newly generated data points can not be identified with original data points. However, not much research has been done in this direction yet.

Liew et al. [82] sample new data from probability distributions independently for each dimension, to generate data for use in a statistical database. While this ensures high quality point estimates, higher order dependencies are broken - making it unsuited for use in data mining.

The condensation-based perturbation approach [3] could be regarded as a data generation method, as it samples new data points from clusters. However, as mentioned above, it suffers from the same problems as perturbation techniques.

Problem statement

Reviewing the goals and pitfalls of existing PPDM methods, we conclude that a good technique should not only preserve privacy but also quality. This is formulated in the following problem statement:

A database $\mathcal{D}_{priv}$ induced from a database $\mathcal{D}_{orig}$ is privacy and quality preserving iff:

1. no sensitive information in $\mathcal{D}_{orig}$ can be derived from $\mathcal{D}_{priv}$ given a limited amount of external information (privacy requirement).

2. models and patterns derived from $\mathcal{D}_{priv}$ by data mining techniques are also valid for $\mathcal{D}_{orig}$ (quality requirement).

From this statement follows a correlated data generation approach to induce a privacy and quality preserving database $\mathcal{D}_{priv}$ from a database $\mathcal{D}_{orig}$, for which the above requirements can be translated into concrete demands.

Using Krimp, construct a model that encapsulates the data distribution of $\mathcal{D}_{orig}$ in the form of a code table consisting of frequent patterns. Subsequently, transform this code table into a pattern-based generator that is used to generate $\mathcal{D}_{priv}$.

It is hard to define an objective measure for the privacy requirement, as all kinds of 'sensitive information' can be present in a database. We guarantee privacy in two ways. Firstly, the probability that a transaction in $\mathcal{D}_{orig}$ is also present in $\mathcal{D}_{priv}$ should be small. Secondly, the more often a transaction occurs in $\mathcal{D}_{orig}$, the less harmful it is if it also occurs in $\mathcal{D}_{priv}$. This is encapsulated in the *Anonymity Score*, in which transactions are grouped by the number of times a transaction occurs in the original database (support):

Definition 7. *For a database $\mathcal{D}_{priv}$ based on $\mathcal{D}_o$, define the* Anonymity Score *(AS) as:*

$$AS(\mathcal{D}_{priv}, \mathcal{D}_{orig}) = \sum_{supp \in \mathcal{D}_{orig}} \frac{1}{supp} P(t \in \mathcal{D}_{priv} | t \in \mathcal{D}_{orig}^{supp})$$

In this definition, $\mathcal{D}^{supp}$ is defined as the selection of $\mathcal{D}$ with only those transactions having a support of *supp*. For each support level in $\mathcal{D}_{orig}$, a score is obtained by multiplying a penalty of 1 divided by the support with the probability that a transaction in $\mathcal{D}_{orig}$ with given support also occurs in $\mathcal{D}_{priv}$. These scores are summed to obtain AS. Note that when all transactions in $\mathcal{D}_{orig}$ are unique (i.e. have a support of 1), AS is equal to the probability that a transaction in $\mathcal{D}_{orig}$ also occurs in $\mathcal{D}_{priv}$.

Worst case is when all transactions in $\mathcal{D}_{orig}$ also occur in $\mathcal{D}_{priv}$. In other words, if we choose $\mathcal{D}_{priv}$ equal to $\mathcal{D}_{orig}$, we get the highest possible score for this particular database, which we can use to normalise between 1 (best possible privacy) and 0 (no privacy at all):

Definition 8. *For a database $\mathcal{D}_{priv}$ based on $\mathcal{D}_{orig}$, define the* Normalised Anonymity Score *(NAS) as:*

$$NAS(\mathcal{D}_{priv}, \mathcal{D}_{orig}) = 1 - \frac{AS(\mathcal{D}_{priv}, \mathcal{D}_{orig})}{AS(\mathcal{D}_{orig}, \mathcal{D}_{orig})}$$

To conform to the quality requirement, the frequent pattern set of $\mathcal{D}_{priv}$ should be very similar to that of $\mathcal{D}_{orig}$. We will measure pattern-similarity in two ways: 1) on database level through a database dissimilarity measure (see Chapter 3) and 2) on the individual pattern level by comparing frequent pattern sets. For the second part, pattern-similarity is high iff the patterns in $\mathcal{D}_{orig}$ also occur in $\mathcal{D}_{priv}$ with (almost) the same support. So:

$$P(|supp_{priv} - supp_{orig}| > \delta) < \epsilon \tag{5.1}$$

The probability that a pattern's support in $\mathcal{D}_{orig}$ differs much from that in $\mathcal{D}_{priv}$ should be very low: the larger δ, the smaller ϵ should be. Note that this second validation implies the first: only if the pattern sets are highly similar, the code tables become similar, which results in low measured dissimilarity. Further, it is computationally much cheaper to measure the dissimilarity than to compare the pattern sets.

5.3 Preliminaries

In this chapter we discuss categorical databases, and therefore adopt slightly different notation than in the previous chapters. A database $\mathcal{D}$ is a bag of

tuples (or transactions) that all have the same attributes $\{A_1, \ldots, A_n\}$. Each attribute A_i has a discrete domain of possible values $V_i \in \mathcal{V}$.

The Krimp algorithm introduced in Chapter 2 playes a major role here. No pruning strategy is applied in this chapter, since keeping all patterns in the code table causes more diversity during data generation, as will become clear later.

Krimp operates on itemset data, as which categorical data can easily be regarded. The union of all domains $\cup V_i$ forms the set of items $\mathcal{I}$. Each transaction t can now also be regarded as a set of items $t \subseteq \mathcal{P}(\mathcal{I})$. An itemset $X \subseteq \mathcal{I}$ occurs in a transaction $t \in \mathcal{D}$ iff $X \subseteq t$. The support of X in $\mathcal{D}$ is the number of transactions in the database in which X occurs. Speaking in market basket terms, this means that each item for sale is represented as an attribute, with the corresponding domain consisting of the values 'bought' and 'not bought'.

Further, in this work we use the database dissimilarity measure DS introduced in Chapter 3. Recall that two databases are deemed very similar (possibly identical) iff the score is 0, higher scores indicate higher levels of dissimilarity. As the code tables consist of frequent patterns, it is especially good at measuring the pattern similarity on a database level, as experiments in Chapter 3 confirmed. We will therefore use it in our experimental section to quantify the differences between original and generated data, helping to verify the quality requirement of the problem statement.

5.4 Krimp Categorical Data Generator

In this section we present our categorical data generation algorithm. We start off with a simple example, sketching how the algorithm works by generating a single transaction. After this we will detail the scheme formally and provide the algorithm in pseudo-code.

Generating a transaction, an example

Suppose we need to generate a new transaction for a simple three-column categorical database. To apply our generation scheme, we need a domain definition $\mathcal{V}$ and a Krimp code table CT, both shown in Figure 5.1.

We start off with an empty transaction and fill it by iterating over all domains and picking an itemset from the code table for each domain that has no value yet. We first want to assign a value for the first domain, V_1, so we have to select one pattern from those patterns in the code table that provide a value for this domain. This subset is shown as selection CT^{V_1}.

Using the usage frequencies of the code table elements as probabilities, we randomly select an itemset from CT^{V_1}; elements with high usage occur more often in the original database and are thus more likely to be picked. Here we

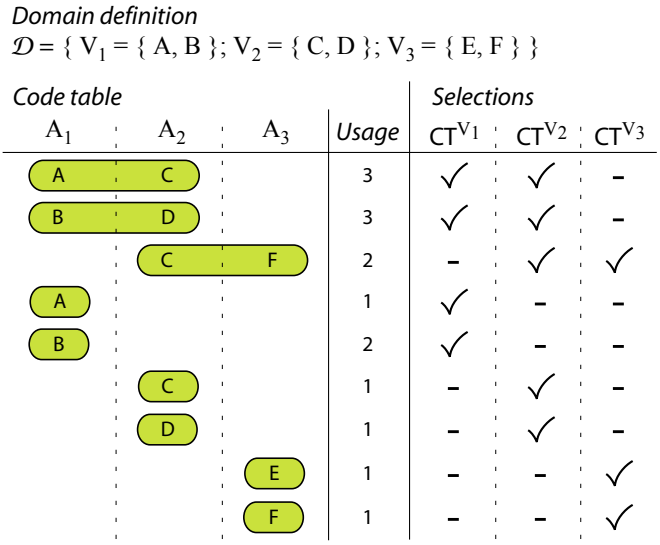


Figure 5.1: Example for a 3–column database. Each usage frequency is Laplace corrected by 1.

randomly pick 'BD' (probability 3/9). This set selects value 'B' from the first domain, but also assigns a value to the second domain, namely 'D'.

To complete our transaction we only need to choose a value for the third domain. We do not want to change any values once they are assigned, as this might break associations within an itemset previously chosen. So, we do not want to pick any itemset that would re-assign a value to one of the first two domains. Considering the projection for the third domain, CT^{V_3}, we thus have to ignore set CF(2), as it would re-assign the second domain to 'C'. From the remaining sets E(3) and F(3), both with usage 3, we randomly select one - say, 'E'. This completes generation of the transaction: 'BDE'. With different rolls of the dice it could have generated 'BCF' by subsequently choosing CF(2) and B(3), and so on.

Here we will detail our data generator more formally. First, define the projection CT^V as the subset of itemsets in CT that define a value for domain $V \in \mathcal{V}$. To generate a database, our categorical data generator requires four ingredients: the original database, a Laplace correction value, a *minsup* value for mining candidates for the Krimp algorithm and the number of transactions that is to be generated. We present the full algorithm as Algorithm 8.

Generation starts with an empty database $\mathcal{D}_g$ (line 1). To obtain a code

Algorithm 8 Krimp Categorical Data Generator

GenerateDatabase $(\mathcal{D}, laplace, minsup, numtrans)$:

1: $\mathcal{D}_g \leftarrow \emptyset$
2: $CT \leftarrow$ Krimp($\mathcal{D}$, MineCandidates($\mathcal{D}, minsup$))
3: **for each** itemset $X \in CT$ **do**
4: $\quad usage_{\mathcal{D}}(X) \leftarrow usage_{\mathcal{D}}(X) + laplace$
5: **end for**
6: $\mathcal{V} \leftarrow$ domains from $\mathcal{D}$
7: **while** $|\mathcal{D}_g| < numtrans$ **do**
8: $\quad \mathcal{D}_g \leftarrow \mathcal{D}_g \cup \{$ GenerateTransaction$(CT, \mathcal{V})$ $\}$
9: **end while**
10: **return** $\mathcal{D}_g$

GenerateTransaction $(CT, \mathcal{V})$:

11: $t \leftarrow \emptyset$
12: **while** $\mathcal{V} \neq \emptyset$ **do**
13: $\quad V \leftarrow$ a random element from $\mathcal{V}$
14: $\quad X \leftarrow$ PickRandomItemSet(CT^V)
15: $\quad t \leftarrow t \cup X$
16: $\quad$ **for each** domain W for which X has a value **do**
17: $\quad\quad CT \leftarrow CT \setminus CT^W$
18: $\quad\quad \mathcal{V} \leftarrow \mathcal{V} \setminus W$
19: $\quad$ **end for**
20: **end while**
21: **return** t

PickRandomItemSet (CT) :

22: $weights \leftarrow \{\ usage_{\mathcal{D}}(X) \mid X \in CT\ \}$
23: $X \leftarrow$ WeightedSample$(weights, CT)$
24: **return** X

table CT, the Krimp algorithm is applied to the original database $\mathcal{D}$ (2). A Laplace correction *laplace* is added to all elements in the code table (lines 3 and 4). Next, we return the generated database when it contains *numtrans* transactions (lines 7 to 10).

Generation of a transaction is started with an empty transaction t (line 11). As long as $\mathcal{V}$ is not empty (12), our transaction is not finished and we continue. First, a domain V is randomly selected (13). From the selection CT^V, one itemset is randomly chosen, with probabilities defined by their relative usage frequencies (14). After the chosen set is added to t (15), we filter from CT all sets that would redefine a value - i.e. those sets that intersect with the definitions of the domains for which t already has a value (lines 16 and 17).

Further, to avoid reconsideration we also filter these domains from $\mathcal{V}$ (18). After this the next domain is picked from $\mathcal{V}$ and another itemset is selected; this is repeated until $\mathcal{V}$ is empty (and t thus has a value from each domain).

Note that code table elements are treated fully independently, as long as they do not re-assign values. Correlations between dimensions are stored explicitly in the itemsets and are thus taken into account implicitly.

Besides the original database and the desired number of generated transactions, the database generation algorithm requires two other parameters: *laplace* and *minsup*. Both fulfil an important role in controlling the amount of privacy provided in the generated database, which we will discuss here in more detail.

A desirable parameter for any data generation scheme is one that controls the data diversity and strength of the correlations. In our scheme this parameter is found in the form of a Laplace correction. Before the generation process, a small constant is added to the usage frequencies of the code table elements. As code tables always contains all single values, this ensures that all values for all categories have at least a small probability of being chosen. Thus, 1) a complete transaction can always be generated and 2) all possible transactions can be generated. For this purpose the correction needs only be small. However, the strength of the correction influences the chance an otherwise unlikely code table element is used; with larger correction, the influence of the original data distribution is dampened and diversity is increased.

The second parameter to our database generation algorithm, *minsup*, has a strong relation to the k-anonymity blend-in-the-crowd approach. The *minsup* parameter has (almost) the same effect as k: patterns that occur less than *minsup* times in the original database are not taken into account by Krimp. As they cannot get in the code table, they cannot be used for generation either. Particularly, complete transactions have to occur at least *minsup* times in order for them to make it to the code table. In other words, original transactions that occur less often than *minsup* can only be generated if by chance often occurring patterns are combined such that they form an original transaction. As code table elements are regarded independent, it follows that when more patterns have to be combined, it becomes less likely that transactions are generated that also exist in the original database.

5.5 Experiments

In this section we will present empirical evidence of the method's ability to generate data that provides privacy while still allowing for high quality conclusions to be drawn from the generated data.

Table 5.1: Database characteristics and dissimilarities.

			Krimp	Dissimilarity (DS)	
Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{V}\rvert$	*minsup*	$\mathcal{D}_{priv}$ *vs.* $\mathcal{D}_{orig}$	$\mathcal{D}_{orig}$ *internal*
Chess (kr–k)	28056	7	1	0.037	0.104
Iris	150	5	1	0.047	0.158
Led7	3200	8	1	0.028	0.171
Letter recog	20000	17	50	0.119	0.129
Mushroom[2]	8124	22	20	0.010	0.139
Nursery	12960	9	1	0.011	0.045
Page blocks	5473	11	1	0.067	0.164
Pen digits	10992	17	50	0.198	0.124
Pima	786	9	1	0.110	0.177
Quest A	4000	8	1	0.016	0.077
Quest B	10000	16	1	0.093	0.223

Database characteristics, candidate minsup and dissimilarity measurements (between original and generated datasets) for a range of datasets. As candidates, frequent itemsets up to the given minimum support level were used.

Experimental setup

In our experiments, we use a selection from the commonly used UCI repository [33]. Also, we use two additional databases that were generated with IBM's Quest basket data generator [6]. To ensure that the Quest data obeys our categorical data definition, we transformed it such that each original item is represented by a domain with two categories, in a binary fashion (present or not). Both Quest datasets were generated with default settings, apart from the number of columns and transactions.

Characteristics of all used datasets are summarised in Table 5.1, together with the minimum support levels we use for mining the frequent itemsets that function as candidates for Krimp.

For all experiments we used a Laplace correction parameter of 0.001, an arbitrarily chosen small value solely to ensure that otherwise zero-usage code table elements can be chosen during generation. All experimental results presented below are averaged over 10 runs and all generated databases have the same number of transactions as the originals, unless indicated otherwise.

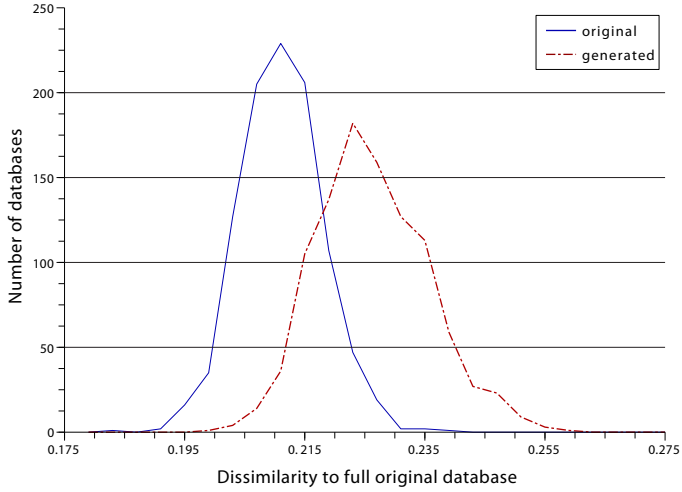


Figure 5.2: Histogram of dissimilarities between samples (original and generated) and the full original database, *Chess (kr–k)*.

Results

To quantify the likeness of the generated databases to their original counterparts, we use the database dissimilarity measure as described in Chapter 3. To judge these measurements, we also provide the dissimilarity between the original database and independent random samples of half the size from the original database.

In Table 5.1 we show both these internal dissimilarity scores and the dissimilarity measurements between the original and generated databases. To put the reported dissimilarities in perspective, note that the dissimilarity measurements between the classes in the original databases range from 0.29 up to 12 (see Chapter 3). The measurements in Table 5.1 thus indicate clearly that the generated databases adhere very closely to the original data distribution; even better than a randomly sampled subset of 50% of the original data captures the full distribution.

To show that the low dissimilarities for the generated databases are not caused by averaging, we provide a histogram in Figure reffig:chessdissimhist for the *Chess (kr–k)* dataset. We generated thousand databases of 7500 transactions, and measured the dissimilarity of these to the original database. Likewise, we also measured dissimilarity to the original database for equally many and equally sized independent random samples. The peaks for the distance histograms lie very near to each other at 0.21 and 0.22 respectively. This and

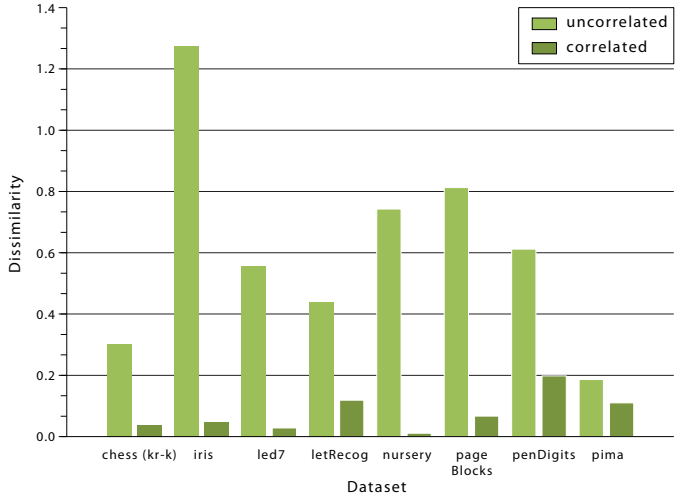


Figure 5.3: Dissimilarity scores between generated (with and without correlations) and original databases.

the very similar shapes of the histograms confirm that our generation method samples databases from the original distribution.

Turning back to Table 5.1, we notice that databases generated at higher values of the *minsup* parameter show slightly larger dissimilarity. The effect of this parameter is further explored in Figures 5.3 and 5.4. First, the bar diagram in Figure 5.3 shows a comparison of the dissimilarity scores between uncorrelated and correlated generation: uncorrelated databases are generated by a code table containing only individual values (and thus no correlations between domains can exist), correlated databases are generated using the *minsup* values depicted in Table 5.1 (at which the correlations in the data are captured in the patterns in the code table). We see that when generation is allowed to take correlations into account, the generated databases are far more similar to the original ones.

Secondly, the graph in Figure 5.4 shows the dissimilarity between the original *Pen digits* database and databases generated with different values for *minsup*. As expected, lower values of *minsup* lead to databases more similar to the original, as the code table can better approximate the data distribution of the original data. For the whole range of generated databases, individual value frequencies are almost identical to those of the original database; the increase in similarity is therefore solely caused by the incorporation of the right (type and strength of) correlations.

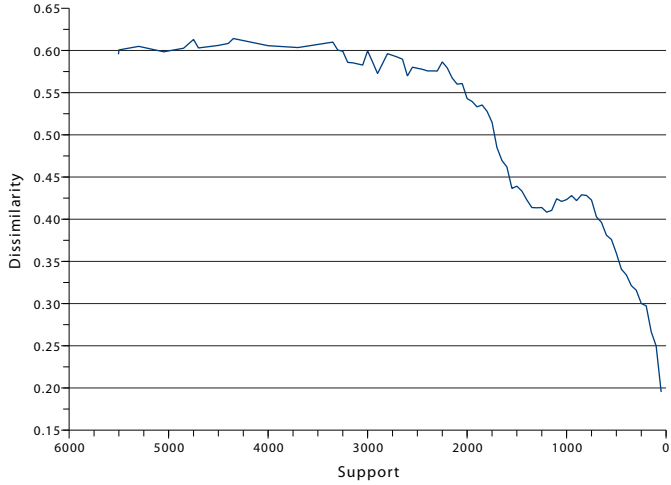


Figure 5.4: Dissimilarity between generated database (at different *minsup*s) and the original database for *Pen digits*.

Now that we have shown that the quality of the generated databases is very good on a high level, let us consider quality on the level of individual patterns. For this, we mined frequent itemsets from the generated databases with the same parameters as we did for candidate mining on the original database. A comparison of the resulting sets of patterns is presented in Table 5.2. We report these figures for those databases for which it was feasible to compute the intersection of the frequent pattern collections.

Large parts of the generated and original frequent pattern sets consist of exactly the same items sets, as can be seen from the first column. For example, for *Led7* and *Nursery* about 90% of the mined itemsets is equal. In the generated *Pen digits* database a relatively low 25% of the original patterns are found. This is due to the relatively high *minsup* used: not all correlations have been captured in the code table. However, of those patterns mined from the generated database, more than 90% is also found in the original frequent itemset collection.

For itemsets found in both cases, the average difference in support between original and generated is very small, as the second column shows. *Iris* is a bit of an outlier here, but this is due to the very small size of the dataset. Not only the average is low, standard deviation is also small: as can be seen from Figure 5.5, almost all sets have a very small support difference. The generated databases thus fulfil the support difference demands we formulated

Table 5.2: Frequent pattern set comparison.

Dataset	*% equal itemsets*	*% diff in supp equal itemsets*	*% supp new itemsets*
Chess (kr–k)	71	0.01	0.01
Iris	83	1.69	0.80
Led7	89	0.14	0.06
Nursery	90	0.04	0.03
Page blocks	75	0.06	0.02
Pen digits	25	0.50	0.59
Pima	60	0.30	0.14

Comparison between the frequent itemset collections mined on the original and generated data. Shown are, per dataset, the percentage of itemsets in both collections. For these itemsets, the average difference in relative support. For the itemsets only found in the generated data, their average relative support.

in Equation 5.1.

The third column of Table 5.2 contains the average supports of itemsets that are newly found in the generated databases; these supports are very low. All this together clearly shows that there is a large pattern-similarity, thus showing a high quality according to our problem statement.

However, this quality is of no worth if the generated data does not also preserve privacy. To measure the level of provided anonymity, we calculate the Normalised Anonymity Score as given by Definition 8. These scores are presented in Table 5.3.

As higher scores indicate better privacy, some datasets (e.g. *Mushroom*, *Pen digits*) are anonymised very well. On the other hand, other datasets (*Page blocks*, *Quest*) do not seem to provide good privacy. As discussed in Section 5.4, the *minsup* parameter of our generation methods doubles as a k-anonymity provider.

This explains that higher values for *minsup* result in better privacy, as the measurements for *Letter recognition*, *Mushroom* and *Pen digits* indeed show. Analogously, the (very) low *minsup* values used for the other databases result in lower privacy (aside from data characteristics to which we'll return shortly).

To show the effect of *minsup* in action, as an example we increase the *minsup* for the *Chess* database to 50. While the so-generated database is still very similar to the original (dissimilarity of 0.19), privacy is considerably increased - which is reflected by a Normalised Anonymity Score of 0.85. For further evidence of the k-anonymity obtained, we take a closer look at *Pen*

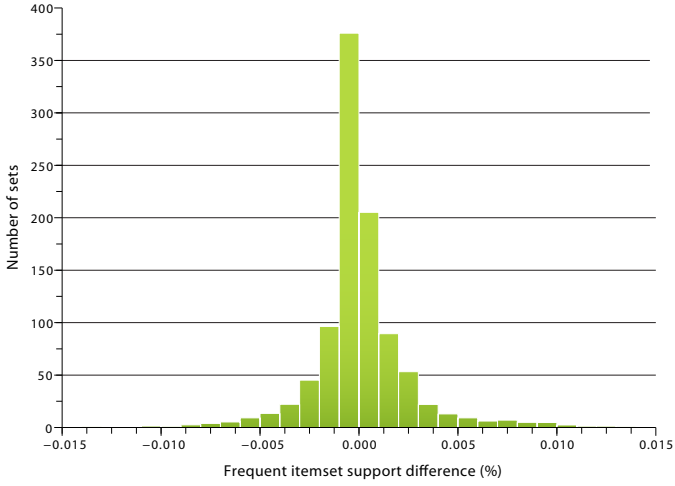


Figure 5.5: Difference in support, $supp(\mathcal{D}_{priv}) - supp(\mathcal{D}_{orig})$, for identical itemsets in generated and original *Led7*.

Table 5.3: Normalised Anonymity Scores.

Dataset	*NAS*
Chess (kr–k)	0.70
Iris	0.28
Led7	0.34
Letter recognition	0.69
Mushroom	0.91
Nursery	0.51
Page blocks	0.23
Pen digits	0.78
Pima	0.36
Quest A	0.16
Quest B	0.19

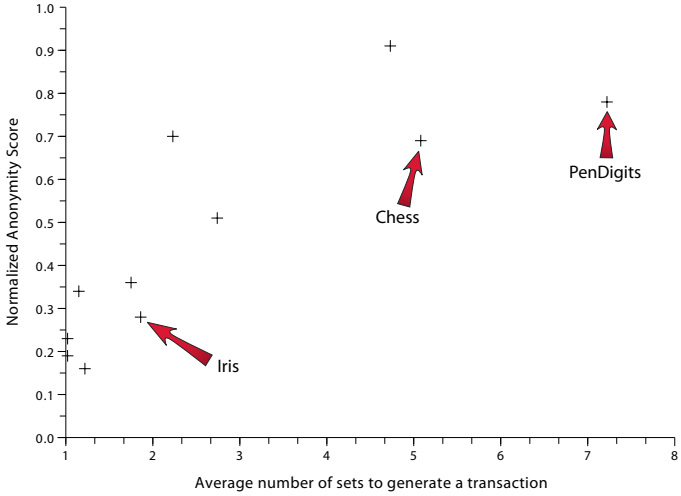


Figure 5.6: Average number of patterns used to generate a transaction versus Normalised Anonymity Score, for all datasets in Table 5.1.

digits, for which we use a *minsup* of 50. Of all transactions with support < 50 in the generated database, only 3% is also found in the original database with support < 50. It is thus highly unlikely that one picks a 'real' transaction from the generated database with support lower than *minsup*.

Although not all generated databases preserve privacy very well, the results indicate that privacy can be obtained. This brings us to the question when privacy can be guaranteed. This not only depends on the algorithm's parameters, but also on the characteristics of the data. It is difficult to determine the structure of the data and how the parameters should be set in advance, but during the generation process it is easy to check whether privacy is going to be good.

The key issue is whether transactions are generated by only very few or many code table elements. In Figure 5.6 we show this relation: for each dataset in Table 5.1, a cross marks the average number of itemsets used to generate a single transaction and the Normalised Anonymity Score. In the top-right corner we find the generated databases that preserve privacy well, including *Pen digits* and *Letter recognition*. At the bottom-left reside those databases for which too few elements per transaction are used during generation, leading to bad privacy; *Quest*, *Page blocks* and *Led7* are the main culprits. Thus, by altering the *minsup*, this relation allows for explicit balancing of privacy and quality of the generated data.

5.6 Discussion

The experimental results in the previous section show that the databases generated by our KRIMP Categorical Data Generator are of very high quality; pattern similarity on both database level and individual pattern level is very high. Furthermore, we have shown that it is possible to generate high quality databases while privacy is preserved. The Normalised Anonymity Scores for some datasets are rather high, indicating that hardly any transactions that occur few times in the original database also occur in the generated database. As expected, increasing *minsup* leads to better privacy, but dissimilarity remains good and thus the trade-off between quality and privacy can be balanced explicitly.

A natural link between our method and k-anonymity is provided by the *minsup* parameter, of which we have shown that it works in practice. While we haven't explored this parameter in this work, it is also possible to mimic l-diversity, as in our method the *laplace* parameter acts as diversity control. The higher the Laplace correction, the less strong the characteristics of the original data are taken into account (thus degrading quality, but increasing diversity). Note that one could also increase the Laplace correction for specific domains or values, thereby dampening specific (sensitive) correlations - precisely the effect l-diversity aims at.

To obtain even better privacy, one can also directly influence model construction: for example, by filtering the KRIMP candidates prior to building the code table. Correlations between specific values and/or categories can be completely filtered. If correlations between values A and B are sensitive, then by removing all patterns containing both A and B from the candidate set, no such pattern can be used for generation.

From Figure 5.6 followed that the number of patterns used to generate a transaction greatly influences privacy: more elements leads to higher anonymity. In the same line of thought, the candidate set can be filtered on pattern length; imposing a maximum length directly influences the number of patterns needed in generation, and can thus increase the provided anonymity.

The average number of patterns needed to generate a transaction is a good indication of the amount of anonymity. We can use this property to check whether parameters are chosen correctly and to give a clue on the characteristics of the data. If already at high *minsup* few patterns are needed to encode a transaction, and thus hardly any 'sensitive' transactions occur, the database is not 'suited' for anonymisation through generation.

Reconsidering our problem statement in Section 5.2, the KRIMP generator does a good job as solution for this PPDM problem. The concrete demands we posed for both the quality and privacy requirements are met, meaning that databases generated by our method are privacy and quality preserving as we interpreted this in our problem statement. Generating new data is therefore a

good alternative to perturbing the original data.

Our privacy-preserving data generation method could be well put to practice in the distributed system Merugu and Ghosh [94] proposed: to cluster privacy-preserving data in a central place without moving all the data there, a privacy-preserving data generator for each separate location is to be built. This is exactly what our method can do and this would therefore be an interesting application. As the quality of the generated data is very high, the method could also be used in limited bandwidth distributed systems where privacy is not an issue. For each database that needs to be transported, construct a code table and communicate this instead of the database. If precision on the individual transaction level is not important, new highly similar data with the same characteristics can be generated.

In this chapter, we generated databases of the same size as the original, but the number of generated transactions can of course be varied. Therefore, the method could also be used for up-sampling. Furthermore, it could be used to induce probabilities that certain transactions or databases are sampled from the distribution represented by a particular code table.

5.7 Conclusions

We introduce a pattern-based data generation technique as a solution to the privacy-preserving data mining problem in which data needs to be anonymised. Using the MDL-based Krimp algorithm we obtain accurate approximations of the data distribution, which we transform into high-quality data generators with a simple yet effective algorithm.

Experiments show that the generated data meets the criteria we posed in the problem statement, as privacy can be preserved while the high quality ensures that viable conclusions can still be drawn from it. The quality follows from the high similarity to the original data on both the database and individual pattern level. Anonymity scores show that original transactions occurring few times only show up in the generated databases with very low probability, giving good privacy.

Preserving privacy through data generation does not suffer from the same weaknesses as data perturbation. By definition, it is impossible to reconstruct the original database from the generated data, with or without prior knowledge. The privacy provided by the generator can be regulated and balanced with the quality of the conclusions drawn from the generated data. For suited data, the probability of finding a 'real' transaction in the generated data is extremely low.

CHAPTER 6

Krimp Minimisation for Missing Data Estimation

Many data sets are incomplete. For correct analysis of such data, one can either use algorithms that are designed to handle missing data or use imputation. Imputation has the benefit that it allows for any type of data analysis. Obviously, this can only lead to proper conclusions if the provided data completion is both highly accurate and maintains all statistics of the original data.

In this chapter, we present three data completion methods that are built on the MDL-based Krimp algorithm. Here, we also follow the MDL principle, i.e. the completed database that can be compressed best, is the best completion because it adheres best to the patterns in the data.

By using local patterns, as opposed to a global model, Krimp captures the structure of the data in detail. Experiments show that both in terms of accuracy and expected differences of any marginal, better data reconstructions are provided than the state of the art, Structural EM.[1]

[1] This chapter is an extended, edited, version of work published as [122]:
Vreeken, J., Siebes, A. (2008) Filling in the Blanks – Krimp Minimisation for Missing Data. In *Proceedings of the ICDM'08*, pages 1067-1072.

6.1 Introduction

Many data sets are incomplete. Whether dealing with surveys, DNA micro arrays or medical data, missing values are commonplace. However, properly dealing with missing values remains an open problem in data analysis. While some specialised algorithms exist that are designed to handle missing data, many are not, and can at most ignore missing values. From statistics, we know [55] this leads to biases in the outcome of the analysis.

There are two ways to properly analyse incomplete data:

- Using specialised algorithms designed to handle missing data
- Completing the data by imputation

Of these two, imputation has the practical advantage that one can analyse the completed database using any tool or method desired. Obviously, this does require the imputation to be as accurate as possible. That is, all statistics that one computes from the completed database should be as close as possible to those of the original data. The problem of imputation is thus: complete the database as well as possible.

However, determining what is 'good' cannot just be measured through accuracy: only for 100% correct estimations we know for sure that all statistics are maintained. In this chapter we consider 0/1 databases in particular and categorical databases in general. This allows us to properly validate the quality of a completed database in the following manner: we compare the support of a random itemset in the original (complete) database with its support in the completed database. The rationale is as follows. Analysing binary data is largely based on counting. If the support of all itemsets are correct, all counts will be correct. So, if for random itemsets that difference in support is nil, we know that all counts are identical. Consequently, all statistics computed on the completed database will be correct.

To achieve such high quality imputation we use the practical variant of Kolmogorov complexity, MDL (minimum description length), as our guiding principle: the completed database that can be compressed best is the best completion. The driving thought behind this approach is that a completion should comply to the local patterns in the database: not just filling in what globally would lead to the highest accuracy. By taking into account how specific values co-occur locally, not only the global statistics on the data will be correct but also those measured on the local level.

We approximate this best result using KM, which stands for Krimp Minimisation. It is an iterative approach in which each successive completion has a lower complexity as measured through the compressibility of the data. KM is built on the MDL-based Krimp algorithm (see Chapter 2), that provides

high quality data descriptions through compression of the data using frequent itemsets.

Most good algorithms for missing data first estimate a model on the data. Structural EM [45] is a good example of this. Within the iterative EM process the learning of a Bayes net is integrated. This leads to very good approximations of the networks underlying the data, as well as state of the art imputations.

KM is different in that it looks at the local patterns in the data, rather than a building global model; such local patterns are often smoothed out from a global stance. The experiments show that the local approach is indeed superior to the global approach, both in terms of accuracy and quality of the completed databases.

6.2 The Problem

Preliminaries

As in this chapter we have to determine whether items are present or not, i.e. whether their value is 0 or 1, we have to introduce some further notation.

Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ be a set of binary (0/1 valued) attributes. That is, the domain V_i of item I_i is $\{0, 1\}$. A transaction (or tuple) over $\mathcal{I}$ is an element of $\prod_{i \in \{1,\ldots,n\}} V_i$. A database $\mathcal{D}$ over $\mathcal{I}$ is a bag of tuples over $\mathcal{I}$. This bag is indexed in the sense that we can talk about the i-th transaction. Further, we follow the same notation as in Chapter 2.

Note that while we restrict ourself to binary databases in the description of our problem and algorithms, there is a trivial generalisation to categorical databases. In the experiments, we use such categorical databases.

In this chapter, the KRIMP algorithm plays an important role. For an introduction to MDL and KRIMP, please refer to Chapter 2.

Missing data

A database $\mathcal{D}$ has *missing data,* if some of its values are denoted by '?'. The ?-values denote that we do not know what the actual value is, it might be a 0 or a 1. In the traditional market-basket example, this means, e.g. that we do not know whether or not *beer* was bought in a given transaction.

The literature, see, e.g. [83, 112], distinguishes the following three different types of missing data mechanisms.

MCAR which stands for *missing completely at random.* It means that the fact that a data value is missing does not depend on any of the values in the transaction, including itself.

MAR which stands for *missing at random.* This means that the fact that a data value is missing may depend on one or more of the observed values, it does *not* depend on the 'true' value of any of the missing values.

NMAR which stands for *not missing at random.* This means that the fact that a data value is missing may depend on the true value of a missing data value.

NMAR is a very problematic case. Without background knowledge, unbiased analysis of the data is impossible. As the vast majority of the literature, we restrict ourselves to MAR and MCAR only.

Database completion

Completing a database simply means that each question mark is replaced by a definite value, i.e. a 1 or a 0. More formally we have the following definition.

Definition 9. *Let $\mathcal{D}_M$ and $\mathcal{D}_C$ be two databases over $\mathcal{I}$. Moreover, let $\mathcal{D}_M$ have missing data, whereas $\mathcal{D}_C$ is* complete*, i.e. $\mathcal{D}_C$ has no missing data. Then, $\mathcal{D}_C$ is a* completion *of $\mathcal{D}_M$ iff*

1. *$\mathcal{D}_M$ and $\mathcal{D}_C$ both have k transactions, $\mathcal{D}_M = \{t_1, \ldots, t_k\}$ and $\mathcal{D}_C = \{s_1, \ldots s_k\}$;*
2. *$\forall i \in \{1, \ldots, k\}\ \forall I \in \mathcal{I} : \pi_I(t_i) \in \{0, 1\} \rightarrow \pi_I(t_i) = \pi_I(s_i)$*

An algorithm $\mathcal{A}$ that completes any incomplete database is called a completion algorithm*.*

There are many possible completions of an incomplete database. In fact, if $\mathcal{D}_M$ has k unknown values, there are 2^k completions. Clearly, not all completions are equally useful. To define quality measures, we assume that we know the true complete database, denoted by $\mathcal{D}^T$. The most obvious quality measure of a completion is accuracy.

Definition 10. *Let $\mathcal{D}_M, \mathcal{D}_C$, and $\mathcal{D}^T$ be databases over $\mathcal{I}$, such that $\mathcal{D}_M$ is incomplete, $\mathcal{D}^T$ is the true completion of $\mathcal{D}_M$ and $\mathcal{D}_C$ is an arbitrary completion of $\mathcal{D}_M$. Moreover, let m be the number of missing values in $\mathcal{D}_M$ and n the number of missing data values in $\mathcal{D}_M$ on which $\mathcal{D}^T$ and $\mathcal{D}_C$ agree. The accuracy of $\mathcal{D}_C$ is given by*

$$acc(\mathcal{D}_C) = \frac{n}{m}$$

Clearly, a 100% accurate completion of $\mathcal{D}_M$ will allow for unbiased estimates on $\mathcal{D}_C$. However, accuracy is a very strict measure. If $\mathcal{D}_C$ is simply a permutation of the rows of $\mathcal{D}^T$, the accuracy can be arbitrarily low. Whereas

such a permutation still allows for unbiased estimates. Still, accuracy is the most generally used quality measure for data completion [83].

Alternatively, we could define accuracy upto permutations. However, this would yield its computation rather hard. It would require the search for a permutation that yields maximal accuracy (in the strict sense as defined above).

To define a less strict quality measure, recall that the goal of a completion is to allow unbiased statistics. That is, statistics or models computed on $\mathcal{D}_C$ should be as close as possible to their counterparts computed on $\mathcal{D}^T$. Most statistical analysis of categorical data depends crucially on counts and sums. Often subtables are created using selections and projections, and counts and sums on these subtables are computed.

In the case of 0/1 data, selections correspond to itemsets; in fact we have the following simple result.

Theorem 6. *Let $\mathcal{D}$ be a complete database over $\mathcal{I}$, let $\mathcal{J}, \mathcal{K} \subseteq \mathcal{I}$, with $\mathcal{J} \cap \mathcal{K} = \emptyset$, and let $I \in \mathcal{I} \setminus (\mathcal{J} \cup \mathcal{K})$. The number of 1's I has in the subtable created by the selection*

$$\bigwedge_{J \in \mathcal{J}} J = 1 \ \wedge \bigwedge_{K \in \mathcal{K}} K = 0$$

on $\mathcal{D}$, is given by

$$supp_{\mathcal{D}}(\mathcal{J} \cup \{I\}) - supp_{\mathcal{D}}(\mathcal{J} \cup \mathcal{K} \cup \{I\})$$

Similarly, the number of 0's for I is given by

$$supp_{\mathcal{D}}(\mathcal{J}) - supp_{\mathcal{D}}(\mathcal{J} \cup \mathcal{K} \cup \{I\})$$

Proof. A transaction t satisfies the selection if it has 1's for all elements of $\mathcal{J}$ and 0's for all elements of $\mathcal{K}$. In other words, it should be in the support of $\mathcal{J}$, but not in the support of $\mathcal{K}$. □

Given that sums are counts on 1's on 0/1 databases and that the above theorem is invariant under suitable projections, we have the following corollary.

Corollary 7. *Let $\mathcal{D}_M, \mathcal{D}_C$, and $\mathcal{D}^T$ be databases over $\mathcal{I}$, such that $\mathcal{D}_M$ is incomplete, $\mathcal{D}^T$ is the true completion of $\mathcal{D}_M$ and $\mathcal{D}_C$ is an arbitrary completion of $\mathcal{D}_M$. If for all itemsets $X \subseteq \mathcal{I}$,*

$$supp_{\mathcal{D}_C}(X) = supp_{\mathcal{D}^T}(X)$$

then all counts and sums on project-select subtables on $\mathcal{D}_C$ equal their counterpart on $\mathcal{D}^T$.

With this result in mind, our new quality measure, we can measure how good the support of itemsets in a completed database is.

Definition 11. *Let $\mathcal{D}_M, \mathcal{D}_C$, and $\mathcal{D}^T$ be databases over $\mathcal{I}$, such that $\mathcal{D}_M$ is incomplete, $\mathcal{D}^T$ is the true completion of $\mathcal{D}_M$ and $\mathcal{D}_C$ is an arbitrary completion of $\mathcal{D}_M$. Moreover, let $\epsilon, \delta \in \mathbb{R}$. $\mathcal{D}_C$ is* (ϵ, δ)-correct *if for a random (frequent) itemset X*

$$P(\ |supp_{\mathcal{D}^T}(X) - supp_{\mathcal{D}_C}(X)| > \epsilon\ ) \leq \delta$$

In other words, the support of itemsets on (ϵ, δ)-correct completions are almost always close to the correct value. The lower ϵ and δ are, the better the completion is. As an aside, note that (ϵ, δ)-correctness is invariant under permutations; the sum is a commutative operator.

There is a simple relation between accuracy and (ϵ, δ)-correctness, as given by the following theorem.

Theorem 8. *Let $\mathcal{D}_M, \mathcal{D}_C$, and $\mathcal{D}^T$ be databases over $\mathcal{I}$, such that $\mathcal{D}_M$ is incomplete, $\mathcal{D}^T$ is the true completion of $\mathcal{D}_M$ and $\mathcal{D}_C$ is an arbitrary completion of $\mathcal{D}_M$. If $\mathcal{D}_C$ is $(0,0)$-correct, then there exists a permutation σ of the rows of $\mathcal{D}_C$ such that $\sigma(\mathcal{D}_C) = \mathcal{D}^T$*

Proof. Let ψ be a maximal injective partial function $\psi : \mathcal{D}^T \to \mathcal{D}_C$ such that $\psi(t) = t$. Moreover, let $s \in \mathcal{D}_C \setminus \psi(\mathcal{D}^T)$. Let the pair $(\mathcal{J}, \mathcal{K})$ be the partition of $\mathcal{I}$, such that s has value 1 for all items in $\mathcal{J}$ and value 0 for all items in $\mathcal{K}$. This means that s is in the support of $\mathcal{J}$ minus the support of $\mathcal{K}$. Since the support of all itemsets are equal on $\mathcal{D}_C$ and $\mathcal{D}^T$, this means that we can extend ψ to have s in its image. This contradicts maximality and, hence, $\mathcal{D}_C \setminus \psi(\mathcal{D}^T) = \emptyset$. Thus ψ is a bijection between $\mathcal{D}^T$ and $\mathcal{D}_C$. □

Clearly, if a completion is 100% accurate, it is also $(0,0)$-correct. If $(0,0)$-correctness is not attainable, the two measures differ. Due to the invariance of (ϵ, δ)-correctness, it is a more flexible quality measure.

(ϵ, δ)-correctness is defined for a random itemset, whatever the support of an itemset. Many of these itemsets will have a very low support, in fact, the vast majority will have support 0. For statistical analysis, however, item sets with a very low support are not very interesting. Statistics computed on small data sets, including the frequency of an itemset, are not very stable. Small pertubations to the database may cause large changes of these statistics. Hence, for the purposes of subsequent statistical analysis it is better to have a high (ϵ, δ)-correctness considering *frequent* itemsets only, rather than having a high (ϵ, δ)-correctness considering all itemsets.

All frequent itemsets is, unfortunately, still a large space to sample from. It is well-known that the set of all closed frequent item sets is often far smaller than the set of all frequent itemsets. Moreover, for any frequent itemset X, there is a closed frequent itemset Y such that the support of X equals the support of Y. Hence, if we know that for the closed frequent itemsets $supp_{\mathcal{D}^T}(Y) \approx supp_{\mathcal{D}_C}(Y)$, then this also holds for the frequent itemsets.

Given these observations, we sample the closed frequent itemsets to estimate (ϵ, δ)-correctness in our experiments.

The completion problem

With these quality measures at hand we formalise our completion problem as follows.

The Completion Problem:

> *Devise a completion algorithm $\mathcal{A}$ that yields an (ϵ, δ)-correct completion for any incomplete database with ϵ and δ as low as possible.*

We settle for *as low as possible* because there may not be enough information in the database to derive the $(0, 0)$-correct completion. For example, if $\mathcal{I}$ has only one item, the database has only one transaction and its value is missing. Either $\{1\}$ and $\{0\}$ are possible, and there is no algorithm that will reliably choose correctly. For all practical purposes, though, $(0, 0)$ is well approximable.

6.3 Completion Algorithms

Normally, we paraphrase MDL as *Induction by Compression*. Another way to paraphrase the MDL principle is: the better a model compresses the database, the closer it approximates the underlying data distribution. The key point of both MAR and MCAR is that they do not perturb this underlying distribution. Hence, in the spirit of MDL, one can say: the best completion is the completion that allows for the best compression.

A straightforward implementation of this idea, however, runs the risk of surpressing the natural variation in the data. The more data that is missing, the higher this risk becomes. How susceptible an algorithm is to this risk can be analysed by estimating (ϵ, δ)-correctness. The more susceptible an algorithm is, the worse the (ϵ, δ)-correctness will be.

Next, note that for $\mathcal{D}_M$ with n missing binary values the search space consists of 2^n possible completions. Clearly, finding the best completion quickly becomes infeasible, even for moderate amounts of missing values. Further, there is no structure that we can exploit to prune this search space. Therefore, we settle for heuristics that approximate the best compressable $\mathcal{D}_C$ by making local decisions.

We introduce three such completion algorithms, based on Krimp , in this section. For the first two algorithms, Simple Completion and Krimp Completion, we assume that there is enough complete data to allow Krimp to discover

good code tables. Moreover, Krimp Completion uses randomisation to minimise the risk of variation surpression. The third algorithm, called Krimp Minimisation, does no longer rely on the assumption that there is enough complete data.

Simple Completion

A simple way to impute a missing value is using the maximal likelihood estimator. For Krimp this reduces to shortest encoded length. Let $t \in \mathcal{D}_M$ be a transaction with missing values and denote by $C(t)$ the set of all its possible completions. The *Simple Completion* algorithm SC replaces t by that element of $C(t)$ that has the shortest encoded length.

More precisely, let $\mathcal{D}_M = \mathcal{D}_c \cup \mathcal{D}_m$ such that all transactions in $\mathcal{D}_c$ are complete, while all transactions in $\mathcal{D}_m$ are incomplete. The SC algorithm, Fig. 9, first computes a code table CT, by running Krimp on $\mathcal{D}_c$. Next, each incomplete transaction by its shortest completion.

Algorithm 9 The Simple Completion (SC) Algorithm

SC($\mathcal{D}_M$) :
1: $CT \leftarrow$ Krimp ($\mathcal{D}_c$)
2: **for** $t \in \mathcal{D}_m$ **do**
3: $\quad \arg\min_{s \in C(t)} L(s \mid CT)$
4: **end for**
5: **return** $\mathcal{D}_c \cup \mathcal{D}_m$

Krimp Completion

By always choosing the most likely candidate, the SC algorithm may have a detrimental effect on the support of some itemsets. Suppose, e.g. that the encoded length of $(1, 1)$ is slightly shorter than $(1, 0)$. Then each occurrence of $(1, ?)$ will be replaced by $(1, 1)$ by SC. This leads to an overestimate of the support of $(1, 1)$ and an underestimate of the support of $(1, 0)$. The Krimp *Completion* algorithm KC, see Fig. 10, remedies this by choosing an element of $C(t)$ with a chance proportional to its encoded length. More precisely, we again assume that $\mathcal{D}_M = \mathcal{D}_c \cup \mathcal{D}_m$ as before. Again Krimp is first run on $\mathcal{D}_c$. The resulting code table CT defines a probability distribution $P_{CT}(t)$ on $C(t)$ given by:

$$P_{CT}(t)(s) = \frac{2^{-L(s|CT)}}{\sum_{u \in C(t)} 2^{-L(u|CT)}}$$

The completion of t is chosen from $C(t)$ according to this distribution. The function CHOICE$(C(t), CT)$ makes this random choice.

Algorithm 10 The KRIMP COMPLETION (KC) Algorithm

KC$(\mathcal{D}_M)$:
1: $CT \leftarrow$ KRIMP $(\mathcal{D}_c)$
2: **for** $t \in \mathcal{D}_m$ **do**
3: $\quad t \leftarrow$ CHOICE $(C(t), CT)$
4: **end for**
5: **return** $\mathcal{D}_c \cup \mathcal{D}_m$

Krimp Minimisation

For KC to work, we need enough complete data for KRIMP to compute a good code table. If there is not enough complete data, the result of KC may be arbitrarily bad. To handle such a lack, we take an EM-like [37] approach.

The KRIMP *Minimisation* algorithm KM starts with a random completion of the incomplete database $\mathcal{D}_M$. Then it iterates through a number of KRIMP and KC steps. In the KRIMP step it compresses the current *complete* database. In the KC step it completes the *incomplete* database $\mathcal{D}_M$ using the code table computed in the KRIMP step. This is continued as long as the total encoded length of the completed database shrinks. The algorithm returns the final completed database. It has the shortest encoded length of the considered completions, hence the name of the algorithm.

Note that KM will always terminate. The total encoded size shrinks with every step. Since the encoded size of any finite database is finite, KM can only execute a finite number of iterations.

Algorithm 11 The KRIMP MINIMISATION (KM) Algorithm

KM$(\mathcal{D}_M)$:
1: $\mathcal{D}_C \leftarrow$ random completion of $\mathcal{D}_M$
2: **do**
3: $\quad CT \leftarrow$ KRIMP $(\mathcal{D}_C)$
4: $\quad \mathcal{D}_C \leftarrow$ KC $(\mathcal{D}_M)$
5: **while** *compressed size of* $\mathcal{D}_C$ *has not yet converged*
6: **return** $\mathcal{D}_C$

6.4 Related Work

Imputation has a long history. One of the first known examples, Hot Deck imputation [8], was employed by the US census bureau in the fifties. It replaces missing records by random draws from complete records from the same local area. As such, it may be regarded as a crude form of k nearest-neighbour imputation [132]. Since, more advanced systems for editing survey data have been developed, in particular for hierarchical demographic data. Examples include GEIS and SPEER [76] for continuous and DISCRETE [29] and CANCEIS [10] for discrete survey data. These systems all rely on nearest-neighbour algorithms for imputation [21]. As such, they require a distance function on the data, unlike parameter-free methods.

Regression, mean substitution and mean-mode [55] imputation have a greedy nature that harms the variance in the completed data [8]. Using some randomness circumvents this, which is why both Multiple Imputation (MI) [110] and Expectation Maximisation (EM) [37] are still the current state of the art.

To start with the latter, imputation through EM is the process of maximising the likelihood of the data given a distribution. Iteratively, it adapts the model to the data and re-imputes it. EM has been shown to provide very accurate probability estimations. Its model, however, has obviously to be chosen according to the data. For categorical data the log-linear model may be used. Still, by its exponential size in the number of attributes, this is only feasible for datasets with only few variables [112]. Krimp Minimisation follows a similar iterative approach. However, it optimises the compressed size of the database, not its likelihood.

Integrating a structure learner into the EM process leads to even better results. Structural EM (SEM) [44, 45] learns Bayes nets during the modeling phases. SEM has been shown to provide high quality probability estimates, and very good approximations of the original Bayes nets. However, it is computationally expensive and thereby only feasible for moderately sized datasets. A stark difference to our approach is that SEM learns a global Bayes network on the data, whereas Krimp considers the data in more detail by using local patterns.

Multiple imputation [110] states that the data should be imputed multiple times, thus providing different datasets. It does not dictate which data completion algorithm should be used, though typically either by sampling from predefined distributions [23] or by applying EM. The resulting datasets need to be analysed individually, after which the results are aggregated. For many data mining approaches this is non-trivial.

Table 6.1: Statistics of the datasets used in the experiments.

				Imputation accuracy (%)	
Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert$	*minsup* (%)	*random*	*baseline*
Alarm	5000	105	50.0	37.8	79.9
Chess (kr vs k)	28056	58	0.7	16.5	23.0
Led 7	3200	24	0	50.0	61.2
Letter recognition	20000	102	1.2	20.2	57.5
Mushroom	8124	119	1.2	28.8	57.6
Pen digits	10992	86	0.9	21.8	35.7
Tic–tac–toe	958	29	0	33.2	45.2
Wine	178	68	0	20.5	43.5

Shown are, per dataset, the number of transactions, the number of binary attributes and the relative *minsup* threshold used to mine frequent itemsets as KRIMP candidates. $minsup = 0$ indicates that *all* frequent itemsets, i.e. $supp \geq 1$, were used. Imputation accuracies were attained by randomly choosing an imputed value, as well as choosing the most frequent of the possible values.

6.5 Experiments

In this section we empirically evaluate the performance of the three proposed data completion methods. All results of the experiments in this section are averaged over 10 independent runs, unless indicated otherwise. Further, in these experiments we consider the case of 1 missing value per transaction, on average. Again, unless indicated otherwise. The (ϵ, δ)-correctness is calculated over the closed frequent itemset collection as mined on the complete test data.

Datasets

We use a range of data sets to validate our methods. From the widely used UCI repository [33] we took seven databases. Further, for fair comparison to the Bayesian Structural EM method, we generated data from the well-known *Alarm* network. This data was generated using the GenInstance program, made available by Nir Friedman in the LibB Bayes Network tool library [46].

The details for these data sets are depicted in Table 6.1. Apart from their base statistics, we provide the imputation accuracies on MCAR data as achieved by 1) choosing a random possible value and 2) the baseline estimator that chooses the most frequent of the possible values.

Table 6.2: Imputation quality given complete training data.

					MCAR			
	%		*baseline*		SC		KC	
Dataset	*miss. values*	ϵ	δ	*acc.*	δ	*acc.*	δ	*acc.*
Alarm	2.7	0.1	39.1	79.9	4.2	84.0	5.5	80.0
Chess	14.3	0.4	50.2	23.0	1.1	28.3	0.5	25.4
Led7	12.5	0.2	43.0	61.2	3.1	85.9	2.8	80.8
Letter	5.9	0.1	26.0	57.6	0.2	70.1	0.0	67.1
Mushroom	4.3	0.5	30.3	57.6	0.0	76.9	0.0	74.7
Pen digits	5.9	0.04	82.1	35.7	5.3	70.0	5.4	67.3
Tic-tac-toe	10.0	0.5	10.0	45.2	0.0	84.9	0.0	82.8
Wine	7.1	1.1	6.0	43.5	3.0	53.0	3.0	51.9
					MAR			
					SC		KC	
					δ	*acc.*	δ	*acc.*
Alarm					6.9	85.1	5.6	81.2
Chess					1.4	28.2	0.3	25.8
Led7					3.5	87.3	3.0	82.2
Letter					0.4	71.4	0.0	68.6
Mushroom					0.3	77.1	0.5	76.3
Pen digits					6.5	68.7	6.6	65.6
Tic-tac-toe					0.0	84.8	0.0	82.3
Wine					3.0	53.0	3.0	49.7

Missing values estimated using Simple Completion (SC) and Krimp Completion (KC), trained on complete data, using 10-fold cross-validation. For the (ϵ, δ)-measurements, ϵ was fixed, calculating δ per imputed database. All values relative (%).

We use the closed frequent pattern set as candidates for Krimp. Being generated from a Bayes net, the *Alarm* dataset contains no local structure. The absence hereof leads to a gigantic explosion in the number of patterns; hence we have to use a high minsup for this database.

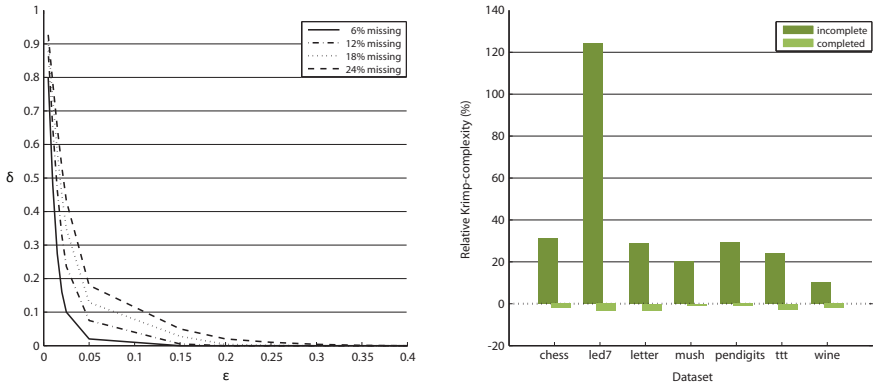


Figure 6.1: (left) (ϵ, δ)-correctness of Simple Completion on the *Letter Recognition* dataset. 10 fold cross-validated, trained on complete data. One missing value missing on average per transaction equals 5.9% missing data, 2 values 11.8%, 3 values 17.6% and 4 values 23.5%. ϵ in % (!). (right) Relative difference in Krimp complexity for incomplete and KM-completed data.

Creating missing values

We consider two types of missing data: missing completely at random (MCAR) and missing at random (MAR).

To create MCAR test data, we start with complete data. From it, for as many missing values we want to create, transactions are independently uniformly sampled. From each, one value is uniformly chosen and removed. For the MAR case, we use the class labels and the three-valued *bp* variable for *Alarm* as the depending attributes. For each of their values, a probability table was generated to indicate the chance of the other attributes being missing. We sample over this table to remove values, while still choosing the target transactions uniformly.

Sufficient complete data

First, we study how our methods perform provided with complete training data. Here, we thus consider Simple Completion and Krimp Completion. This experiment was set up using 10-fold cross validation. Leaving the training-data undamaged, we created missing values in the test–folds.

The results of this experiment are presented in Table 6.2. The accuracy scores of both our methods are much better than that of the baseline methods (scores for the random chooser in Table 6.1). This is even more the case for

correctness: for all these databases the random estimator scores a 100% chance of finding a marginal differing more than ϵ. Choosing the most frequent value is a better strategy, but still scores up to 50% chance of finding incorrect counts. Both SC and KC, however, do get very close to the optimal (0,0)–score.

Comparing between our two methods directly, we see that the greedy SC acquires better accuracy scores, as expected. However, this comes at a surprisingly slight cost on the (ϵ, δ)-correctness. Given sufficient complete training data even the greedy method maintains the proper local variance of the data. Both the SC and KC imputed data can be regarded statistically identical to the original. This goes for both MCAR and MAR, for which no strong differences in performance were found.

These experiments further show that there is no strong degradation of performance for increasing amounts of missing data. To show this in more detail, in the left hand side plot in Figure 6.1 we show a plot of the (ϵ, δ)-correctness of Simple Completion of the *Letter* dataset for 6 to 24% missing data. While the related accuracy scores only drop marginally (from 70.1 to 69.1), the attained (ϵ, δ)-correctness is even more impressive. Even with almost a quarter of the data missing, SC achieves (0.003, 0.01)-correctness.

Insufficient complete data

Second, we consider the problem for when no (sufficient) complete training data is available. We therefore now employ Krimp Minimisation instead of Krimp Completion.

The results of this set of experiments is presented in Table 6.3. If we first look at the accuracy measurements, we notice that the imputation accuracies are actually often higher than we saw just before. Through the (ϵ, δ)-correctness we can see that no magic is going on, as for all datasets these scores have actually decreased. Thus, through the incompleteness of the training data both methods are hindered in grasping the true data distribution.

Further, we see that the greedy nature of SC expectedly leads to a decrease in the data quality. For all datasets the accuracy provided by KM are only slightly lower than SC. This small loss in accuracy is compensated for by a strong gain in (ϵ, δ)-correctness. For KM, these scores are up to an order better than SC and often approximate the scores attained on the complete training data.

The right hand plot of Figure 6.1 shows further evidence of the data reconstruction ability of KM. To compress the data with missing values, Krimp typically requires 30% more bits than it does to encode the original data. Clearly, the missing values refrain it from encoding using the most appropriate patterns. However, through iterative imputation, KM is able to approximate the Krimp complexity of the original data within a single percent. As noise is canceled, the KM-imputed data has slightly lower complexity than the unseen

Table 6.3: Imputation quality measurements.

			MCAR				*MAR*			
			SC		KM		SC		KM	
Dataset	*% miss. values*	ϵ	δ	*acc.*	δ	*acc.*	δ	*acc.*	δ	*acc.*
Alarm	2.7	0.7	16.4	84.0	3.0	82.5	17.3	85.7	5.4	83.6
Chess	14.3	0.1	15.8	35.4	3.6	33.7	18.8	32.7	4.2	28.2
Led7	12.5	1.0	32.7	72.7	1.2	79.2	17.2	82.8	1.2	81.1
Letter	5.9	0.8	16.4	64.2	4.8	61.9	14.2	67.1	5.9	65.3
Mushroom	4.3	0.5	4.4	76.3	4.1	74.2	4.1	74.5	0.3	70.9
Pen digits	5.9	0.1	8.8	66.6	3.8	67.2	11.8	64.6	5.3	65.7
Tic–tac–toe	10.0	0.5	3.6	50.5	2.7	46.4	5.2	47.6	3.7	42.3
Wine	7.1	1.1	5.3	55.2	4.9	54.6	6.9	50.3	5.1	48.8

Missing values estimated using Simple Completion (SC) and Krimp Minimisation (KM), trained on incomplete data. For the (ϵ, δ)-measurements, ϵ was fixed, calculating δ per imputed database. All values relative (%).

original.

These experiments also showed that KM rapidly converges to this approximate original complexity: only three iterations were required for six of the datasets, two more for the data on *Mushroom* edibility. Further, we noticed that while KM is nondeterministic in initialisation and imputation, the resulting accuracies, correctness and data complexities are all virtually equal.

Comparing to SEM

Now that we have verified that KM provides good data completions, we will compare it to the state of the art in imputation: Bayes Structural EM (SEM) [45]. Structural EM incorporates the learning of a Bayes net on within the EM [37] process. Iteratively it is learned and used to re-impute the data, until the process converges.

In order to learn Bayes Nets on incomplete training data, we used Friedman's publicly available implementation of Structural EM [46]. Its search process can be initialised in various ways, here we used initialisation with random trees. Other settings were explored, but no significant differences in performance were found. For the actual inference on these Bayes nets for the missing values, we used the Bayes Network Toolbox for Matlab (BNT) [99]. As both learning the Bayes nets and inference are computationally expensive, we do not

Table 6.4: Imputation quality measurements for MCAR and MAR test data.

			MCAR				*MAR*			
	%		KM		SEM		KM		SEM	
Dataset	*miss. values*	ϵ	δ	*acc.*	δ	*acc.*	δ	*acc.*	δ	*acc.*
Alarm	2.7	0.5	6.7	82.5	15.2	80.9	8.0	83.6	37.8	82.5
Chess	14.3	0.1	4.1	33.7	24.5	23.5	4.2	28.2	33.2	22.7
Led 7	12.5	0.5	0.9	79.2	2.6	76.1	0.8	81.1	2.3	79.1
Tic–tac–toe	10.0	1.0	0.2	46.4	3.1	36.3	0.4	42.3	3.9	41.4
Wine	7.1	1.6	4.9	54.6	7.9	46.1	1.1	48.8	12.8	27.5

Missing values estimated using Krimp Minimisation (KM) and Structural EM (SEM), trained on incomplete data. For the (ϵ, δ)-measurements, ϵ was fixed, calculating δ per imputed database. All values relative (%).

consider all datasets in this comparison.

The results of this experiment are presented in Table 6.4. From it, we first notice that KM attains higher imputation accuracies than SEM for three out of the five datasets. However, the general quality of the KM imputed databases, as measured through the (ϵ, δ)-correctness scores, is quite dramatically better than the Bayes net driven SEM approach. This shows that the detail provided by the local-pattern based Krimp code tables allow for imputation that adheres much better to the local statistics of the original data than possible from a global model. Even for the *Alarm* dataset (with relatively few missing values), SEM is unable to score a win. Although this dataset contains no local structure, and we were forced to use high minsup values for Krimp, the resulting code tables still allow for better reconstruction of the original data than with the SEM induced global Bayes Net model.

Similar to the previous experiments, no strong trend presents itself when we compare between MCAR and MAR. For KM accuracy is harmed slightly on average, but its (ϵ, δ)-correctness remains stable or even improves. For SEM, however, we do see that for both *Alarm* and *Chess* the data quality does suffer significantly.

6.6 Discussion

The experimental results of our methods show that compression is a viable approach to the data completion problem. All three our parameter-free data

completion algorithms show that approximating the best compressible completed database leads to high quality imputation.

Provided with undamaged training data, SC provides highly accurate estimates. The method was shown to be robust: even up to 24% missing values, both accuracy and (ϵ, δ)-correctness of the completed data are very high.

For the realistic case of only insufficient complete training data being available, Krimp Minimisation is the right choice for a data completion algorithm. It finds good approximations of the best compressible completed database, and is shown to provide both provide high accuracy and very good (ϵ, δ)-scores. Further, the complexity of the original data is approximated within a single percent.

Compared to the state of the art in missing value estimation, Structural EM, the completions that KM offers provide both higher accuracy and adhere better to all count statistics of the original data.

Here we only consider the code tables from the Krimp compressor. As the proposed methods operate straightforwardly, it is possible to employ other compression schemes instead, for instance to better suit other data types.

6.7 Conclusions

In this chapter we considered the problem of high quality imputation of missing data. To test this objectively we propose (ϵ, δ)-correctness to measure the difference between two databases in terms of count statistics.

We presented three Krimp–based methods for imputation of incomplete datasets. All follow the MDL–principle: the completed database that can be compressed best is the best completed database. This, because then the imputations adhere to the local patterns that are present in the database, instead of only keeping its global statistics correct.

Both the greedy Simple Completion and randomised Krimp Completion offer high performance when sufficient complete training data is available. By minimising the compressed size of the imputed data, Krimp Minimisation performs evenly well when no complete data is available. Besides providing high accuracy, all three completion algorithms render imputations that particularly respect the variance of the original data.

Our methods consider local patterns in the data, rather than a global model; such local patterns are often smoothed out from a global stance. The experiments show that the local approach is indeed superior to the global approach, both in terms of accuracy and quality of the completed databases.

CHAPTER 7

Low-Entropy Set Selection

Most pattern discovery algorithms easily generate very large numbers of patterns, making the results impossible to understand and hard to use. Recently, the problem of instead selecting a small subset of informative patterns from a large collection of patterns has attracted a lot of interest. In this chapter we present a succinct way of representing data on the basis of itemsets that identify strong interactions.

This new approach, LESS, provides a more powerful and more general technique to data description than existing approaches. Low-entropy sets consider the data symmetrically and as such identify strong interactions between attributes, not just between items that are present. Selection of these patterns is executed through the MDL-criterion. This results in only a handful of sets that together form a compact lossless description of the data.

By using entropy-based elements for the data description, we can successfully apply the maximum likelihood principle to locally cover the data optimally. Further, it allows for a fast, natural and well performing heuristic. Based on these approaches we present two algorithms that provide high-quality descriptions of the data in terms of strongly interacting variables.

Experiments on these methods show that high-quality results are mined: very small pattern sets are returned that are easily interpretable and understandable descriptions of the data, and can be straightforwardly visualized. Swap randomization experiments and high compression ratios show that they capture the structure of the data well.[1]

[1] This work was originally published as [60]:
Heikinheimo, H., Vreeken, J., Siebes, A., Mannila, H. (2009) Low-Entropy Set Selection. In *Proceedings of the SDM'09*, pages 569-579.

7.1 Introduction

One of the central research themes in data mining has been the discovery of frequently occurring patterns. Starting from frequent sets and association rules [5], one of the key goals has been completeness in discovery: the task is to find all patterns from a pattern class that satisfy certain conditions. This goal is, in a way, a very useful one: from the answer we know exactly every pattern that fulfils the condition.

The drawback is that the number of patterns returned is typically prohibitively large. Generally, there are lots of patterns satisfying the conditions, but many patterns convey roughly the same information about the data.

Recently, several authors have studied the pattern selection problem: given a large set of patterns, find a small subset of informative patterns. Examples of such work are [20,69,95,113,135]. These proposals all manage to reduce the pattern explosion significantly and achieve massive reductions in the number of patterns. However, whether these describe the data in full, or only partly, has a strong influence on the number of selected patterns: respectively up to hundreds, or only tens.

Lossy approaches, due to the small number of resulting patterns, allow for very easy interpretation. However, they cannot explain the data in full detail and may overlook important and interesting interactions. Lossless approaches, on the other hand, typically result into slightly more patterns. While this improved level of detail allows for thorough data analysis, interpreting or inspecting these groups of patterns by hand can be more difficult.

In this chapter we provide a lossless method for succinct description of datasets using low-entropy itemsets. Our approach obtains very small collections of informative patterns, typically in the order of tens of patterns, that are both readable and provide intuitive descriptions of the data. The method is inspired by two recent approaches: low-entropy sets [59] and the MDL-based method Krimp (see Chapter 2).

Informally, a low-entropy set is a set X of variables such that the distribution of the data on these variables is highly skewed, i.e., has low entropy. For example, consider a set $X = \{A,B,C,D,E\}$ of binary variables. Assume further that the data has 1000 rows where the values of these variables are $(1,0,0,1,0)$, and 2000 for which the values are $(1,0,1,0,0)$, and that the frequencies of the remaining 30 value combinations (2^5 in total) are all negligibly small. Together these variables interact strongly, i.e. their values are strongly structured, and resultantly the entropy of the dataset on the variables in X is small: for a single row of the data we can code the values of the variables in X using only few bits on average (about 0.9, in this case).

Low-entropy sets can be viewed as a stark generalization of frequent itemsets, which just look for sets X such that there are sufficiently many rows that have a 1 in each column of X. Unlike frequent itemsets, low-entropy sets are

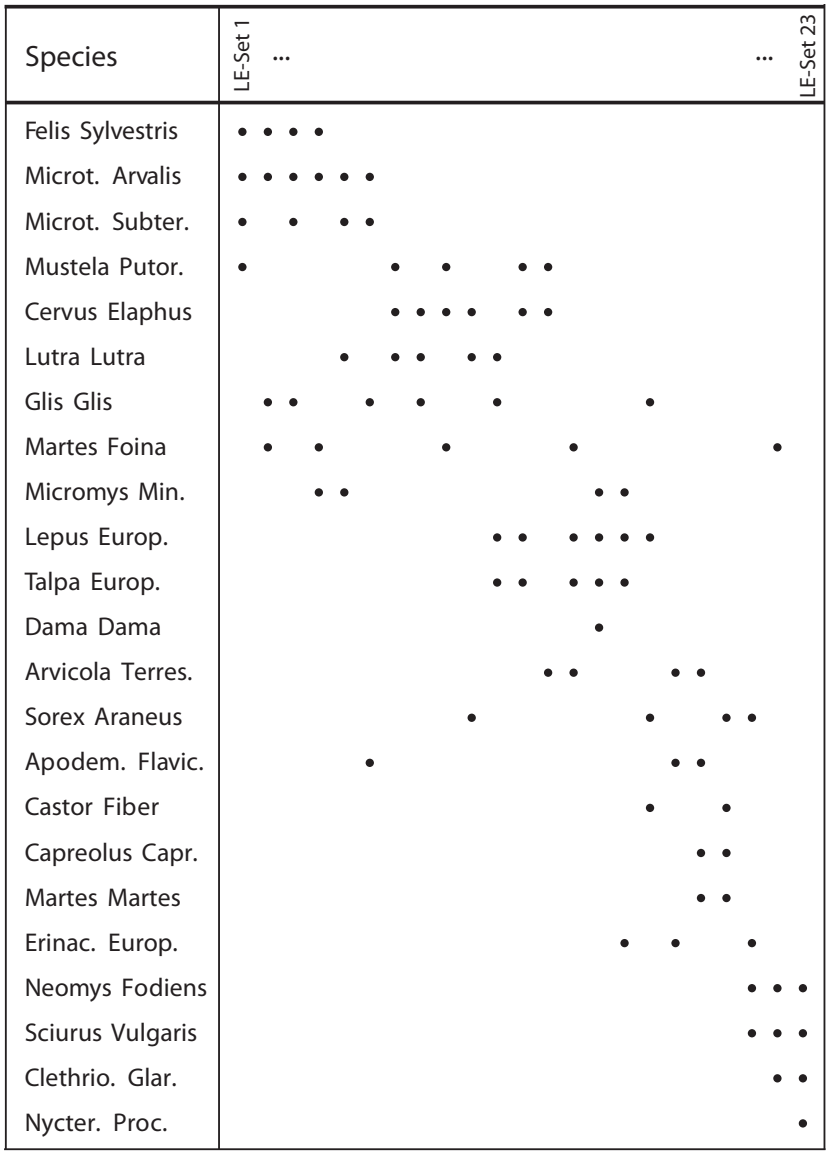


Figure 7.1: Visualization of 23 sets (columns) that our method selected to describe the occurrence interactions of 23 mammal species.

symmetric with respect to 0 and 1. As above, they can locate subsets that have just a few different dominant values. Therefore, they are very applicable for analyzing dense data. However, as with frequent sets, the number of low-entropy patterns can grow prohibitively large: for higher levels of entropy, many more sets are found than is practical for analysis by hand.

The MDL-approach (see Chapter 2) to selecting pattern subsets is based on the idea that the best subset of patterns is the one that compresses the data best. It identifies the best collection of itemsets as the one that requires the fewest bits to describe all of the data. For frequent itemsets, a transaction can be described simply by telling which itemsets together (i.e. their union) form the transaction.

For low-entropy sets this is less straightforward; to (re)construct a data row, we have to identify both the low-entropy sets and their specific variable values. For example, if a data row t would happen to have the combination $(1, 0, 1, 0, 0)$ on the variables of X, to describe t we can say that low-entropy set X is to be used, with $(1, 0, 1, 0, 0)$ as its value combination; as this combination is so frequent, it can be encoded in only a few bits. However, opposed to frequent itemsets, low-entropy sets can be used to describe *any* transaction: there always is one instantiation that fits the row. We are therefore required to use a fine-grained selection to determine which low-entropy set will be used to encode what part of the data. Turning this to our advantage, we provide two methods based on the maximum likelihood principle to optimally cover the data locally.

Using these principled selection strategies, we employ the MDL criterion to encode the whole data succinctly using low-entropy sets. As such, the method requires only tens of patterns for a detailed lossless description of the full data. Consequently, the outcome can be interpreted very easily.

As an example, consider Figure 7.1. It visualizes the results of our method on a dataset concerning the geographical presence of mammal species. In this picture, we show the low-entropy sets (the columns) that our method selected to describe the interactions between the 23 species. These sets were selected by our method out of the 67677 low-entropy patterns of entropy ≤ 3.3 bits. Indeed, it is a compact set of species interactions that together well describe the main essence of the data.

The reason why the data can be characterized by such a small number of patterns is the fact that one low-entropy set may capture multiple interactions at once for the same attribute set. Consider for instance the left–most column of Figure 7.1, showing an interaction pattern between two Vole species (*M. Arvalis* and *M. Subterraneus*) together with the predators Wild cat (*F. Sylvestris*) and European Polecat (*M. Putor.*). Table 7.1 takes a closer look at the usage counts of the individual variable combinations of the set as they are used to describe the data. As the table suggests, relevant interactions involve several occurrence

Table 7.1: Detailed view of how LESS uses a low-entropy set.

F. Sylvestris	M. Arvalis	M. Suberraneus	M. Putor.	counts
0	0	0	0	44
0	0	0	1	10
0	1	0	0	18
0	1	0	1	199
0	1	1	1	248
1	0	0	0	5
1	0	0	1	7
total # usage				531

Detailed view of how LESS uses the left-most low-entropy set of Figure 7.1 to encode the data. The set depicts major presence-interactions of four mammal species.

combinations, as well as absence of the species. Hence, if the same interactions would have to be described using regular itemsets many separate sets would be required, instead of the single low-entropy set required here.

In this study we provide the methodological and algorithmic solutions necessary to use the MDL framework for low-entropy set patterns. Experiments show that the end result yields easily interpretable small collections of low-entropy sets. The quality of the mined pattern groups is first verified through compression. By swap randomization experiments we affirm that these sets grasp the significant structure in the data. Further, we provide evidence that these sets together describe multiple distributions by comparing them to the cluster centroids of the data. In summary, the results show that our method only requires as few patterns as lossy methods do to provide a high-quality lossless description of the data.

The roadmap of this chapter is as follows. First we introduce some preliminaries on low-entropy sets and how MDL can be used to select the most interesting subset of low-entropy sets. In Section 7.3 we present the LESS algorithm for Low Entropy Set Selection, as well as a principled way of encoding the data locally optimally. Next, in Section 7.4 we empirically evaluate the proposed method. Related studies are discussed in Section 7.5. We round up with discussion and conclusions.

7.2 Problem Definition

In this section we introduce preliminaries and notations used in subsequent sections. As we regard the data 0/1 symmetric in this chapter, the notation used here differs strongly from previous chapters.

Low-entropy sets

Let $\mathcal{I}$ be a set of 0–1 valued attributes. A *transaction* t over $\mathcal{I}$ is a binary vector of length $|\mathcal{I}|$. A dataset $\mathcal{D}$ is simply a bag of transactions, the number of which is denoted by $|\mathcal{D}|$. We denote *attribute sets*, i.e., subsets of $\mathcal{I}$, by X and Y. For singleton sets we omit the braces, e.g., we write A instead of $\{A\}$.

We use $\pi_A(t)$ to refer to the value of attribute A on row t (1 or 0). Given an attribute set X, we denote by $\pi_X(t)$ the projection of the transaction t onto X. In other words, $\pi_X(t)$ is a 0–1 vector of values $\pi_A(t)$ defined by the attributes $A \in X$.

Let Ω_X be the set $\{0,1\}^{|X|}$ of all 0-1 vectors of length $|X|$. We call the vectors $i \in \Omega_X$ the *instantiations* of the attribute set X. We say that the instantiation i fits transaction t iff $i = \pi_X(t)$. The probability $p_X(i)$ of an instantiation i is the relative support in $\mathcal{D}$ of the attributes X having the value of i. More formally,

$$p_X(i) = \frac{|\{t \in \mathcal{D} \mid i = \pi_X(t)\}|}{|\mathcal{D}|}.$$

Or, simply put, the fraction of transactions in $\mathcal{D}$ where i fits. For readability, we write $p(i)$ wherever X is clear from the context.

The entropy of an attribute set X in $\mathcal{D}$ is

$$H(X) = -\sum_{i \in \Omega_X} p(i) \log_2 p(i),$$

where $0 \log_2 0$ is assigned the value of 0 by convention.

Entropy is a measure of skewness in the occurrence distribution of instantiations of X. The lower the entropy, the more structured and more concentrated the instantiations in the database are. From a pattern mining point of view, attribute sets exhibitting structure in the form of low entropy can therefore be considered interesting.

Definition 12. *Given an entropy threshold ϵ, an attribute set X is a low-entropy set (LE-set) in $\mathcal{D}$ if $H(X) \leq \epsilon$.*

It is straightforward to show that low-entropy sets have a monotonicity property. Say we combine attributes A and B into a set X. By definition, $H(X)$ is minimal iff X has no instantiations of lower probability than A or B

separately. In other words, no value combination of A and B is more surprising than any of the value combinations of A or B separately. This is only the case if A and B are either exact copies or exact negatives, resulting in $H(X) = H(A) = H(B)$. Otherwise, if A and B disagree on one or more values we have $H(X) \geq H(A)$ and $H(X) \geq H(B)$. For the low-entropy mining task this monotonicity property allows to use e.g. a level-wise search in similar fashion to that of frequent items [5]. For a formal proof, and more details on mining low-entropy sets, see [59].

MDL for low-entropy sets

Like stated in Chapter 2, one can summarize the MDL approach to induction by the slogan: *the best model compresses the data best.* For a short introduction to the MDL principle, see Chapter 2, or refer to [52].

Constructing a compression scheme that is based on LE-sets is not trivial. Unlike for itemsets, see Chapter 2, unambiguous decoding is impossible if only the sets are encoded: we also have to identify which individual instantiations are used. Here, we want to describe the data primarily in terms of low-entropy sets, and are less interested in their value instantiations. Therefore those value identifying codes should provide as little as possible bias to which LE-sets are chosen, while at the same time the complexity of the model should be weighed properly. This, we reach by encoding the sets and the instantiations separately. Most importantly, we make the code lengths for the instantiations independent from those of the LE-sets.

The basic idea is as follows: to describe a transaction t, we tell which LE-sets and which instantiations are used to obtain the values $\pi_A(t)$ for each attribute A. During compression a *code table* is induced, a two-column table containing a list of LE-sets and the codes used to identify them. The codes come from a prefix-code to allow for unambiguous decoding. The more often a set is used to encode the transactions in the database, the shorter its associated code.

Example 9. *Given a transaction t we code it by giving a sequence of LE-sets and instantiations for these. As a simple example, consider the attributes $\{ABCD\}$ and a transaction $t = \{A, D\}$ (i.e., the vector $(1, 0, 0, 1)$). This transaction can be described by the LE-sets $\{A, B\}$ and $\{C, D\}$ with instantiations $(1, 0)$ and $(0, 1)$ respectively. Let the code associated with $\{A, B\}$ be c_1 and let c_2 be the code associated with $\{C, D\}$. The naïve way to store this would be c_1c_2 for the coded transaction and $((1, 0), (0, 1))$ as the indication of the values. This last part is simply a representation of t as a binary vector. Although a possible encoding, as it completely ignores any structure it is hardly a way to compress.*

So, we have to refine the encoding of the instantiations. Note that $\{(0, 0), (0, 1), (1, 0), (1, 1)\}$ is the set of instantiations of all two element LE-sets. Hence, one

of these will be used whenever a two element LE-set is part of the cover of a transaction. Therefore we assign a (prefix) code to each of these instantiations; again, the more often an instantiation is used, the shorter its code. The codes for instantiations are called indicators. Continuing the example, let l_1 *be the code associated with* $(1, 0)$ *and* l_2 *the one with* $(0, 1)$*. The transaction* $\{A, D\}$ *is then encoded by the pair of codes* c_1c_2 *and* l_1l_2*.*

From the example follows that to compress transactions we require two code tables. The first, the LE-set code table denoted by CT_{LE}, is defined as a two-column table of which the first column contains LE-sets and the second contains their codes. Second, the indicator code table, denoted by CT_I, is analogously defined as a two-column table of which the first column contains indicators and the second column the associated codes. The encoding of a database $\mathcal{D}$ with this pair of code tables results in a pair of encodings. The first, $\mathcal{D}_{LE}$, contains the codes from CT_{LE}, the second, $\mathcal{D}_I$ contains the codes from CT_I.

As done in Chapter 2, we also require CT_{LE} to contain at least the singleton attribute sets, such that all possible transactions can be encoded using any valid code table. Similarly, if the largest LE-set in CT_{LE} contains n attributes, CT_I is defined to contain all possible indicators for one-element LE-sets up to those for n-element LE-sets. That is, CT_I will have $2^{n+1} - 1$ entries.

Coding the transactions

To determine the appropriate code for the elements of both CT_{LE} and CT_I we need to know how often an LE-set and its instantiations are used. That is, we have to define which elements of CT_{LE} are used to cover a transaction t and which instantiations of those elements are used.

Informally, a *cover function* provides a set of non-overlapping LE-sets such that they describe all attributes $\mathcal{I}$ of a transaction t of database $\mathcal{D}$. We formalize this as follows:

Definition 13. *A* cover *function is a function that given a LE-set code table* CT_{LE} *and an indicator code table* CT_I *assigns to each transaction* t *a list of pairs*

$$(X, i) \;\; X \in CT_{LE}, i \in CT_I$$

such that the union of the instantiated LE-sets equals t*. Slightly abusing notation, we write both* $X \in cover(CT, t)$ *and* $i \in cover(CT, t)$ *whenever* $(X, i) \in cover(CT, t)$*.*

Since the CT_{LE} elements in the result of a cover function are non-overlapping, *cover* only needs to return a list of CT_{LE} elements. The associated indicators can easily be reconstructed by considering this list and the transaction.

The number of times a CT_{LE} element X is used in the cover of a transaction $t \in \mathcal{D}$ is called its *usage* frequency. The usage frequency of an indicator $i \in CT_I$ is defined similarly:

$$\begin{aligned} usage_{CT}(X) &= |\{t \in \mathcal{D} \mid X \in cover(CT, t)\}| \\ usage_{CT}(i) &= |\{t \in \mathcal{D} \mid i \in cover(CT, t)\}| \end{aligned}$$

The probability that X or i is used in the cover of a randomly selected transaction t is thus

$$\begin{aligned} P(X \mid \mathcal{D}) &= \frac{usage_{CT}(X)}{\sum_{Y \in CT} usage_{CT}(Y)}, \\ P(i \mid \mathcal{D}) &= \frac{usage_{CT}(i)}{\sum_{j \in CT} usage_{CT}(j)}. \end{aligned}$$

To compress the database optimally, we use the Shannon code [53] for both code tables. That means that the length (in bits) of the codes for X and i is

$$\begin{aligned} L_{\mathcal{D}}(code(X)) &= -\log(P(X \mid \mathcal{D})), \\ L_{\mathcal{D}}(code(i)) &= -\log(P(i \mid \mathcal{D})). \end{aligned}$$

Note that we are only interested in the lengths of these codes, not the actual codes themselves. Then, we can calculate the size of the encoded databases $\mathcal{D}_{LE}$ and $\mathcal{D}_I$, encoded respectively by CT_{LE} and CT_I, as

$$\begin{aligned} L(\mathcal{D}_{LE} \mid CT_{LE}) &= \sum_{X \in CT_{LE}} -usage_{CT}(X) \log(P(X \mid \mathcal{D})), \\ L(\mathcal{D}_I \mid CT_I) &= \sum_{i \in CT_I} -usage_{CT}(i) \log(P(i \mid \mathcal{D})). \end{aligned}$$

The encoded size of the full database then is

$$L(\mathcal{D} \mid CT) = L(\mathcal{D}_{LE} \mid CT_{LE}) + L(\mathcal{D}_I \mid CT_I).$$

For the two code tables, we already know the size of the codes, viz. $L_{\mathcal{D}}(code(X))$ and $L_{\mathcal{D}}(code(i))$ as defined above. To compute, respectively, the sizes of the LE-sets and the indicators these codes stand for, we have to define how we encode them.

For CT_{LE}, we encode the LE-sets by what we define as the *standard* code table ST, which is the simplest valid code table: the code table that only

contains the singleton attribute sets. Hence, the size of CT_{LE} is (ignoring elements $X \in CT_{LE}$ with $usage_{CT}(X) = 0$)

$$L_{\mathcal{D}}(CT_{LE}) = \sum_{X \in CT_{LE}} L_{\mathcal{D}}(code(X)) + L_{\mathcal{D}}(code_{ST}(X)).$$

For CT_I, we simply use the bit-representation of the instantiations. That is, the instantiation $(0, 1)$ is represented by 01. We denote the bit-representation of $i \in CT_I$ by $bit(i)$. Note that if the largest LE-set in CT_{LE} has n items, then

$$\sum_{i \in CT_I} bit(i) = \sum_{j=1} nj2^j = 2 + (n-1)2^{n+1}.$$

Hence, the size of CT_I is

$$L_{\mathcal{D}}(CT_I) = \sum_{i \in CT_I} L_{\mathcal{D}}(code(i)) + bit(i).$$

Then, we have as the total size for the code tables,

$$L_{\mathcal{D}}(CT) = L_{\mathcal{D}}(CT_{LE}) + L_{\mathcal{D}}(CT_I).$$

The formal problem statement

Now that all the details of MDL for LE-sets have been defined, we can formally state our problem.

> *Let $\mathcal{D}$ be a transaction database, and* cover *a cover function. Find the code tables CT_{LE} and CT_I minimizing the* total encoded size
>
> $$L(\mathcal{D}, CT) = L(\mathcal{D} \mid CT) + L_{\mathcal{D}}(CT).$$

So, the actual problem is now to find the best code table. Note that given CT_{LE} and a *cover* function determining CT_I is trivial.

Still, the search space we have to consider for this problem is huge. First, it consists of all possible code tables CT_{LE}: all possible subsets of $\mathcal{P}(\mathcal{I})$ that contain at least the singleton sets $\mathcal{I}$. So, there are

$$\sum_{k=0}^{2^{|\mathcal{I}|}-|\mathcal{I}|-1} \binom{2^{|\mathcal{I}|} - |\mathcal{I}| - 1}{k}$$

possible code tables. In order to determine which one minimizes the total encoded size, we have to consider these using every possible cover function.

This translates to using every possible cover order per transaction. Since there are $n!$ possible orders for a set of length n, the total size of the search space is

$$\sum_{k=0}^{2^{|\mathcal{I}|}-|\mathcal{I}|-1} \left(\binom{2^{|\mathcal{I}|} - |\mathcal{I}| - 1}{k} \times (k + |\mathcal{I}|)! \times |\mathcal{D}| \right).$$

In short, it is prohibitively large. To make matters worse, there is no useable structure that allows us to prune this search space. Hence, we need to use heuristics.

7.3 Algorithms

Our approach for finding the best possible code table can be divided into two main important elements:

- *transaction encoding phase*, where a good compression for each transaction (cover) is found using the patterns in the code table.
- *the search strategy*, which is the way in which the search space of all possible pattern subsets is traversed to find a good code table.

The method follows the general framework of Chapter 2. However, we apply very different technical solutions within the different parts of the approach. We will discuss transaction encoding in the next Subsection and the search strategy afterwards.

Transaction encoding

As discussed in Section 7.2, when encoding the database the task is to compress the data as well as possible using only the LE-sets in the code table. This is done by encoding each transaction with a set of patterns from the code table. However, as pointed out in that section, an LE-set always has an instantiation that can be used to describe a transaction. Therefore, there are often many different ways (depending on the order in which we use sets from the code table) that a transaction can be covered, and hence compressed.

Our strategy to select good covers for each transaction is to take advantage of the statistical nature of low-entropy patterns and use the maximum likelihood (ML) principle. The idea is to take the cover C that maximizes the conditional probability $p(t|C)$ (likelihood) of the transaction given all possible covers. We define the likelihood of transaction t as follows:

Definition 14. *Let t be a transaction and C a cover of t.*

$$llh(t, C) = \sum_{X \in C} \log p(\pi_X(t)) \tag{7.1}$$

is the log–likelihood of t given C.

For each transaction t, the task is then to find the optimal cover

$$C^* = \arg\max_C llh(t, C)$$

from the set of all possible covers, such that llh is maximized.

Maximum likelihood is a widely used and well principled way of selecting between alternative models for the data. That is, in our case to choose between different covers of a transactions. Moreover, the ML-principle has an intuitive connection to the overall task of minimizing the total encoded length of the data.

In more detail, the connection is as follows. First, let us consider a set of LE-sets that form a cover C of some transaction t. In order for C to provide a good compression of both this transaction and the whole of the data, the associated codes should be as short as possible. Hence, both the LE-sets $X \in C$ and the instantiations $\pi_X(t)$ should be used as often as possible. Now, let's assume that for all the rest of the transactions in the data the instantiation $\pi_X(t)$ and LE-set $X \in C$ are used to cover every transaction where $\pi_X(t)$ fits, and that the instantiation $\pi_X(t)$ is not used anywhere else. That is, more formally for the case of the instantiation,

$$usage_{CT}(\pi_X(t)) = |\mathcal{D}| \cdot p(\pi_X(t)),$$

where $p(\pi_X(t))$ is the frequency of the transactions for which $\pi_X(t)$ fits. From this it follows that the code length of $\pi_X(t)$ will be strictly proportional[2] to its negative log–likelihood. More formally,

$$L(code(\pi_X(t))) \propto -log\frac{usage_{CT}(\pi_X(t))}{|\mathcal{D}|} = -\log p(\pi_X(t)).$$

Therefore, optimizing equation 7.1 also optimizes the entire coding length of transaction t. Assuming this for every transaction in the database, the encoding size of the entire database will be proportional to the negative log–likelihood of the data.

The above assumption will not strictly hold in every case. However, in general patterns with instantiations of high likelihood (high support) are likely to behave approximately like this and will consequently optimize the encoded length of the data well.

[2]Notice that in Section 7.2 the probability $P(i|\mathcal{D})$ is normalized with the usage frequencies of all of the elements in CT_I instead of the size of the data $|\mathcal{D}|$. Hence, the code length is proportionally but not exactly the same as the negative log–likelihood.

Finding the Best Cover

It is quite clear that finding an optimal cover for a transaction is NP-complete [47] and hence exact solutions cannot be computed for large datasets. However, many covering problems are known to be well approximable using greedy heuristics.

In this subsection, we first study covering a transaction optimally according to the maximal likelihood principle, with an exhaustive search strategy using pruning. In the next subsection, we discuss a greedy heuristic that can be applied for larger datasets.

To compute the optimal cover for transaction t, we start with a code table that is ordered on the per attribute likelihood addition. The idea is that given a transaction t we assign to LE-set $X \in CT_{LE}$ a weight $w(t, X)$ that is equal to the per attribute addition in likelihood that X would give, if it were to be added to the cover. More formally

$$w(t, X) = \log\big(p(\pi_X(t))\big)/|X|. \tag{7.2}$$

Given this ordered code table, we need to enumerate all possible covers in a depth first manner. We start from LE-set X with the largest weight $w(t, X)$ and greedily continue to add, non-overlapping, sets to the cover in decreasing order; backtracking the search at each time when reaching a full cover.

By taking this order into account, we can cut the search space down considerably using the following proposition.

Proposition 10. *Consider covering transaction t with disjunct attribute sets in strictly decreasing order according to the weight function w. Now, when at the* i*th attribute set, already having covered attributes Y with cover C_Y, we know that the resulting cover will have an apriori coding length of at most*

$$llh(t, C) \leq m \cdot w(t, X_i) + llh(t, C_Y), \tag{7.3}$$

where m is the number of previously uncovered attributes.

The proposition follows straightforwardly from the fact that, if X_i covers all the previously uncovered attributes without overlapping Y, the likelihood of the resulting covering will be exactly the right hand side of inequality 7.3. Otherwise, we'll have to include some other set, which by the ordering on w will provide an equal or smaller addition in likelihood per attribute. Hence, this will result in a smaller overall likelihood for the transaction. Thus, Proposition 10 defines an upper bound that we can compare to the best found solution so far; and thus to decide whether it makes sense to continue building the current cover or to start backtracking already.

Written in pseudo code, this optimal covering approach is depicted in Algorithm 12. However, as the optimal covering method considers a prohibitively

Algorithm 12 The OptimalCover Algorithm

OptimalCover($\mathcal{I}, CT, t$) :
1: OrderOnLikelihoodPerAttribute(CT, t)
2: **return** Optimal($\mathcal{I}, CT, \emptyset, \emptyset$)

Optimal($\mathcal{I}, CT, C, BestC$) :
4: **if** $|C| \leftarrow |\mathcal{I}|$ **then**
5: **return** C
6: **end if**
7: $\Delta \leftarrow \emptyset$
8: $m \leftarrow |\mathcal{I}| - |C|$
9: **for** $e \in CT$ **do**
10: $\Delta \leftarrow e \cup \Delta$
11: **if** $e \cap C \leftarrow \emptyset$ **then**
12: **if** $m \cdot w(e) + llh(C) > llh(BestC)$ **then**
13: $CandC \leftarrow$ Optimal($CT \setminus \Delta, C \cup e, BestC$)
14: $BestC \leftarrow \arg\min_{C} (llh(BestC), llh(CandC))$
15: **end if**
16: **end if**
17: **end for**
18: **return** $BestC$

large search space, it only makes sense to apply it to moderately sized databases of up to about 25 attributes. To allow for more practical application, we present a fast, heuristic alternative that follows very naturally from the minimum apriori encoding length/maximum likelihood principle.

Approximating the Best Cover

Recall that our goal is to cover using elements that provide as high a gain as possible in the overall likelihood. The initial order used by the optimal algorithm provides us a good approximation, as it orders the elements on the gain in likelihood per attribute. If we use this order in a greedy fashion (without overlap), the resulting cover is the same as the first full cover the optimal cover strategy considers.

We present, as Algorithm 13, the translation of this simple scheme into pseudo code. Note that for an actual implementation a lot of speed can be gained as one can easily cache the per-transaction orders. As code tables remain very small this is fully feasible.

Algorithm 13 The GreedyCover Algorithm

GreedyCover$(\mathcal{I}, CT, t)$:

```
1: Cover ← ∅
2: for X ∈ CT, in order of gain in likelihood per attribute for t, do
3:   if X ∩ Cover = ∅ then
4:     Cover ← X ∪ Cover
5:     if |Cover| = |I| then
6:       return Cover
7:     end if
8:   end if
9: end for
```

Search strategy

To cut down a large part of the search space, we use the following simple greedy search strategy:

- Start with the code table consisting only of the singleton attribute sets
- Add the low-entropy sets one by one. If the resulting codes lead to a better compression, keep it. Otherwise, discard the set.

By its iterative nature, the success of this strategy largely depends on the order in which the patterns are considered.

Ordering the Candidate Sets

Using the strategy above, the optimal compression can be approximated best by trying all possible orders. However, as the number of possible orders of a set of size n equals $n!$ this clearly is infeasible for but the smallest of pattern collections. So, our last step to reduce the search space of our problem heuristically is to introduce an order on the candidate set $\mathcal{F}$. We order the candidates such that sets that have a good chance of being used – those with high likelihood addition over size – are at the top of the list. On the candidate set level, this translates into preferring sets that have a low entropy over size, or $H(X)/|X|$.

The Low-Entropy Set Selection algorithm

Now the main ingredients for our low-entropy set based compression algorithm are in place, we can assemble these into the Low-Entropy Set Selection (LESS for short) algorithm. We present it in pseudo–code as Algorithm 14.

As input, it requires the attribute set $\mathcal{I}$, the candidate set of low-entropy sets $\mathcal{F}$, and a database $\mathcal{D}$. Also, one of the above discussed cover strategies for

Algorithm 14 The LESS Algorithm

LESS($\mathcal{I}, \mathcal{F}, \mathcal{D}$) :

1: $CT \leftarrow$ **Standard Code Table**($\mathcal{I}, \max |F \in \mathcal{F}|$)
2: **for** $F \in \mathcal{F}_o \setminus \mathcal{I}$ in **Candidate Order do**
3: $CT_c \leftarrow (CT \cup \{F\})$
4: **if** $L(CT_c, \mathcal{D}) < L(CT, \mathcal{D})$ **then**
5: $CT \leftarrow CT_c$
6: **end if**
7: **end for**
8: **return** CT

transaction encoding has to be chosen. Our naive compression process starts with the simplest description of the data, using only singletons to encode the data, together with the fully initialized indicator code table. Then, iteratively (in **Candidate Order**) low-entropy sets are added to the code table one by one. Each time, using this new code table the new total compressed size of the database is calculated. If this addition improves the attained compression, the set is kept, otherwise it is permanently discarded.

In the course of this iterative process it is very well possible that by adding a new element, the usage of other patterns in the code table suddenly strongly decreases; thereby increasing their code lengths and possibly hindering overall compression. We therefore introduce a pruning variant of our method. Once an element $F \in \mathcal{F}$ is accepted into the code table, we reconsider all other elements $X \in CT_{LE}$ iteratively by temporarily removing them and calculating the compressed size. By MDL-principle, we then go for the best compression, permanently removing those elements that no longer help the compression.

7.4 Experiments

In this section we experimentally evaluate our methods. We first investigate the differences between OptimalCover and GreedyCover. Next, we evaluate whether our method models relevant structure of the data. Thirdly, we look at the size of the resulting pattern sets and compare these to two other existing methods. Last, we examine these pattern sets in detail.

Datasets

For the experimental validation of our methods we use a wide range of datasets. From the widely used UCI repository [33] we take some of the largest and most dense databases. Further, we use two databases for which we know low-entropy analysis is well suited: the *Mammals* and *Courses* databases. The former con-

Table 7.2: Statistics of the datasets used in the experiments.

Dataset	$\|\mathcal{I}\|$	$\|\mathcal{D}\|$	*density*	$L(\mathcal{D} \mid ST)$
Adult	97	48842	15.3	34229566
Course	83	2405	20.5	1422594
Heart	50	303	28.0	134588
Letter recognition	102	20000	16.7	14954124
Mammals	40	2183	47.0	552457
Mammals_{20}	20	2183	53.1	248665
Mushroom	119	8124	19.3	7898102
Pen digits	86	10992	19.8	6757243

Per dataset the number of attributes, the number of transactions, the density (percentage of 1's) and the number of bits required by LESS to compress the data using the singleton-only standard code table ST.

sists of presence/absence records of European mammals[3] within geographical areas of 50x50 kilometers [96]. The *Courses* data describes courses taken by students at the Department of Computer Science of the University of Helsinki. As we want to focus on interesting variable interactions, we disregard attributes with extremely high (about 1.0) or very low (about 0.0) support.

The details for these datasets are depicted in Table 7.2. For each database we show the number of attributes, the number of rows and the density: the percentage of 'present' attributes. The next column indicates the total compressed size in bits by using the singletons – only standard code table ST.

Due to the high density most of these datasets, they are not well suited for analysis by frequent itemset mining: far too many co-occurences exist. For example, at 10% support already over 11 million frequent itemsets are discovered in the *Mammals* dataset and up to 5.5 billion can be extracted from the *Mushroom* data.

When mining for low-entropy sets, those itemsets of which the attributes are too weakly correlated (i.e. their entropy is above the threshold) are ignored. In order to compress the data optimally, we have to allow LESS to consider as many low-entropy sets as possible. We therefore set the entropy threshold ϵ as low as feasible with our current low-entropy set mining implementation.

[3]The full version of the *Mammals* dataset is available for research purposes upon request from the Societas Europaea Mammalogica.
`http://www.european-mammals.org`

Optimal and greedy covering

To compare the performance between the optimal and the greedy covering strategies (Algorithms 12 and 13), we use $Mammals_{20}$, which constains the 20 most varying attributes of the full *Mammals* dataset. On this data we mine low-entropy sets using a maximum entropy threshold of $3\frac{1}{3}$ bits, resulting in 2321 low-entropy sets.

For each transaction we compute a cover using both GreedyCover and OptimalCover on all of the 2321 sets as the code table, as well as a baseline cover of only singleton sets. The results for each single transaction are presented in Figure 7.2. First of all, we see that using sets pays off both for the optimal and the greedy methods with an increase in log–likelihood for each transaction. Moreover, we see that the optimal transaction cover does indeed result in the highest log–likelihood scores. However, the much faster greedy approach finds covers that approximate the optimal score closely.

Next, we test these strategies with LESS using the LE-sets as candidates, but without pruning. In two hours, the optimal variant selected 25 sets to compress the data into 184572 bits. In one minute, the greedy approach finds a description of only 163908 bits, using 148 sets. By using a larger number of sets, it attains a higher likelihood over the data (-22091 and -19767, respectively). Analyzing the resulting code tables, it is evident that the optimal method is more picky and less promiscuous: if a set is allowed into the code table, it will stay in use and will not be fully traded in, opposed to what the greedy method does. However, the resulting code tables are very similar to those of the greedy algorithm when pruning is *enabled*. The greedy approach seems to concatenate the 'optimal' sets together into 12 sets to achieve a likelihood of -22330, while using only 145940 bits. Overall, the greedy cover algorithm allows MDL to condense the data better. When considered together, this tells us that GreedyCover can be used as a fast and high quality alternative for OptimalCover. For the remainder of this section, we'll therefore use GreedyCover.

Modeling relevant structure

To evaluate whether LESS models the relevant structure in the data, we compare the compression scores of the actual data to those obtained from 1000 swap randomized [49] versions of that data. This process preserves the row and column margins of the given data set, but obscures the internal dependencies of the data. The idea is that if the true structure in the data is captured, there should be significant differences between the models found on the original and randomized datasets.

For this large number of experiments we used the $Mammals_{20}$ dataset. We applied as many swaps as there are 1's in the data. Figure 7.3 shows the

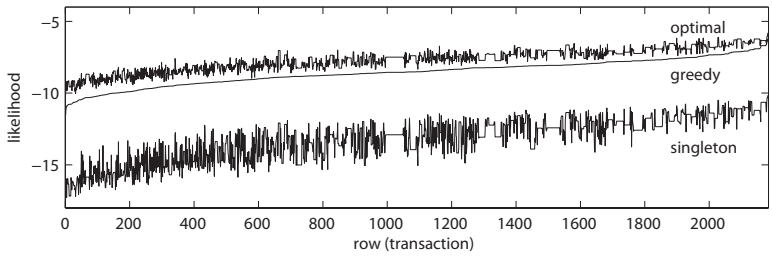


Figure 7.2: The likelihood scores for each transaction in the $Mammals_{20}$ dataset using the OptimalCover Algorithm, the GreedyCover Algorithm and singleton (all attributes independent) covering. The transactions have been sorted according to the likelihood score of GreedyCover.

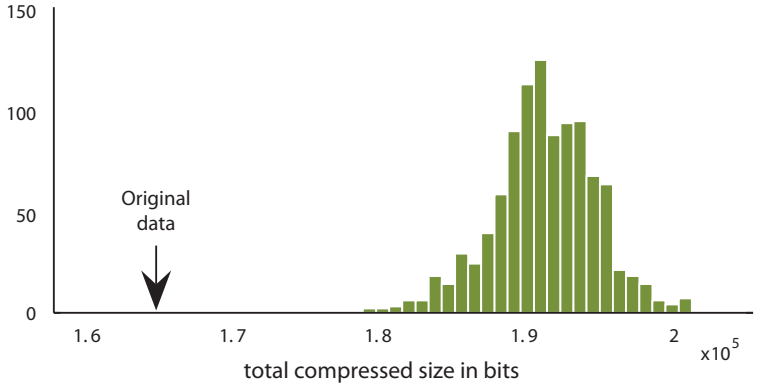


Figure 7.3: Distribution of the compressed sizes of the swap randomized $Mammals_{20}$ datasets. The compressed size of the original dataset as indicated by the arrow, and is 1.64×10^5 bits.

histogram of the compressed sizes of these 1000 databases.

The picture shows that the original data can be compressed significantly better than that of the randomized datasets (p-value of 0). Also, we noted that the log–likelihood score was much higher for the real data, even though we directly optimize the compression and not this score. Further, analyzing the contents of the code tables, we also note a significant difference in set cardinality. For the real data, the average set was 2.35 attributes long, while for the randomized data we see elements with an average length of 1.89.

Table 7.3: Results of LESS, using the GreedyCover algorithm.

	Candidates		LESS				Krimp	
Dataset	ϵ	$\lvert\mathcal{F}\rvert$	*prune*	$\lvert CT\rvert$	$L(CT,\mathcal{D})$	$L\%$	*ms*	$\lvert CT\rvert$
Adult	2.9	143766	no	**153**	27149706	79.3	1	1941
			yes	**10**	26678350	77.9		1303
Courses	2.8	455709	no	**918**	1003730	70.6	100	551
			yes	**28**	877284	61.7		285
Heart	3.3	414589	no	**115**	117343	87.2	1	108
			yes	**49**	115539	85.8		79
Letter	3.3	368889	no	**838**	11375860	76.1	50	3395
			yes	**21**	10547561	70.5		1259
Mammals	3.8	250628	no	**488**	359116	65.0	200	536
			yes	**30**	314932	57.0		254
Mushroom	2.8	437239	no	**241**	5802484	73.5	1	689
			yes	**10**	5474484	69.3		424
Pen digits	2.5	71994	no	**160**	4088429	60.5	50	2667
			yes	**28**	3778077	55.9		1091

Results of LESS using the GreedyCover algorithm for a variety of datasets, with comparison to the frequent itemset based method Krimp. Shown are the threshold ϵ (in bits) for mining low-entropy sets and the number of sets discovered. For LESS, for both pruning disabled and enabled, the number of LE-sets selected into the code table, the total compressed size of the data and the achieved compression ratio. For Krimp the minimal support threshold *ms* for mining frequent itemsets and the number of selected itemsets without and with pruning, resp.

Reduction and improvement

In Table 7.3 we present the quantified results of running LESS using the GreedyCover algorithm. The main outcome this table shows is the large reduction in number of low-entropy sets that the algorithm attains. Even for relatively low entropy thresholds, up to 5 orders of magnitude fewer sets are selected. At the same time, the small compressed sizes of the databases show the quality of these descriptions.

We also see that enabling pruning has a strong effect on the number of selected sets: roughly an order of magnitude. Inspection of the code tables shows that the two strategies provide slightly different views on the data. Without

pruning, the likelihood maximization process selects more specific sets. Consequently, we see that of the selected sets typically only a few (one or two) very characteristic instantiations find major use. With pruning enabled the process is forced to select more general patterns. This effect is clearly illustrated by the much smaller number of returned sets, of which now multiple instantiations are used often. The much better compression scores show that pruning results in better data descriptions.

Next, we compare the number of patterns our method returns to two other data description methods. First, we compare to KRIMP, which uses frequent itemsets for a lossless compression. Table 7.3 shows that our method requires far fewer patterns, even although these do describe *all* interactions in the data, instead of just the 1's. It also illustrates that our method is well suited to deal with dense databases, for which the differences grow even larger. For example, at a *minsup* of 10% already over 11 million frequent itemsets are mined on *Mammals*. Entropy recognises the structure in the data and returns a fraction of this amount in low-entropy sets.

Second, we compare our scores to those of the lossy method proposed by Bringmann and Zimmermann [20] on the largest dataset they considered: *Mushroom*. Depending on the selection criterium, their approach returns 21 to 71 itemsets to describe only part of the dataset. Our method, on the other hand, requires only 10 LE-sets to provide a detailed lossless description of the data.

The runtimes of the experiments ranged from one minute up to ten hours. Analysis shows that the runtime is mainly dependent on the number of transactions and particularly the size of the code tables. Hence, the time required for the experiments where pruning keeps the code tables small are in the order of few minutes up to one hour – typically 45 minutes. The experiments with pruning disabled (where the code tables are allowed to grow to hundreds of elements) typically took up to three hours, with an exception for *Mushroom* of ten hours.

Examining the code tables

The code tables produced by LESS are small enough for human analysis, for instance through visualization. Figure 7.1 in Section 7.1 provides a good example of such a visualization. Each column in the table presents a LE-set with bullets marking the species (rows) that are included in the set. Moreover, as Table 7.1 shows, one can also zoom in further and investigate the interaction combinations between the variables in full detail.

We observe in Figure 7.1 that some attributes are included in more than just one set in the code table. At first this looks like redundancy in the description. However, computing a centroid vector for each LE-set in the code table according to the rows it covers and comparing these to the centroids found

from the data by the k-means algorithm [88] we notice that the overlapping sets are mostly used in different clusters in the data. For instance the LE-sets {*F. Sylvestris*, *M. Arvalis*, *G. Glis*, *M. Foina*} and {*F. Sylvestris*, *M. Subter.*, *G. Glis*, *M. Foina*} are associated to different clusters. Thus, these mappings show that the overlapping sets are not redundant but tuned to describe specific parts of the data.

7.5 Related Work

Lately, the pattern explosion problem has attracted a lot of research. For frequent pattern mining, lossless methods such as closed [103] and non-derivable [25] itemsets were proposed to remove the redundancy within the pattern set. However, the attained reduction deteriorates heavily under noise. Methods that provide a lossy representation of the complete pattern set include maximal itemsets [12]. Yan et al. [135] proposed a method that selects k representative patterns that together summarize the pattern set well.

Low-entropy sets [59] are a more expressive, entropy-based, generalization of frequent patterns. These allow for more thorough data analysis and reduce the pattern explosion at the same time. However, at high entropy levels the pattern set may still grow prohibitively large. Other related information-theoretic pattern definitions include [70, 98] as well as work on correlated pattern mining [66].

Recently, the approach of finding small subsets of informative patterns has attracted a significant amount of research [20, 69, 95, 113]. Pattern Teams [69] are groups of k non-redundant patterns that have been exhaustively (k<10) optimized according to criterions such as joint entropy. Bringmann et al. [20] proposed a greedy variant that can consider larger (100's) pattern sets. Either method is lossy, in the sense that it finds pattern sets that cover only part of the data.

Alternatively, pattern sets can be selected to describe the data best, which falls naturally in the compression approach to data mining [42]. Recently, we introduced [113] the MDL based itemset selection algorithm Krimp (see Chapter 2). Although we follow a similar selection approach, the generality and applicability of the methods is rather different. By considering data 0/1–symmetric we can capture all major interactions between attributes, not just co-occurences. Partly thanks to this generalization, LESS yields in the order of tens of patterns, opposed to hundreds to thousands for Krimp. Through these much smaller numbers inspection by hand is now possible. Also, these pattern sets have a different meaning, as they view the data in terms of strongly interacting variables; not just present items.

Further, the technical solutions we propose are more general. Instead of using ad-hoc order heuristics to determine which patterns describe what part

of the data, we introduce a principled way of finding locally optimal covers of the data through the maximum likelihood principle. By using two separate encodings, one identifying the pattern and the other its value instantiation, our framework is more generally applicable. For instance, a promising future research direction would be to expand it to other pattern selection settings where the patterns lack a one-to-one mapping to a specific value, like selecting the most interesting subgroups identified by SQL queries.

7.6 Discussion

Our novel combination of compression and entropy finds very short, high-quality descriptions of the data. As these descriptions are easily visualized, they can easily be interpreted by humans. They show what goes on in the data, on two levels of detail: an overview of the strongly interacting variables in general, and specifying in detail what are the most prominent interactions.

By basing our cover strategies on the maximum likelihood principle, we have a very natural approach to only use instantiations to describe data where this makes sense. Consequently, the code tables capture the significant structure in the data, as the swap-randomization experiments show.

Reconsidering older code table elements once a new LE-set has been admitted increases the quality of the data description even further. Although our method needs to compress the data for each candidate, the measured running times show this approach to be realistic for analysis of large and dense datasets in particular. The candidate set is determined by the max entropy parameter, which may be set as high as is feasible for mining, or makes sense from an analysis point of view. Further, besides the decision of whether or not to prune, there are no parameters: MDL selects the best code table.

LESS combines the best of the lossless and lossy approaches to data description; the number of returned patterns is comparable to the latter, while at the same time our pattern sets do provide a lossless description of the data. Further, these patterns consider both 0's and 1's.

Even though our current implementation is unpolished, the recorded running times show the method can already realistically be applied for data analysis. However, many possible optimizations are available. One of the most promising would be to just calculate the change the current candidate implies to the previously found best optimum; opposed to calculating a full database cover every time. Speedup on the subset matching could be gained by using a true bitmap representation of the database and the instantiations. Thirdly, paralellization can easily be applied to LESS, both in respect to distributing parts of the database, as well as considering of the candidates distributedly. Using either, or all, these optimizations LESS would become even more applicable for analysis of very large dense databases.

7.7 Conclusions

We presented LESS, a method for selecting very small collections of highly descriptive low-entropy sets through compression. The small size of these collections facilitates thorough analysis by experts. The interpretability of low-entropy sets makes this analysis even easier. By using entropy instead of frequency, it is particularly suited for mining dense datasets. Further, by regarding data 0/1 symmetric, LESS captures all major interactions in the data, not just co-occurences.

Clearly, entropy is not just defined for binary data, but also for other types, such as real-valued data. Hence, the generalization of this work to such other types of data would make for a both useful and challenging future research direction. Another promising direction would be to apply our framework to other pattern types that lack one-to-one value associations, such as for instance the queries used in subgroup discovery.

CHAPTER 8

Finding Good Itemsets by Packing Data

The problem of selecting small groups of itemsets that represent the data well has recently gained a lot of attention. We approach the problem by searching for the itemsets that compress the data efficiently. As a compression technique we use decision trees combined with a refined version of MDL. More formally, assuming that the items are ordered, we create a decision tree for each item that may only depend on the previous items. Our approach allows us to find complex interactions between the attributes, not just co-occurrences of 1s. Further, we present a link between the itemsets and the decision trees and use this link to export the itemsets from the decision trees. In this chapter we present two algorithms. The first one is a simple greedy approach that builds a family of itemsets directly from data. The second one, given a collection of candidate itemsets, selects a small subset of these itemsets. Our experiments show that these approaches result in compact and high quality descriptions of the data.[1]

[1] This work was originally published as [118]:
Tatti, N., Vreeken, J. (2008) Finding Good Itemsets by Packing Data. In *Proceedings of the ICDM'08*, pages 588-597.

8.1 Introduction

One of the major topics in data mining research is the discovery of interesting patterns in data. From the introduction of frequent itemset mining and association rules [5], the pattern explosion was acknowledged: at high frequency thresholds only common knowledge is revealed, while at low thresholds prohibitively many patterns are returned.

Part of this problem can be solved by reducing these collections either lossless or lossy, however even then the resulting collections are often so large that they cannot be analyzed by hand or even machine. Recently, it was therefore argued [54] that while the efficiency of the search process has received ample attention, there still exists a strong need for pattern mining approaches that deliver compact, yet high quality, collections of patterns (see Section 8.6 for a more detailed discussion). Our goal is to identify the family of itemsets that form the best description of the data. Recent proposals to this end all consider just part of the data, by either only considering co-occurrences (i.e. Krimp, see Chapter 2) or being lossy in nature [20,69,127]. In this chapter, we present two methods that do describe all interactions in the data. Although different in approach, both methods return small families of itemsets, which are selected to provide high-quality lossless descriptions of the data in terms of local patterns. Importantly, our parameterless methods regard the data symmetrically. That is, we consider not just the 1s in the data, but also the 0s. Therefore, we are able to find patterns that describe *all* interactions between items in the data, not just co-occurrences.

As a measure of quality for the collection of itemsets we employ the practical variant of Kolmogorov Complexity [81], the Minimum Description Length (MDL) principle [53]. This principle implies that we should do induction through compression. It states that the best model is the model that provides the best compression of the data: it is the model that captures best the regularities of the data, with as little redundancy as possible.

The main idea of our approach is to use decision trees to determine the shortest possible encoding of an attribute, by using the values of already transmitted attributes. For example, let us assume two binary attributes A and B. Now say that for 90% of the time when the attribute A has a value of 1, the attribute B has a value of 0. If this situation occurs frequently, we recognize this dependency, and include the item A in the tree deciding how to encode B.

Using such trees allows us to find complex interactions between the items while at the same time MDL provides us with a parameter-free framework for removing fake interactions that are due to the noise in the data. The main outcome of our methods is not the decision trees, but the group of itemsets that form their paths: these are the important patterns in the data since they capture the dependencies between the attributes implied by the decision trees.

The two algorithms we introduce to this end are orthogonal in approach.

Our first method builds the encoding decision trees directly from the data; it greedily introduces splits until no split can help to compress the data further. Just as naturally as we can extract itemsets from these trees, we can consider the trees that can be built from a collection of itemsets. That link is exploited by our second method, which tries to select the best itemsets from a larger collection.

Experimental evaluation shows that both methods return small collections of itemsets that provide high quality data descriptions. These sets allow for very short encoding of the data, which inherently shows that the most important patterns in the data are captured. As the number of itemsets are small, we can easily expose the resulting itemsets to further analysis, either by hand or by machine.

The rest of this chapter is as follows. After the covering preliminaries in Section 8.2, we discuss how to use decision trees to optimally encode the data succinct in Section 8.3. Next, in Section 8.4, we explain the connection between decision trees and itemsets. Section 8.5 introduces our method with which good itemsets can be selected by weighing these through our decision tree encoding. Related work is discussed in Section 8.6, after which we present the experiments on our methods in Section 8.7. We round up with discussion and conclusions in Sections 8.8 and 8.9.

8.2 Preliminaries

In this chapter we introduce a starkly different way of encoding data and selecting itemsets, for which we here introduce preliminaries and notations used in subsequent sections.

A *binary dataset* $\mathcal{D}$ is a collection of $|\mathcal{D}|$ *transactions*, binary vectors of length K. The ith element of a random transaction is represented by an *attribute* a_i, a Bernoulli random variable. We denote the collection of all the attributes by $A = \{a_1, \ldots, a_K\}$. An *itemset* $X = \{x_1, \ldots, x_L\} \subseteq A$ is a subset of attributes. We will often use the dense notation $X = x_1 \cdots x_L$.

Given an itemset X and a binary vector v of length L, we use the notation $p(X = v)$ to express the probability of $p(x_1 = v_1, \ldots, x_L = v_L)$. If v contains only 1s, then we will use the notation $p(X = 1)$, if v contains only 0s, then we will use the notation $p(X = 0)$.

Given a binary dataset $\mathcal{D}$ we define $q_{\mathcal{D}}$ to be an *empirical distribution*,

$$q_{\mathcal{D}}(A = v) = |\{t \in \mathcal{D} \mid t = v\}|/|\mathcal{D}|.$$

We define the frequency of an itemset X to be $fr(X) = q_{\mathcal{D}}(X = 1)$.

In this chapter we use the common convention $0 \log 0 = 0$ and all logarithms are of base 2.

In the subsequent sections we will need some knowledge of graphs. All the graphs in this chapter are directed. Given a graph G we denote by $V(G)$ the set of vertices and by $E(G)$ the edges of G. A directed graph is said to be *acyclic* (DAG) if there is no cycle in the graph. A directed graph is said to be *directed spanning tree* if each node (except one special node) has exactly one outgoing edge. The special node has no outgoing edge and is called *sink*.

8.3 Packing Binary Data with Decision Trees

In this section we present our model for packing the data and a greedy algorithm for searching good models.

The model

Our goal in this section is to define a model that is used to transmit a binary dataset D from a transmitter to a receiver. We do this by transmitting one transaction at the time, the order of which does not matter. Within a single transaction we transmit the items one at the time.

Assume that we are transmitting an attribute a_t. As the attribute may have two values, we need to have two codes to indicate its value. We define the table in which these two codes are stored to be a *coding table*. Obviously, the codes need to be optimal, that is, as short as possible. From information theory [35], we have the optimal Shannon codes of length $-\log(p(x))$. Here, the optimal code lengths are thus $-\log q_{\mathcal{D}}(a_t = 1)$ and $-\log q_{\mathcal{D}}(a_t = 0)$. We need to transmit the attribute $|\mathcal{D}|$ times. The cost of these transmissions is

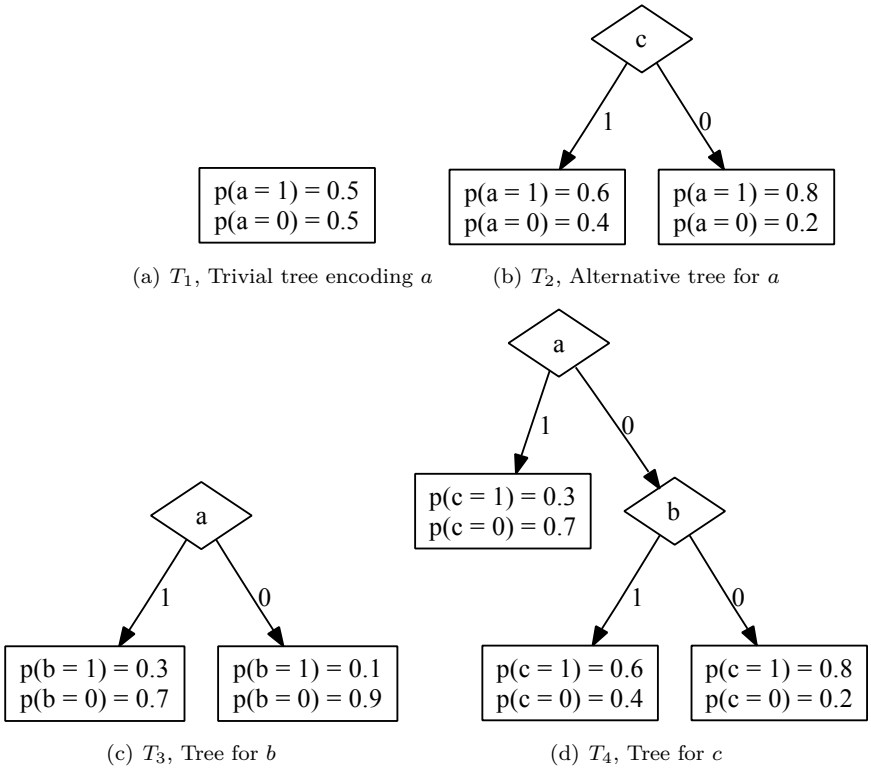
$$-|\mathcal{D}| \sum_{v=\{0,1\}} q_{\mathcal{D}}(a_t = v) \log q_{\mathcal{D}}(a_t = v)\,.$$

This is the simplest case of encoding a_t. Note that we are not interested in the actual codes, but only in their lengths: they allow us to determine the complexity of a model.

A more complex and more interesting approach to encode a_t succinct is to have several coding tables from which the transmitter chooses one for transmission. Choosing the coding table is done via a decision tree that branches on the values of other attributes in the same transaction. That is, we have a decision tree used for encoding a_t in which each leaf node is associated with a different coding table of a_t. The leaf is selected by testing the values of other attributes within the same transaction.

Example 11. *Assume that we have three attributes, a, b, and c and consider the trees given in Figure 8.1. In Figure 8.1(a) we have the simplest tree, a simple coding table with no dependencies at all. A more complex tree is given*

in Figure 8.1(b) where the transmitter chooses from two coding table for a based on the value of c. Similarly in, Figure 8.1(d) we have three different coding tables for c. The choice of the coding table in this case is based on the values of a and b.



(a) T_1, Trivial tree encoding a (b) T_2, Alternative tree for a

(c) T_3, Tree for b (d) T_4, Tree for c

Figure 8.1: Toy decision trees.

Let us introduce some notation. Let T be a tree encoding a_t. We use the notation $t(T) = a_t$. We set $src(T)$ to be the set of all items used in T for choosing the coding table.

Example 12. *For the tree T_3 in Figure 8.1(c) we have $t(T_3) = b$ and $src(T_3) = \{a\}$ and for T_4 in Figure 8.1(d) we have $t(T_4)$ and $src(T_4) = \{a, b\}$.*

To define the cost of transmitting a_t we first define $lvs(T)$ to be the set of all leaves in T. Let $L \in lvs(T)$ be a leaf and $q_{\mathcal{D}}(L)$ be the probability of L being chosen. Further, $q_{\mathcal{D}}(a_t = v \mid L)$ is the probability of $a_t = v$ given that L

is chosen. We now know that the optimal cost, denoted by $c_D(T)$, is

$$-|\mathcal{D}| \sum_{L \in lvs(T)} \sum_{v=\{0,1\}} q_{\mathcal{D}}(a_t = v, L) \log q_{\mathcal{D}}(a_t = v \mid L)\,.$$

Example 13. *The number of bits needed by T_1 in Figure 8.1(a) to transmit a in a random transaction is*

$$-0.5 \log 0.5 - 0.5 \log 0.5 = 1.$$

Similarly, if we assume that $q_{\mathcal{D}}(a = 1) = q_{\mathcal{D}}(a = 0) = 0.5$, the number of bits needed by T_3 to transmit c in a random transaction is

$$\begin{aligned} &0.5\,(-0.3 \log 0.3 - 0.7 \log 0.7) + \\ &\quad 0.5\,(-0.1 \log 0.1 - 0.9 \log 0.9) = 0.62. \end{aligned}$$

In order for the receiver to decode the attribute a_t he must know what coding table was used. Thus, he must be able to use the same decision tree that the transmitter used for encoding a_t. To ensure this, the transmitter must know $src(T)$ when decoding a_t. So, the attributes must have an order in which they are sent *and* the decision trees may only use the attributes that have already been transmitted.

The aforementioned requirement is easily characterized by the following construction. Let G be a directed graph with K nodes, each node corresponding to an attribute. The graph G contains all the edges of form (a_t, a_s) where $a_s \in src(T)$, where T is the tree encoding a_t. We call G the *dependency graph*. It is easy to see that there exists an order of the attributes if and only if G is an acyclic graph (DAG). If G constructed from a set of trees $\mathcal{T} = \{T_1, \ldots, T_K\}$ is indeed DAG we call the set $\mathcal{T}$ a *decision tree model*.

Example 14. *Consider a graph given in Figure 8.2(a) constructed from the trees T_2, T_3, and T_4 (Figure 8.1). We cannot use this combination of trees for encoding since there is a cycle in the graph. On the other hand if we use trees T_1, T_3, and T_4, then the resulting graph (given in Figure 8.2(b)) is acyclic and thus these trees can be used for the transmission.*

Encoding data

In order for the receiver to be able to decode the attributes, he must know both the coding tables and the trees. Hence, we need to transmit both of these. First, we cover how the coding tables, the leafs of the decision trees, are transmitted.

To transmit the coding tables we use the concept of Refined MDL [53]. Refined MDL is an improved version of the more traditional two-part MDL

(a) Dependency graph with cycles (b) Dependency acyclic graph

Figure 8.2: Dependency graphs constructed from the trees given in Figure 8.1.

(sometimes referred to as crude MDL). The basic idea of the refined variant is that instead of transmitting the coding tables, the transmitter and the receiver use so called universal codes. Universal codes are the cornerstone of Refined MDL. As these are codes can be derived without any further shared information, this allows for a good weighing of the actual complexity of the data and model, with virtually no overhead. While the practicality of applying such codes depends on the type of the model, our decision trees are particularly well-suited.

These universal codes provide a cost called the *complexity* of the model. This cost can be calculated as follows: let L be a leaf in the decision tree (i.e. coding table), and M be the number of transactions for which L is used. Then the complexity of this leaf, denoted by $c_{\mathrm{MDL}}(L)$, is

$$c_{\mathrm{MDL}}(L) = \log \sum_{k=0}^{M} \binom{M}{k} \left(\frac{k}{M}\right)^{k} \left(\frac{M-k}{M}\right)^{M-k} .$$

In general, there is no known closed formula for the complexity of the model. Hence estimates are usually employed [109]. However, for our tree models we can apply an existing linear-time algorithm that solves the complexity for multinomial models [72]. We should also point out that the Refined MDL is asymptotically equivalent to Bayes Information Criteria (BIC) if the number of transactions goes to infinity and the number of free parameters stays fixed. However, for moderate numbers of transactions there may be significant differences [53].

Now that the coding tables can be transmitted, we need to know how to transmit the actual tree T. To encode the tree we simply transmit the nodes of the tree in a sequence. We use one bit to indicate whether the node is a leaf, or an intermediate node $N \in \mathit{intr}(T)$. For an intermediate node we additionally use $\log K$ bits, where K is the number of attributes in $\mathcal{D}$, to indicate the item that is used for the split.

The combined cost of a tree T, denoted by $c(T)$, is

$$c(T) = \sum_{N \in intr(T)} \big(1 + \log K\big) + c_D(T) + \sum_{L \in lvs(T)} \big(1 + c_{\text{MDL}}(L)\big),$$

that is, the cost $c(T)$ is the number of bits needed to transmit the tree *and* the attribute a_t in each transaction of $\mathcal{D}$.

Example 15. *Assume that we have a dataset with* 100 *transactions and* 3 *items. Assume also that* $q_{\mathcal{D}}(a = 0) = q_{\mathcal{D}}(a = 1) = 0.5$. *We know that the complexity of the leaves in this case is* $c_{\text{MDL}}(L) = 3.25$. *The cost of the tree* T_3 *(Figure 8.1(c) is*

$$\begin{aligned} c(T_3) =& 1 + \log 3 \\ &+ 1 + 3.25 + 50\,(-0.3 \log 0.3 - 0.7 \log 0.7) \\ &+ 1 + 3.25 + 50\,(-0.1 \log 0.1 - 0.9 \log 0.9) \\ =& 69.8. \end{aligned}$$

Given a decision tree model $\mathcal{T} = \{T_1, \ldots, T_K\}$ we define the cost $c(\mathcal{T}) = \sum_i c(T_i)$. The cost $c(\mathcal{T})$ is the number of bits needed to transmit the trees, one for each attribute, and the complete dataset $\mathcal{D}$.

We should point out that for data with many items, the term $\log K$ grows and hence the threshold increases for selecting an attribute into any decision tree. This is an interesting behavior, as due to the finite number of transactions, for datasets with many items there is an increased probability that two items will correlate, even though they are independent according to the generative distribution.

Greedy algorithm

Our goal is to find the decision tree model with the lowest complexity cost. However, since many problems related to the decision trees are **NP**-complete [100] we will resort to a greedy heuristic to approximate the decision tree model $\mathcal{T}$ with the lowest $c(\mathcal{T})$. It is based on the ID3 algorithm.

To fully introduce the algorithm we need some notation: By TrivialTree(a_t) we mean the simplest tree packing a_t without any other attributes (see Figure 8.1(a)). Given a tree T, a leaf $L \in lvs(T)$, and an item c not occurring in the path from L to the root of T, we define SplitTree(T, L, c) to be a new tree where L is replaced by a non-leaf node testing the value of c and having two leaves as the branches.

The algorithm GreedyPack starts with a tree model consisting only of trivial trees. The algorithm finds the tree which saves the most bits by splitting. To ensure that the decision tree model is valid, GreedyPack builds a dependency graph G describing the dependencies of the trees and makes sure that G is acyclic. The algorithm terminates when no further split can be made that saves any bits.

Algorithm 15 GreedyPack algorithm constructs a decision tree model $\mathcal{T} = \{T_1, \ldots, T_K\}$ from a binary dataset $\mathcal{D}$.

GreedyPack ($\mathcal{D}$) :
$V \leftarrow \{v_1, \ldots, v_K\}$, $E \leftarrow \emptyset$
$G \leftarrow (V, E)$
$T_i \leftarrow \text{TrivialTree}(a_i)$, for $i = 1, \ldots, K$
while there are changes **do**
  for $i = 1, \ldots, K$ **do**
    $O_i \leftarrow T_i$
    for $L \in lvs(T_i)$, $j = 1, \ldots, K$ **do**
      if $E \cup (v_i, v_j)$ is acyclic **and** $a_j \notin path(L)$ **then**
        $U \leftarrow \text{SplitTree}(T_i, L, a_j)$
        if $c(U) < c(O_i)$ **then**
          $O_i \leftarrow U$, $s_i \leftarrow j$
        end if
      end if
    end for
  end for
  $k \leftarrow \arg\min_i \{c(O_i) - c(T_i)\}$
  if $c(O_k) < c(T_k)$ **then**
    $T_k \leftarrow O_k$
    $E \leftarrow E \cup (v_k, v_{s_k})$
  end if
end while
return $\{T_1, \ldots, T_K\}$

8.4 Itemsets and Decision Trees

So far we have discussed how to transmit binary data by using decision trees. In this section we present how to select the itemsets representing the dependencies implied by the decision trees. We will use this link in Section 8.5. A similar link between itemsets and decision trees is explored in [102] although our setup and goals are different.

Given a leaf L, the dependency of the item a_t is captured in the coding table of L. Hence we are interested in finding itemsets that carry the same information. That is, itemsets from which we can compute the coding table. To derive the codes for the leaf L it is sufficient to compute the probability

$$q_{\mathcal{D}}(a_t = 1 \mid L) = q_{\mathcal{D}}(a_t = 1, L) \,/ q_{\mathcal{D}}(L)\,. \tag{8.1}$$

Our goal is to express the probabilities on the right side of the equation using itemsets. In order to do that let P be the path from L to its root. Let $pos(L)$ be the items along the path P which are tested positive. Similarly, let $neg(L)$ be the attributes which are tested negative. Using the inclusion-exclusion principle we see that

$$\begin{aligned} q_{\mathcal{D}}(L) &= q_{\mathcal{D}}(pos(L) = 1, neg(L) = 0) \\ &= \sum_{V \subseteq neg(L)} (-1)^{|V|} fr(pos(L) \cup V)\,. \end{aligned} \tag{8.2}$$

We compute $q_{\mathcal{D}}(a_t = 1, L)$ in a similar fashion. Let us define $sets(L)$ for a given leaf L to be

$$\begin{aligned} sets(L) = &\{V \cup pos(L) \mid V \subseteq neg(L)\} \\ &\cup \{V \cup pos(L) \cup \{a_t\} \mid V \subseteq neg(L)\}\,. \end{aligned}$$

Combining Eqs. 8.1–8.2 we see that the collection $sets(L)$ satisfies our goal.

Proposition 16. *The coding table associated with the leaf L can be computed from the frequencies of $sets(L)$.*

Example 17. *Let L_1, L_2, and L_3 be the leaves (from left to right) of T_4 in Figure 8.1(d). Then the corresponding families of itemsets are $sets(L_1) = \{a, ac\}$, $sets(L_2) = \{b, ab, bc, abc\}$, and $sets(L_3) = \{\emptyset, a, b, ab, c, ac, bc, abc\}$.*

We can easily see that the family $sets(L)$ is essentially the smallest family of itemsets from which the coding table can be derived uniquely.

Proposition 18. *Let $\mathcal{G} \neq sets(L)$ be a family of itemsets. Then there are two data sets, say $\mathcal{D}_1$ and $\mathcal{D}_2$, for which $q_{\mathcal{D}_1}(a_t = 1 \mid L) \neq q_{\mathcal{D}_2}(a_t = 1 \mid L)$ but $fr(\mathcal{G}; \mathcal{D}_1) = fr(\mathcal{G}; \mathcal{D}_2)$.*

Given a tree T we define $sets(T)$ to be $sets(T) = \bigcup_{L \in lvs(T)} sets(L)$. We also define $sets(\mathcal{T}) = \bigcup_i sets(T_i)$ where $\mathcal{T} = \{T_1, \ldots, T_K\}$ is a decision tree model.

8.5 Choosing Good Itemsets

The connection between itemsets and decision trees made in the previous section allows us to consider an orthogonal approach to identify good itemsets. Informally, our goal is to construct decision trees from a family of itemsets $\mathcal{F}$, selecting the subset from $\mathcal{F}$ that provides the best compression of the data. More formally, our new approach is as follows: given a downward closed family of itemsets $\mathcal{F}$, we build a decision tree model $\mathcal{T} = \{T_1, \ldots, T_K\}$ providing a good compression of the data, with $sets(\mathcal{T}) \subseteq \mathcal{F}$.

Before we can describe our main algorithm, we need to introduce some further notation. Firstly, given two trees T_p and T_n not using attribute c, we define JoinTree(c, T_p, T_n) to be the join tree with c as the root node, T_p as the positive branch of c, and T_n as the negative branch of c. Secondly, to define our search algorithm we need to find the best tree

$$bt(a_t; S, \mathcal{F}) = \arg\min_T \{c(T) \mid t(T) = a_t, \\ src(T) \subseteq S, sets(T) \subseteq \mathcal{F}\},$$

that is, $bt(a_t; S, \mathcal{F})$, returns the best tree for a_t for which the related sets are in $\mathcal{F}$ and only splits on attributes in S.

To compute the optimal tree $bt(a_t; S, \mathcal{F})$, we use the exhaustive method (presented originally in [102]) given in Algorithm 16. The algorithm is straightforward: it tests each valid item as the root and recurses itself on both branches.

Algorithm 16 Generate algorithm for calculating $bt(a_t; S, \mathcal{F})$, that is, the best tree T for a_t using only S as source and having $sets(T) \subseteq \mathcal{F}$.

Generate $(a_t, S, \mathcal{F})$:
1: $B \leftarrow S \cap (\bigcup \mathcal{F})$
2: $\mathcal{C} \leftarrow$TrivialTree (a_t)
3: **for all** $b \in B$ **do**
4: $\quad \mathcal{G} \leftarrow \{X - b \mid b \in X \in \mathcal{F}\}$
5: $\quad (\mathcal{D}_p, \mathcal{D}_n) \leftarrow$Split $(\mathcal{D}, b)$
6: $\quad T_p \leftarrow$Generate $(a_t, \mathcal{G}, S, \mathcal{D}_p)$
7: $\quad T_n \leftarrow$Generate $(a_t, \mathcal{G}, S, \mathcal{D}_n)$
8: $\quad \mathcal{C} \leftarrow \mathcal{C} \cup$ JoinTree (b, T_p, T_n)
9: **end for**
10: **return** $\arg\min_T \{c(T) \mid T \in \mathcal{C}\}$

We can now describe the actual algorithm for constructing decision tree models with a low cost. Our method automatically discovers the order in which the attributes can be transmitted most succinct. For this, it needs to find sets of attributes S_i for each attribute a_i such that these should be encoded

before a_i. The collection $\mathcal{S} = \{S_1, \ldots, S_K\}$ should define an acyclic graph and the actual trees are $bt(a_i; S_i, \mathcal{F})$. We use $c(\mathcal{S})$ as a shorthand for the total complexity $\sum_i c(bt(a_i; S_i, \mathcal{F}))$ of the best model built from $\mathcal{S}$.

We construct the set $\mathcal{S}$ iteratively. At the beginning of the algorithm we have $S_i = \emptyset$ and we increase the sets S_i one attribute at a time. We allow ourselves to mark the attributes. The idea is that once the attribute a_i is marked, then we are not allowed to augment S_i any longer. At the beginning none of the nodes are marked.

To describe a single step in the algorithm we consider a graph $H = (v_0, \ldots, v_K)$, where $v_1, \ldots, v_K$ represent the attributes and v_0 is a special auxiliary node. We start by adding edges (v_i, v_0) having the weight $c(bt(a_i; S_i, \mathcal{F}))$, thus the cost of the best tree possible from $\mathcal{F}$ using only the attributes in S_i. Then, for each unmarked node v_i we find out what other extra attribute will help most to encode it succinct. To do this, we add the edge (v_i, v_j) for each v_j with the weight $c(bt(a_i; S_i \cup \{a_j\}, \mathcal{F}))$. Now, let U be the minimum directed spanning tree of H having v_0 as the sink. Consider an unmarked node v_i such that $(v_i, v_0) \in E(U)$. That node is now the best choice to be fixed, as it helps to encode the data best. We therefore mark attribute a_i and add a_i to each S_j for each ancestor v_j of v_i in U. This process is repeated until all attributes are marked. The details of the algorithm are given in Algorithm 17.

The marking of the attributes guarantees that there can be no cycles in $\mathcal{S}$. In fact, the marking order also tells us a valid order for transmitting the attributes. Further, as at least one attribute is marked at each step, this guarantees that the algorithm terminates in K steps.

Let $\mathcal{S}$ be the collection of sources. The following proposition tells us that the augmentation performed by SetPack does not compromise the optimality of collections next to $\mathcal{S}$.

Proposition 19. *Assume the collection of sources $\mathcal{S} = \{S_1, \ldots, S_K\}$. Let $\mathcal{O} = \{O_1, \ldots, O_K\}$ be the collection of sources such that $S_i \subseteq O_i$ and $|O_i| \leq |S_i| + 1$. Let $\mathcal{S}'$ be the collection that Algorithm 17 produces from $\mathcal{S}$ in a single step. Then there is a collection $\mathcal{S}^*$ such that $S'_i \subseteq S^*_i$ and that $c(\mathcal{S}^*) \leq c(\mathcal{O})$.*

Proof. Let G be the graph constructed by Algorithm 17 for the collection S. Construct the following graph W: For each O_i such that $O_i = S_i$ add the edge (v_i, v_0). For each $O_i \neq S_i$ add the edge (v_i, v_j), where $\{a_j\} = O_i - S_i$. But W is a directed spanning tree of G. Let U be the directed minimum spanning tree returned by the algorithm. Let $S^*_i = S'_i$ if $(v_i, v_0) \in E(U)$ and $S^*_i = S'_i \cup \{a_j\}$ if $(v_i, v_j) \in E(U)$. Note that $\mathcal{S}^*$ defines a valid model and because U is optimal we must have $c(\mathcal{S}^*) \leq c(\mathcal{O})$. □

Corollary 20. *Assume that $\mathcal{F}$ is a family of itemsets having 2 items, at maximum. The algorithm* SetPack *returns the optimal tree model.*

Algorithm 17 The algorithm SetPack constructs a decision tree model $\mathcal{T}$ given a family of itemsets $\mathcal{F}$ such that $sets(\mathcal{T}) \subseteq \mathcal{F}$. Returns a DAG, a family $S = (S_1, \ldots, S_K)$ of sets of attributes. The trees are $T_i = bt(a_i, S_i, \mathcal{F})$.

SetPack $(\mathcal{D}, \mathcal{F})$:
$\mathcal{S} = (S_1, \ldots, S_K) \leftarrow (\emptyset, \ldots, \emptyset)$
$r = (r_1, \ldots, r_K) \leftarrow (\textbf{false}, \ldots, \textbf{false})$
$V \leftarrow \{v_0, \ldots, v_K\}$
while there exists $r_i = \textbf{false}$ **do**
  $E \leftarrow \emptyset$
  for $i = 1, \ldots, K$ **do**
    $E \leftarrow E \cup (v_i, v_0)$
    $w(v_i, v_0) \leftarrow c(bt(a_i; S_i, \mathcal{F}))$
    if $r_i = \textbf{false}$ **then**
      for $j = 1, \ldots, K$ **do**
        $T \leftarrow bt(a_i; S_i \cup \{a_j\}, \mathcal{F})$
        if $c(T) \leq w(v_i, v_0)$ **then**
          $E \leftarrow E \cup (v_i, v_j)$, $w(v_i, v_j) \leftarrow c(T)$
        end if
      end for
    end if
  end for
  $U \leftarrow dmst(V, E)$ {Directed Min. Spanning Tree.}
  for $(v_i, v_0) \in E(U)$ **and** $r_i = \textbf{false}$ **do**
    $r_i \leftarrow \textbf{true}$.
    for v_j is a parent of v_i in U **do**
      $S_j \leftarrow S_j + a_i$
    end for
  end for
end while
return S

Let us consider the complexity of the algorithms. The algorithm SetPack runs in a polynomial time. By using dynamic programming we can show that Generate runs in $O(|F|^2)$ time. We also tested a faster variant of the algorithm in which the exhaustive search in Generate is replaced by the greedy approach similar to the ID3 algorithm. We call this variant SetPackGreedy.

8.6 Related Work

Finding interesting itemsets is a major research theme in data mining. To this end, many measures have been suggested over time. A classic measure

for ranking itemsets is frequency, for which there exist efficient search algorithms [5,56]. Other measures involve comparing how much an itemset deviates from the independence assumption [2,18,19,41]. In yet other approaches more flexible models are used, such as, Bayes networks [62,63], Maximum Entropy estimates [92,116]. Related are also low-entropy sets: itemsets for which the entropy of the data is low [59].

Many of these approaches suffer from the fact that they require a user-defined threshold and further that at low thresholds extremely many itemsets are returned, many of which convey the same information. To address the latter problem we can use closed [103] or non-derivable [25] itemsets that provide a concise representation of the original itemsets. However, these methods deteriorate even under small amounts of noise.

Alternative to these approaches of describing the pattern set, there are methods that instead pick groups of itemsets that describe the data well. As such, we are not the first to embrace the compression approach to data mining [42]. In Chapter 2 we introduced the MDL-based Krimp algorithm to battle the frequent itemset explosion at low support thresholds. It returns small subsets of itemsets that together capture the distribution of the data well. Like detailed in the previous chapters, these *code tables* have been successfully applied in various data mining problems. While these applications shows the practicality of the approach, Krimp can only describe the patterns between the items that are present in the dataset. On the other hand, we consider the 0s and the 1s in the data symmetrically and hence we are able to provide more detailed descriptions of the data; including patterns between the presence and absence of items.

More different from our methods are the lossy data description approaches. These strive to describe just part of the data, and as such may overlook important interactions. Summarization [27] is a compression approach that identifies a group of itemsets such that each transaction is summarized by one set with as little loss of information as possible. Yet different are *pattern teams* [69], which are groups of most-informative length-k itemsets [70], selected through an external interestingness measure. As this approach is computationally intensive, the number of team members is typically < 10. Bringmann et al. [20] proposed a similar selection method that can consider larger pattern sets. However, it also requires the user to choose a quality measure to which the pattern set has to be optimized, unlike our parameter-free and lossless method.

Alternatively we can view the approach in this chapter as building a global model for data and then selecting the itemsets that describe the model. This approach then allows us to use MDL as a model selection technique. In a related work [117] the authors build decomposable models in order to select a small family of itemsets that model the data well.

The decision trees returned by our methods, and particularly the DAG

that they form, have a passing resemblance to Bayes networks [13]. However, as both the model construction and complexity weighing differ strongly, so do the outcomes. To be more precise, in our case the distributions $p(x, par(x))$ are modeled and weighted via decision trees whereas in the Bayes network setup any distribution is weighted equally. Furthermore, we use the correspondence between the itemsets and the decision trees to output local patterns, as opposed to Bayes networks which are traditionally used as global models.

8.7 Experiments

This section contains the results of the empirical evaluation of our methods using toy and real datasets.

Datasets

For the experimental validation of the two packing strategies we use a group of datasets with strongly differing statistics. From the LUCS/KDD repository [33] we took a number of often used databases to allow for comparison to other methods. To test our methods on real data we used the *Mammals* presence database and the *Courses* dataset. The latter contains the enrollment records of students taking courses at the Department of Computer Science of the University of Helsinki. The *Mammals* dataset consists of the absence/presence of European mammals [96] in geographical areas of 50×50 kilometers[2]. The details of these datasets are provided in Table 8.1.

Table 8.1: Statistics of the datasets used in the experiments.

Dataset	$\lvert\mathcal{D}\rvert$	K	% of 1's
Anneal	898	71	20.1
Breast	699	16	62.4
Courses	3506	98	4.6
Mammals	2183	40	46.9
Mushroom	8124	119	19.3
Nursery	12960	32	28.1
Pageblocks	5473	44	25.0
Tic–tac–toe	958	29	34.5

[2]The full version of the dataset is available for research purposes upon request, `http://www.european-mammals.org`.

Experiments with toy datasets

To evaluate whether our method correctly identifies (in)dependencies, we start our experimentation using two artificial datasets of 2000 transactions and 10 items. For both databases, the data is generated per transaction, and the presence of the first item is based on a fair coin toss. For the first database, the other items are similarly generated. However, for the second database, the presence of an item is 90% dependent on the previous item. As such, both datasets have item densities of about 50%.

If we apply GreedyPack, our greedy decision tree building method, to these datasets we see that it is unable to compress the independent database at all. Opposing, the dependently generated dataset can be compressed into only 50% of the original number of bits. Inspection of the resulting itemsets show that the resulting model correctly describes the dependencies in detail: The resulting 19 itemsets are $\{a_1, \ldots, a_{10}, a_1a_2, \ldots, a_9a_{10}\}$.

The greedy method

Recall that our goal is to find high quality descriptions of the data. Following the MDL principle, the quality of the found descriptions can objectively be measured by the compression of the data. We present the compressed sizes for GreedyPack in Table 8.2. The encoding costs $c(\mathcal{T})$ include the size of the encoded data and the decision trees. The initial costs, as denoted by $c(\mathcal{T}_b)$, are those of encoding the data using naïve single-node TrivialTrees. Each of these experiments required 1–10 seconds runtime, with an exception of 60s for *Mushroom*.

From Table 8.2, we see that all models returned by GreedyPack strongly reduce the number of bits required to describe the data; this implicitly shows that good models are returned. The quality can be gauged by taking the compression ratios into account. In general, our greedy method reduces the number of bits to only half of what the independent model requires. As two specific examples of the found dependencies, in the *Courses* dataset the course Data Mining was packed using Machine Learning, Software Engineering, Information Retrieval Methods and Data Warehouses. Likewise, AI and Machine Learning were used to pack the Robotics course.

Like discussed above, our approach and Krimp (Chapter 2) have stark differences in what part of the data is considered. However, as both methods use compression, and result good itemsets, it is insightful to compare the algorithms. For the latter we here allow it to compress as well as possible, and thus, consider candidates up to as low min-sup thresholds as feasible.

Let us compare between the outcomes of either method. For Krimp these are itemsets, for ours it is the combination of the decision trees and the related itemsets. We see that Krimp typically returns fewer itemsets than Greedy-

Table 8.2: Compression, number of trees and numbers of extracted itemsets for the greedy algorithm and Krimp.

	GreedyPack				
Dataset	$c(\mathcal{T}_b)$ (bits)	$c(\mathcal{T})$ (bits)	$\frac{c(\mathcal{T})}{c(\mathcal{T}_b)}$(%)	*# trees*	*# sets*
Anneal	23104	12342	**53.4**	71	1203
Breast	8099	2998	**37.0**	16	17
Courses	76326	61685	**80.8**	98	1230
Mammals	78044	50068	**64.2**	40	845
Mushroom	442062	115347	**26.1**	119	999
Nursery	337477	180803	**53.6**	32	3409
Pageblocks	15280	7611	**49.8**	44	219
Tic–tac–toe	25123	14137	**56.3**	29	619
	Krimp				
Dataset	*minsup*	*# bits*	*compr. ratio (%)*		*# sets*
Anneal	1	22154	34.6		102
Breast	1	4613	16.9		30
Courses	2	71019	79.3		148
Mammals	200	90192	42.3		254
Mushroom	1	231877	20.9		424
Nursery	1	258898	45.5		260
Pageblocks	1	10911	5.0		53
Tic–tac–toe	1	28812	62.3		162

Pack. However, our method returns itemsets that describe interactions between both present *and* absent items.

Next, we observed that especially the initial Krimp compression requires many more bits than ours, and as such Krimp attains better compression ratios. However, if we disregard the ratios and look at the raw number of bits the two methods require, we see that Krimp generally requires twice as many bits to describe *only* the 1's in the data than GreedyPack does to represent *all* of the data.

Validation through classification

To further assess the quality of our models we use a simple classification scheme (see Chapter 2). First, we split the training database into separate

class-databases. We pack each of these. Next, the class labels of the unseen transactions were assigned according to the model that compressed it best.

We ran these experiments for three databases, viz. *Mushroom*, *Breast* and *Anneal*. A random 90% of the data was used to train the models, leaving 10% to test the accuracy on. The accuracy scores we noted, resp. 100%, 98.0% and 93.4%, are fully comparable to (and for the second, even better than) the classifiers considered in Chapter 2.5.

Choosing good itemsets

Table 8.3: Candidate Itemsets for Selection, and Selection by Krimp

	Candidate Itemsets		Krimp	
Dataset	*min-sup*	*# sets*	*# bits*	*# sets*
Anneal	175	8837	31196	53
Breast	1	9920	4613	30
Courses	55	5030	73287	93
Mammals	700	7169	124737	125
Mushroom	1000	123277	474240	140
Nursery	50	25777	265064	225
Pageblocks	1	63599	10911	53
Tic–tac–toe	7	34019	28957	159

Table 8.4: Itemset selection by SetPack and SetPackGreedy

	SetPack			SetPackGreedy		
Dataset	$c(\mathcal{T})$	$\frac{c(\mathcal{T})}{c(\mathcal{T}_b)}(\%)$	*# sets*	$c(\mathcal{T})$	$\frac{c(\mathcal{T})}{c(\mathcal{T}_b)}(\%)$	*# sets*
Anneal	20777	**89.9**	103	20781	**89.9**	69
Breast	5175	**63.7**	42	5172	**63.9**	49
Courses	64835	**84.9**	268	64937	**85.1**	262
Mammals	65091	**83.4**	427	65622	**84.1**	382
Mushroom	313428	**70.9**	636	262942	**59.5**	1225
Nursery	314081	**93.0**	276	314295	**93.1**	218
Pageblocks	11961	**78.3**	92	11967	**78.3**	95
Tic–tac–toe	23118	**92.0**	620	23616	**94.0**	277

In this subsection we evaluate SetPack, our itemset selection algorithm. Recall that this algorithm selects itemsets such that they allow for building succinct encoding decision trees. The difference with GreedyPack is that in this setup the resulting itemsets should be a subset of a given candidate family. Here, we consider frequent itemsets as candidates. We set the support threshold such that the experiments with SetPack were finished within $\frac{1}{2}$–2 hours, with an exception of 23 hours for considering the large candidate family for *Mushroom*. For comparison we use the same candidates for Krimp. We also compare to SetPackGreedy, which required 1–12 minutes, 7 minutes typically, with an exception of $2\frac{1}{2}$ hours for *Mushroom*.

Comparing the results of this experiment (Tables 8.3 and 8.4) with the results of GreedyPack in the previous experiment, we see that the selection process is more strict: now even fewer itemsets are regarded as interesting enough. Large candidate collections are strongly reduced in number: up to three orders of magnitude. On the other hand, the compression ratios are still very good. The reason that GreedyPack produces smaller compression ratios is because it is allowed to consider any itemset.

Further, the fact alone that even with this very strict selection the compression ratios are generally well below 90% show that these few sets are indeed of high importance to describing the major interactions in the data.

If we compare the number of selected sets to Krimp, we see that our method returns in the same order as many itemsets. These descriptions require far less bits than those found by Krimp. As such, ours are a better approximation of the Kolmogorov complexity of the data.

Between SetPack and SetPackGreedy the outcomes are very much alike; this goes for both the obtained compression as well as the number of returned itemsets. However, the greedy search of SetPackGreedy allows for much shorter running times.

8.8 Discussion

The experimentation on our methods validates the quality of the returned models. The models correctly detect dependencies in the data while ignoring independencies. Only a small number of itemsets is returned, which are shown to provide strong compression of the data. By the MDL principle we then know these describes all important regularities in the data distribution in detail efficiently and without redundancy. This claim is further supported by the high classification accuracies our models achieve.

The GreedyPack algorithm generally uses more itemsets and obtains better packing ratios than SetPack. While GreedyPack is allowed to use any itemset, SetPack may only use frequent itemsets. This suggests that we may able to achieve better ratios if we use different candidates, for example, low-

entropy sets [59].

The running times of the experiments reported in this work range from seconds to hours and depend mainly on the number of attributes and rows of the datasets. The exhaustive version SetPack may be slow on very large candidate sets, however, the greedy version SetPackGreedy can even handle such families well. Considering that our current implementation is rather naïve and the fact that both methods are easily parallelized, both GreedyPack and SetPackGreedy are suited for the analysis of large databases.

The main outcomes of our models are the itemsets that identify the encoding paths. However, the decision trees from which these sets are extracted can also be regarded as interesting as these provide an easily interpretable view on the major interactions in the data. Further, just considering the attributes used in such a tree as an itemset also allows for simple inspection of the main associations.

In this work we employ the MDL criterion to identify the optimal model. Alternatively, one could consider using either BIC or AIC, both of which can easily be applied to judge between our decision tree-based models.

8.9 Conclusions

In this chapter we presented two methods that find compact sets of high quality itemsets. Both methods employ compression to select the group of patterns that describe *all* interactions in the data best. That is, the data is considered symmetric and thus both the 0s and 1s are taken into account in these descriptions. Experimentation with our methods showed that high quality models are returned. Their compact size, typically tens to thousands of itemsets, allow for easy further analysis of the found interactions.

CHAPTER 9

Conclusions

In this thesis we have discussed how pattern mining can be made useful. Pattern mining holds strong promise for extracting useful insight from large collections of data. In practice, however, patterns are found and returned too easily. It has proven difficult to select potentially interesting patterns on their individual merit, leading to high numbers of highly redundant results. This prohibits the found patterns to be analysed and used practically. This thesis proposes a different approach, namely to mine the best *set of patterns* instead. Extensive evaluation and application shows this approach to successfully address our research goal: making pattern mining useful.

The main contributions of this thesis can be summarised as follows.

- We proposed to use the Minimum Description Length principle to select small groups of patterns that together describe the data well. To find these sets of itemsets, we introduced Krimp: a heuristic for finding the frequent itemsets that together optimally compress the database. Through extensive evaluation, we showed the high quality of these *code tables*. Experiments regarding the choices in the algorithm identified the best settings for the algorithm, making it parameter-free for all practical purposes.
- We showed how the difference between transaction databases can be measured and characterised through the use of code tables. We defined a difference measure based on the relative Krimp-compressibility of the datasets. These differences can be characterised on three levels of increasing detail: 1) comparing the code table pattern usage between the databases, 2) analysing how patterns from the one code table are described by the code table for the other database, and 3) zooming in to how individual transactions are described by the different code tables.

- We specified the problem of identifying the parts of a transaction database drawn from different distributions in terms of MDL: the best decomposition minimises the total compressed size. We gave two orthogonal parameter-free algorithms to identify and characterise the components of a database. Data is split into homogeneous blocks, such that the compression is optimised. No prior knowledge on the distributions, distance metric or the number of components has to be known or specified. Experiments showed that highly characteristic components are identified.

- We discussed using Krimp code tables as generative models. The resulting generated data contains the patterns present in the original data, including correct margins. As such, the generated data is virtually indistinguishable from the original. Further, the probability of transactions both present in the original and generated databases can be simply controlled. These traits make the method a good alternative to data perturbation when high-quality data has to be released that may not harm privacy.

- We considered the problem of high quality imputation of missing data. We gave three algorithms for completing data with missing values. All follow the MDL principle: the best completion is the completion that can be compressed best. As an objective test we proposed (ϵ, δ)-correctness to measure the difference between two databases in terms of count statistics. The experiments showed this pattern based approach to be superior to the current state of the art, a global modelling approach, both in terms of accuracy and count statistics.

- We extended the concept of selecting patterns by MDL to low-entropy sets; a generalisation of frequent itemsets that identify strong interactions between attributes. The algorithm we introduced, LESS, selects very small collections of descriptive low-entropy sets: typically only tens of patterns. These small numbers and the interpretability of the patterns facilitates thorough analysis by experts. By using entropy instead of frequency, LESS is particularly suited for mining dense data. Further, by regarding data 0/1 symmetric, all major interactions in the data are captured, not just co-occurences.

- Last, but not least, we presented a method that employs refined MDL to select itemsets that describe *all* interactions in the data: Pack. As models, it employs decision trees to compress data 0/1 symmetrically. These trees can be used to select itemsets from large collection, as well as be mined directly from data. Following, itemsets can be simply extracted from the trees. Experimentation showed that high quality models are returned that provide very high compression ratios. The trees, and the modest number of resulting itemsets, allow for easy further analysis of the found interactions.

This thesis addressed the research goal of making pattern mining useful. From the results a number of strong conclusions can be drawn.

First, the main conclusion is that to the end of making pattern mining useful simply the best *set of patterns* should be mined, as opposed to *all* patterns that satisfy certain criteria. As the target to which the set of patterns should be optimised, the conclusion is that data description is a rather good choice. By mining the sets of patterns that describe the data best, the outcome provides a detailed view on the data. Our doctor, whom we met in the introduction, can use this data description in terms of patterns to overview the data and explore various parts of it. The patterns provide instant characterisation.

The most important conclusion of this thesis, however, is that the MDL principle is particularly well-suited for mining useful patterns. By using this principle to select the set of patterns that describe the data best, we are returned very few, but high-quality, patterns. These patterns characterise the distribution of the data very well, as is shown by the results presented in this thesis. Depending on the pattern type and selection method, only tens to thousands of patterns are mined. In these modest numbers, our doctor can easily inspect these patterns, and use them in further analysis.

The general conclusion of the thesis is that MDL is a natural choice for finding good solutions to many data mining problems. We have shown that by stating a variety of data mining problems in terms of MDL very high performance can be attained. In particular, we have shown that the sets of patterns returned by Krimp are generally applicable and provide such high performance and, again, insightful characterisation of the outcome. Many of the addressed problems, such as incomplete records and unknown dissimilarity, are often faced by our doctor when trying to extract knowledge from data: it is therefore safe to say that the sets of patterns identified through MDL are indeed useful.

Summarising, the results presented in this thesis will make our doctor very happy. Pattern mining can be made even more useful, however, i.e. the ulterior peak of usefulness is not yet reached. This thesis shows a way through which pattern mining delivers on the promise of providing useful insight and practical application. The approach of using MDL to mine sets of patterns comes with many new challenges and opportunities. Ideally, for instance, patterns could be mined directly, without need for first finding large collections of candidate patterns. Along the same line, it would be good if numeric data could be mined for descriptive patterns, without having to discretise it in advance. Further, it would be beneficial to know some bounds, that is, to know how good the heuristic solutions exactly are with regard to the optimum. Addressing these issues will make for rather interesting future research that can further increase the usefulness of pattern mining.

Bibliography

1. C.C. Aggarwal, C. Procopiuc, and P.S. Yu. Finding localized associations in market basket data. *Trans Knowledge and Data Engineering*, 14(1):51–62, 2002.
2. C.C. Aggarwal and P.S. Yu. A new framework for itemset generation. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98)*, pages 18–24. ACM Press, 1998.
3. C.C. Aggarwal and P.S. Yu. A condensation approach to privacy preserving data mining. In *Proceedings of the EDBT'04*, pages 183–199, 2004.
4. D. Agrawal and C.C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the SIGMOD'01*, pages 247–255. ACM, 2001.
5. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI, 1996.
6. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the VLDB'94*, pages 487–499, 1994.
7. R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the SIGMOD'00*, pages 439–450. ACM, 2000.
8. P.D. Allison. *Missing Data – Quantitative Applications in the Social Science*. Sage Publishing, 2001.
9. P. Andritsos, P. Tsaparas, R.J. Miller, and K.C. Sevcik. LIMBO: Scalable clustering of categorical data. In *Proceedings of the EDBT'04*, pages 124–146, 2004.
10. M. Bankier. Canadian census minimum change donor imputation methodology. In *Proceedings of the UN/ECE Workshop on Data Editing*, 2000.
11. R. Bathoorn, A. Koopman, and A. Siebes. Reducing the frequent pattern set. In *Proceedings of the ICDM-Workshops'06*, pages 55–59, 2006.
12. R. Bayardo. Efficiently mining long patterns from databases. In *Proceedings of the SIGMOD'98*, pages 85–93, 1998.

13. J. Bernardo and A. Smith. *Bayesian Theory*. Wiley Series in Probability and Statistics. John Wiley and Sons, 1994.

14. H. Bischof, A. Leonardis, and A. Sleb. Mdl principle for robust vector quantization. *Pattern Analysis and Applications*, 2:59–72, 1999.

15. C. Böhm, C. Faloutsos, J-Y. Pan, and C. Plant. Robust information-theoretic clustering. In *Proceedings of the KDD'06*, pages 65–75, 2006.

16. I. Bouzouita, S. Elloumi, and S. Ben Yahia. GARC: A new associative classification approach. In *Proceedings of the DaWaK'06*, pages 554–565, 2006.

17. T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets. The use of association rules for product assortment decisions: a case study. In *Proceedings of the KDD'99*, pages 254–260, 1999.

18. S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *Proceedings of the SIGMOD'97*, pages 265–276, 1997.

19. S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the SIGMOD'97*, pages 255–264, 1997.

20. B. Bringmann and A. Zimmermann. The chosen few: On identifying valuable patterns. In *Proceedings of the ICDM'07*, pages 63–72, 2007.

21. R. Bruni. Discrete models for data imputation. *Discrete Applied Mathematics*, 144(1-2):59–69, 2004.

22. A. Bürgin-Wolff and F Hadziselimovic. Coeliac disease. *The Lancet*, 362(9393):1418 – 1419, 2003.

23. S. van Buuren. Multiple imputation of discrete and continuous data by fully conditional specification. *Statistical Methods in Medical Research*, 16(6):219–242, 2007.

24. I.V. Cadez, P. Smyth, and H. Mannila. Probabilistic modeling of transaction data with applications to profiling, visualization, and prediction. In *Proceedings of the KDD'01*, pages 37–46, 2001.

25. T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In *Proceedings of the ECML PKDD'02*, pages 74–85, 2002.

26. D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-associations. In *Proceedings of the KDD'04*, pages 79–88, 2004.

27. V. Chandola and V. Kumar. Summarization – compressing data into an informative representation. In *Proceedings of the ICDM'05*, pages 98–105, 2005.

28. V. Chandola and V. Kumar. Summarization – compressing data into an informative representation. *Knowl. Inf. Syst.*, 12(3):355–378, 2007.

29. B. Chen, W.E. Winkler, and R.J. Hemmig. Using the DISCRETE edit system for ACS surveys. Technical report, U.S. Bureau of the Census, 2000.

30. K. Chen and L. Liu. Privacy preserving data classification with rotation pertubation. In *Proceedings of the ICDM'05*, pages 589–592, 2005.

31. R. Cilibrasi and P. Vitanyi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005.

32. E.F. Codd, S.B. Codd, and C.T. Salley. Providing olap (on-lineanalytical processing) to user analyst: An it mandate. `http://www.arborsoft.com/OLAP.html`, 1994.

33. F. Coenen. The LUCS-KDD discretised/normalised ARM and CARM data library. `http://www.csc.liv.ac.uk/~frans/KDD/Software/LUCS-KDD-DN/DataSets/dataSets.html`, 2003.

34. F. Coenen. The LUCS-KDD software library. `http://www.csc.liv.ac.uk/~frans/KDD/Software/`, 2004.

35. T.M. Cover and J.A. Thomas. *Elements of Information Theory, 2nd ed.* John Wiley and Sons, 2006.

36. B. Crémilleux and J-F. Boulicaut. Simplest rules characterizing classes generated by δ-free sets. In *Proceedings of the KBSAAI'02*, pages 33–46, 2002.

37. A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–Ű38, 1977.

38. G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the SIGKDD'99*, pages 43–52, 1999.

39. G. Dong and J. Li. Mining border descriptions of emerging patterns from dataset pairs. *Knowledge and Information Systems*, 8(2):178–202, 2005.

40. R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, New York, 1973.

41. W. DuMouchel and D. Pregibon. Empirical bayes screening for multi-item associations. In *Proceedings of the SIGKDD'01*, pages 67–76, 2001.

42. C. Faloutsos and V. Megalooikonomou. On data mining, compression and Kolmogorov complexity. In *Data Mining and Knowledge Discovery*, volume 15, pages 3–20. Springer-Verlag, 2007.

43. Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comp. Sys. Sci.*, 55(1):119–139, 1997.

44. N. Friedman. Learning bayesian networks in the presence of missing values and hidden variables. In *Proceedings of the ICML'97*, pages 125–Ű133, 1997.

45. N. Friedman. The bayesian structural EM algorithm. In *Proceedings of the UAI'98*, pages 129–Ű138, 1998.

46. N. Friedman and G. Elidan. LibB for Windows/Linux programs 2.1. http://www.cs.huji.ac.il/labs/compbio/LibB, accessed December 2008, 2008.

47. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H.Freeman, 1979.

48. F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. In *Proceedings of the DS'04*, pages 278–289, 2004.

49. A. Gionis, H. Mannila, T. Mielikäinen, and P. Tsaparas. Assessing data mining results via swap randomization. *ACM Trans. Knowl. Discov. Data*, 1(3):14, 2007.

50. B. Goethals and M. Javeed Zaki. Frequent itemset mining implementations repository (FIMI). `http://fimi.cs.helsinki.fi`, 2003.

51. E. Gokcay and J.C. Principe. Information theoretic clustering. *Trans. Pattern Analysis and Machine Intelligence*, 24(2):158–171, 2002.

52. P. D. Grünwald. Minimum description length tutorial. In P.D. Grünwald and I.J. Myung, editors, *Advances in Minimum Description Length*. MIT Press, 2005.

53. P. D. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

54. J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: Current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.

55. J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

56. J. Han and J. Pei. Mining frequent patterns by pattern-growth: methodology and implications. *SIGKDD Explorations Newsletter*, 2(2):14–20, 2000.

57. D. Hand, N. Adams, and R. Bolton, editors. *Pattern Detection and Discovery*. Springer-Verlag, 2002.

58. H. Heikinheimo, M. Fortelius, J. Eronen, and H. Mannila. Biogeography of european land mammals shows environmentally distinct and spatial coherent clusters. *Biogeogr.*, 34(6):1053–1064, 2007.

59. H. Heikinheimo, E. Hinkkanen, H. Mannila, T. Mielikäinen, and J. K. Seppänen. Finding low-entropy sets and trees from binary data. In *Proceedings of the KDD'07*, pages 350–359, 2007.

60. Hannes Heikinheimo, Jilles Vreeken, Arno Siebes, and Heikki Mannila. Low-entropy set selection. In *Proceedings of the SDM'09*, pages 569–579, 2009.

61. Z. Huang, W. Du, and B. Chen. Deriving private information from randomized data. In *Proceedings of the SIGMOD'05*. ACM, 2005.

62. S. Jaroszewicz and T. Scheffer. Fast discovery of unexpected patterns in data, relative to a bayesian network. In *Proceedings of the KDD'05*, pages 118–127, 2005.

63. S. Jaroszewicz and D.A. Simovici. Interestingness of frequent itemsets using bayesian networks as background knowledge. In *Proceedings of the KDD'04*, pages 178–186, 2004.

64. H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. Random-data perturbation techniques and privacy-preserving data mining. *Knowledge and Information Systems*, 4(7):387–414, 2005.

65. Richard M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Proc. of a Symp. on the Complexity of Computer Computations*, pages 85–103, New York, USA, 1972. Plenum Press.

66. Y. Ke, J. Cheng, and W. Ng. Mining quantitative correlated patterns using an information-theoretic approach. In *Proceedings of the KDD'06*, pages 227–236, 2006.

67. E. Keogh, S. Lonardi, and C.A. Ratanamahatana. Towards parameter-free data mining. In *Proceedings of the KDD'04*, pages 206–215, 2004.

68. E. Keogh, S. Lonardi, C.A. Ratanamahatana, L. Wei, S-H. Lee, and J. Handley. Compression-based data mining of sequential data. *Data Min. Knowl. Discov.*, 14(1):99–129, 2007.

69. A. J. Knobbe and E. K. Y. Ho. Pattern teams. In *Proceedings of the ECML PKDD'06*, pages 577–584, 2006.

70. A.J. Knobbe and E.K. Y. Ho. Maximally informative k-itemsets and their efficient discovery. In *Proceedings of the KDD'06*, pages 237–244, 2006.

71. R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDD-Cup 2000 organizers' report: Peeling the onion. *SIGKDD Explorations*, 2(2):86–98, 2000. urlhttp://www.ecn.purdue.edu/KDDCUP.

72. P. Kontkanen and P. Myllymäki. A linear-time algorithm for computing the multinomial stochastic complexity. *Information Processing Letters*, 103(6):227–233, 2007.

73. P. Kontkanen, P. Myllymäki, W. Buntine, J. Rissanen, and H. Tirri. An mdl framework for clustering. Technical report, HIIT, 2004. Technical Report 2004-6.

74. A. Koopman and A. Siebes. Discovering relational items sets efficiently. In Mohammed Zaki and Ke Wang, editors, *Proceedings of the SDM'08*, pages 108–119. SIAM, 2008.

75. A. Koopman and A. Siebes. Characteristic relational patterns. In *Proceedings of the KDD'09*, pages 437–446, 2009.

76. J. Kovar and W.E. Winkler. Comparison of GEIS and SPEER for editing economic data. Technical report, U.S. Bureau of the Census, 2000.

77. M. Koyotürk, A. Grama, and N. Ramakrishnan. Compression, clustering, and pattern discovery in very high-dimensional discrete-attribute data sets. *Trans Knowledge and Data Engineering*, 17(4):447–461, 2005.

78. M. van Leeuwen, J. Vreeken, and A. Siebes. Compression picks the item sets that matter. In *Proceedings of the ECML PKDD'06*, pages 585–Ű592, 2006.

79. M. van Leeuwen, J. Vreeken, and A. Siebes. Identifying the components. *Data Min. Knowl. Discov.*, 19(2):173–292, 2009.

80. M. Li, X. Chen, X. Li, B. Ma, and P.M.B. Vitányi. The similarity metric. *IEEE Trans. on Information Theory*, 50(12):3250–3264, 2004.

81. M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, 1993.

82. C.K. Liew, U.J. Choi, and C.J. Liew. A data distortion by probability distribution. *ACM Trans. on Database Systems*, 3(10):395–411, 1985.

83. R.J.A. Little and D.B. Rubin. *Statistical Analysis with Missing Data (2nd Edition)*. John Wiley and Sons, 2002.

84. B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proceedings of the KDD'98*, pages 80–Ű86, 1998.

85. G. Liu, H. Lu, J. Xu Yu, W. Wei, and X. Xiao. AFOPT: An efficient implementation of pattern growth approach. In *Proceedings of the 2nd workshop on Frequent Itemset Mining Implementations*, 2004.

86. K. Liu, C. Giannella, and H. Kargupta. An attacker's view of distance preserving maps for privacy preserving data mining. In *Proceedings of the ECMLPKDD'06*, pages 297–308, 2006.

87. A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *Proceedings of the ICDE'06*, pages 24–35, 2006.

88. J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Symposium on Mathematical Statistics and Probability*, 1967.

89. H. Mannila and H. Toivonen. Multiple uses of frequent sets and condensed representations. In *Proceedings of the KDD'96*, pages 189–194, 1996.

90. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, pages 241–258, 1997.

91. M. Mehta, R. Agrawal, and J. Rissanen. Sliq: A fast scalable classifier for data mining. In *Advances in database technology*, pages 18–32. Springer, 1996.

92. R. Meo. Theory of dependence values. *ACM Trans. Database Syst.*, 25(3):380–406, 2000.

93. D. Meretakis, H. Lu, and B. Wüthrich. A study on the performance of large bayes classifier. In *Proceedings of the ECML'00*, pages 271–279, 2000.

94. S. Merugu and J. Ghosh. Privacy-preserving distributed clustering using generative models. In *Proceedings of the ICDM'03*, pages 211–218, 2003.

95. T. Mielikäinen and H. Mannila. The pattern ordering problem. In *Proceedings of the ECML PKDD'03*, pages 327–338, 2003.

96. A.J. Mitchell-Jones, G. Amori, W. Bogdanowicz, B. Krystufek, P.J. H. Reijnders, F. Spitzenberger, M. Stubbe, J.B.M. Thissen, V. Vohralik, and J. Zima. *The Atlas of European Mammals*. Academic Press, 1999.

97. K. Morik, J-F. Boulicaut, and A. Siebes, editors. *Local Pattern Detection*. Springer-Verlag, 2005.

98. S. Morishita and J. Sese. Traversing itemset lattice with statistical metric pruning. In *Proceedings of the PODS'00*, pages 226–236, 2000.

99. K. Murphy. Bayes net toolbox for Matlab. `http://www.cs.ubc.ca/\~{}murphyk/Software/BNT/`, accessed December 2008, 1997.

100. K.V.S. Murthy. *On growing better decision trees from data*. PhD thesis, Johns Hopkins Univ., Baltimore, 1996.

101. S. Myllykangas, J. Himberg, T. Böhling, B. Nagy, J. Hollmén, and S. Knuutila. Dna copy number amplification profiling of human neoplasms. *Oncogene*, 25(55):7324Ű7332, 2006.

102. S. Nijssen and É. Fromont. Mining optimal decision trees from itemset lattices. In *Proceedings of the KDD'07*, pages 530–539, 2007.

103. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of the ICDT'99*, pages 398–416, 1999.

104. R. Pensa, C. Robardet, and J-F. Boulicaut. A bi-clustering framework for categorical data. In *Proceedings of the ECML PKDD'05*, pages 643–650, 2005.

105. B. Pfahringer. Compression-based feature subset selection. In *Proceedings of the IJCAI'95 Workshop on Data Engineering for Inductive Learning*, pages 109–119, 1995.

106. J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan-Kaufmann, Los Altos, California, 1993.

107. J.R. Quinlan. FOIL: a midterm report. In *Proceedings of the ECML'93*, 1993.

108. J. Rissanen. Modeling by shortest data description. *Automatica*, 14(1):465–471, 1978.

109. J. Rissanen. Fisher information and stochastic complexity. *IEEE Transactions on Information Theory*, 42(1):40–47, 1996.

110. D.B. Rubin. *Multiple imputation for nonresponse in surveys*. John Wiley and Sons, 1987.

111. P. Samarati. Protecting respondents' identities in microdata release. *IEEE Trans. on Knowledge and Data Engineering*, 13(6):1010–1027, 2001.

112. J.L. Schafer. Analysis of incomplete multivariate data. *Monographs on Statistics and Applied Probability*, 72:1–448, 1997.

113. A. Siebes, J. Vreeken, and M. van Leeuwen. Item sets that compress. In *Proceedings of the SDM'06*, pages 393–404, 2006.

114. A. Stuart, K. Ord, and S. Arnold. *Classical Inference and the Linear Model*, volume 2A of *Kendall's Advanced Theory of Statistics*. Arnold, 1999.

115. J. Sun, C. Faloutsos, S. Papadimitriou, and P.S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the KDD'07*, pages 687–696, 2007.

116. N. Tatti. Maximum entropy based significance of itemsets. *Knowledge and Information Systems (KAIS)*, 17(1):57–77, 2008.

117. N. Tatti and H. Heikinheimo. Decomposable families of itemsets. In *Proceedings of the ECMLPKDD'08*, 2008.

118. N. Tatti and J. Vreeken. Finding good itemsets by packing data. In *Proceedings of the ICDM'08*, pages 588–597, 2008.

119. N. Tishby, F.C. Pereira, and W. Bialek. The information bottleneck methods. In *Proceedings of the Allerton Conf. on Communication, Control and Computing*, pages 368–377, 1999.

120. D. Titterington, A. Smith, and U. Makov. *Statistical Analysis of Finite Mixture Distributions*. John Wiley and Sons, 1985.

121. V.N. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, 1998.

122. J. Vreeken and A. Siebes. Filling in the blanks – Krimp minimisation for missing data. In *Proceedings of the ICDM'08*, pages 1067–1072, 2008.

123. J. Vreeken, M. van Leeuwen, and A. Siebes. Krimp: Mining itemsets that compress. *Data Mining and Knowledge Discovery*. accepted for publication.

124. J. Vreeken, M. van Leeuwen, and A. Siebes. Characterising the difference. In *Proceedings of the KDD'07*, pages 765–Ű774, 2007.

125. J. Vreeken, M. van Leeuwen, and A. Siebes. Preserving privacy through data generation. In *Proceedings of the ICDM'07*, pages 685–690, 2007.

126. C.S. Wallace. *Statistical and inductive inference by minimum message length*. Springer-Verlag, 2005.

127. J. Wang and G. Karypis. SUMMARY: Efficiently summarizing transactions for clustering. In *Proceedings of the ICDM'04*, pages 241–248, 2004.

128. J. Wang and G. Karypis. HARMONY: Efficiently mining the best rules for classification. In *Proceedings of the SDM'05*, pages 205–216, 2005.

129. J. Wang and G. Karypis. On efficiently summarizing categorical databases. *Knowl. Inf. Syst.*, 9(1):19–37, 2006.

130. K. Wang, C. Xu, and B. Liu. Clustering transactions using large items. In *Proceedings of the CIKM'99*, pages 483–490, 1999.

131. H.R. Warner, A.F. Toronto, L.R. Veasey, and R. Stephenson. A mathematical model for medical diagnosis, application to congenital heart disease. *Journal of the American Medical Association*, 177:177–184, 1961.

132. I. Wasito and B. Mirkin. Nearest neighbour approach in the least-squares data imputation algorithms. *Journal of Information Sciences*, 167:1–25, 2005.

133. I.H. Witten and Eibe Frank. *Data Mining:Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.

134. Y. Xiang, R. Jin, D. Fuhry, and F.F. Dragan. Succinct summarization of transactional databases: an overlapped hyperrectangle scheme. In *Proceedigns of the KDD'08*, pages 758–766, 2008.

135. X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing itemset patterns: A profile-based approach. In *Proceedings of the KDD'05*, pages 314–323, 2005.

136. X. Yin and J. Han. CPAR: Classification based on predictive association rules. In *Proceedings of the SDM'03*, pages 331–335, 2003.

137. X. Zhang, D. Guozhu, and K. Ramamohanarao. Information-based classification by aggregating emerging patterns. In *Proceedings of the IDEAL'00*, pages 48–53, 2000.

Index

Samenvatting

Het ontdekken van patronen is een belangrijk onderdeel van 'data mining'. Data mining is een relatief nieuw onderzoeksgebied binnen de informatica dat zich richt op het vergaren van nieuwe kennis uit bestaande gegevens. Netter gezegd, data mining betreft het extraheren van niet-triviale inzichten uit grote verzamelingen van gegevens.

Een patroon is simpelweg een vorm van regelmaat. Voorbeelden van patronen kunnen producten zijn die vaak gezamenlijk verkocht worden, of genen die vooral actief zijn bij een bepaalde ziekte, of gedragingen van klanten die winst opbrachten, enzovoorts. Dat zulke patronen nuttig inzicht kunnen bieden aan experts moge duidelijk zijn. Het extraheren van dergelijke patronen uit gegevens noemt men 'pattern mining', of, in goed Nederlands, patroon mining.

In het algemeen is het vinden van een patroon zeer eenvoudig. Een interessant patroon vinden is echter heel andere koek. Dit proefschrift behandelt hoe juist interessante patronen gevonden kunnen worden. Het gaat over hoe een behapbaar aantal, zeer interessante, regelmatigheden te vinden. En, in het bijzonder, hoe deze patronen nuttig te gebruiken zijn bij het verder analyseren van gegevens. Grof gezegd gaat het over het bruikbaar maken van patroon mining.

Bestaande technieken vinden namelijk met gemak enorme aantallen patronen. Vaak worden meer patronen gevonden dan dat er gegevens in de database zitten. Dit komt met name doordat het stuk voor stuk toetsen of een patroon interessant is niet goed blijkt te werken: het is heel lastig om de mate van interessantheid te beoordelen. Doen we dit te streng, dan worden er niets dan bekende feiten gevonden. Doen we dit te soepel, dan worden al snel vrijwel alle patronen opgeleverd. Veel patronen zijn namelijk kleine variaties op hetzelfde thema, en worden dan allemaal als interessant bestempeld. Bestaande voorstellen om overtollige patronen te verwijderen bieden slechts beperkte verlichting.

In dit licht bezien is patroon mining dus nog niet bruikbaar: experts kunnen de resultaten niet zonder meer analyseren, en ook anderzijds zijn de patronen lastig toepasbaar. Dit proefschrift stelt voor om, in tegenstelling tot álle patronen die aan bepaalde eisen voldoen, de beste *groep* patronen te vinden. Zo'n groep dient de gegevens goed te beschrijven, met zo min mogelijk overbodige

details. Om dit te beoordelen gebruiken we het zogenoemde Minimum Description Length principe. Dit principe stelt dat de meest compacte beschrijving van de gegevens tevens de beste beschrijving is. Anders gezegd, we zijn op zoek naar de groep patronen die de gegevens het beste comprimeren.

Het idee is om patronen in de data te vervangen door codewoorden. Hoe vaker een patroon gebruikt wordt om een stuk data te beschrijven, des te korter het codewoord. Door het vergelijken van de lengte van de verkregen beschrijving, dat wil zeggen, de gecodeerde gegevens en het codeboek, kunnen we bepalen of een groep patronen beter is dan een andere groep. Via deze weg kan gezocht worden naar de beste groep patronen.

Om deze beste groep te vinden introduceren we drie verschillende technieken, genaamd Krimp, LESS en Pack. Elk pakt het probleem op een andere manier aan, met eigen voor- en nadelen. Gemeen hebben de drie dat ze compacte groepjes patronen opleveren: slechts tien- tot duizendtallen, in plaats van miljarden. Hiermee lossen alle drie de explosie van het aantal gevonden patronen op. Via onafhankelijke toetsing wordt getoond dat deze groepjes patronen de gegevens goed karakteriseren. Zo kunnen ze op natuurlijke wijze, en met zeer goede prestaties, gebruikt worden voor classificatie.

De bruikbaarheid van de patronen wordt verder getoetst aan de hand van een aantal open problemen. Zo worden getoond dat via deze patronen het verschil tussen databases gemeten en gekarakteriseerd kan worden. Hierbij kan gedacht worden aan het verschil tussen groepen patienten, of dat van verkooppatronen tussen supermarkten. Gegevens kunnen ook op basis van dergelijke karakteristieken gegroepeerd worden: groepen klanten met hetzelfde koopgedrag, of patienten met dezelfde patronen in leefstijl, behandelgegevens of DNA.

De gevonden patronen kunnen ook worden gebruikt om data met dezelfde karakteristieken als het origineel te genereren. Dit lost het privacy-probleem op dat optreedt wanneer patient-databases publiek gemaakt moeten worden. De gegenereerde gegevens zijn praktisch identiek aan het origineel, met het grote verschil dat de kans dat er een bestaand persoon in aangetroffen wordt nihil is.

Veel databases bevatten ontbrekende informatie, denk aan incompleet ingevulde formulieren of foutieve metingen bij DNA-analyse. We introduceren drie methoden die, gebruik makend van compressie, zeer accurate schattingen voor de missende waarden geven. Hierbij worden specifiek de karakteristieken van de database gevolgd en behouden, waardoor vervolganalyse van de aangevulde gegevens zoveel mogelijk vrij is van verstoringen.

Kort gezegd, dit proefschrift beschrijft hoe een klein aantal, zeer interessante, patronen gevonden kan worden. Deze groepen kunnen gemakkelijk door experts geinspecteerd worden en bieden gedetailleerd inzicht in de te analyseren gegevens. Bovendien toont de succesvolle toepassing van deze patronen, in vijf verschillende data mining problemen, aan dat ze met recht *bruikbaar* genoemd mogen worden.

Dankwoord

Het is cliché, en waar, promoveren doe je niet in je eentje. Daarom zou ik iedereen die linksom, rechtsom of zijdelings bij mijn promotie betrokken is geweest heel graag willen bedanken: zonder jullie had het nooit zo'n leuke en leerzame tijd geworden.

Beste Arno, ik had me geen betere promotor of begeleider kunnen wensen. Net zo makkelijk als dat je ooit een 'leuk-garantie' durfde te geven voor een vak waarvan je de inhoud zelf nog niet bepaald had, had je dat voor mijn promotieonderzoek kunnen doen. In de afgelopen vier jaar heb ik van de geboden vrijheid genoten, veel geleerd en aan je commentaar gehad. Dank je wel.

I would like to thank the members of the reading committee, Heikki Mannila, Johannes Fürnkranz, Peter Grünwald, Toon Calders and Linda van der Gaag for carefully reading, considering and approving my thesis.

Beste (voorheen) leden van de ADA (voorheen LDD) groep, Ad, Hans, Lennart, Arno, Edwin, Ronnie, Carsten, Rainer, Subianto, Jeroen, Nicola, Diyah, dank jullie wel voor het bieden van een heel gezellige en productieve sfeer, als wel het inzien van het belang van ontspanning. Zonder jullie had ik nooit zo veel kunnen oefenen in darts, tafeltennis, schaken, de introquiz of whatthemovie. De schoolboekenborrel, geboden oppertunities en vele whisky's staan in m'n geheugen gegrift.

Beste Arne, naast een hele goede vriend was je een super-kamergenoot. Ik wil je in het bijzonder heel graag bedanken voor alle gekkigheid. Canons fluiten of stoelskwieken is en blijft teamsport. Dat A113 over een tijdje maar een come-back van jewelste mag beleven.

Matthijs, wat moet ik zeggen, na tig jaar vrienden, studiegenoten, collega's en co-auteurs te zijn? Genoeg is genoeg? Ja, wat niezen betreft, ja. Verder, nee, echt niet. Laten we nog veel mooie papers schrijven en biri's drinken. Wat die come-back betreft, ik reken op je.

I would like to thank my Finnish co-authors. Dear Heikki, thank you for inviting me to Helsinki. Dear Hannes, many thanks for showing me around town, a fruitful research project and providing me the opportunity to swim in a nearly frozen Baltic. Dear Nikolaj, you are a theory monster. Thanks for the collaboration, hopefully many may follow.

Maar bovenal wil ik mijn familie heel erg bedanken. Lieve Ieke, je bent een topzus. Lieve paps en mams, dank jullie wel voor alle steun, zorg, alles.

Antwerpen, Oktober 2009

Curriculum Vitae

Jilles Vreeken was born in Amsterdam on the 21st of March in 1981. He got his shoelace tying diploma in 1985, swimming levels A and B in resp. 1986 and 1987. In 1999 he graduated from O.S.G. Broklede in Breukelen at the athenaeum level. He studied computer science ('informatica') at Universiteit Utrecht and graduated with honours in 2004. During his studies, in 2003, he passed his drivers license test. In 2005 he started his Ph.D. studies at the Large Distributed Databases group of professor Arno Siebes.

Jilles is interested in photography, enjoys travelling to remote locations, has a rather broad taste and collection of music and is surprisingly good at guessing the movie title from stills of movies he hasn't seen yet. He greatly disliked writing his master thesis. Much to his surprise, he disliked writing his Ph.D. thesis many times less. However, now that it is finished, he very much looks forward to doing fresh research again, as that is both his favourite mind state, hobby and job.

SIKS Dissertation Series

1998	1	J. van den Akker (CWI), DEGAS - An Active, Temporal Database of Autonomous Objects
	2	F. Wiesman (UM), Information Retrieval by Graphically Browsing Meta-Information
	3	A.N.S. Steuten (TUD), A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective
	4	D. Breuker (UM), Memory versus Search in Games
	5	E.W. Oskamp (RUL), Computerondersteuning bij Straftopmeting
1999	1	M. Sloof (VU), Physiology of Quality Change Modelling
	2	R. Potharst (EUR), Classification using decision trees and neural nets
	3	D. Beal (UM), The Nature of Minimax Search
	4	J. Penders (UM), The practical Art of Moving Physical Objects
	5	A. de Moor (KUB), Empowering Communities
	6	N.J.E. Wijngaards (VU), Re-design of compositional systems
	7	D. Spelt (UT), Verification support for object database design
	8	J.H.J. Lenting (UM), Informed Gambling: Conception and Analysis for Discrete Reallocation
2000	1	F. Niessink (VU), Perspectives on Improving Software Maintenance
	2	K. Holtman (TUE), Prototyping of CMS Storage Management
	3	C.M.T. Metselaar (UVA), Sociaal-organisatorische gevolgen van kennistechnologie
	4	G. de Haan (VU), ETAG, A Formal Model of Competence Knowledge for User Interface Design
	5	R. van der Pol (UM), Knowledge-based Query Formulation in Information Retrieval
	6	R. van Eijk (UU), Programming Languages for Agent Communication
	7	N. Peek (UU), Decision-theoretic Planning of Clinical Patient Management
	8	V. Coupé (EUR), Sensitivity Analyis of Decision-Theoretic Networks
	9	F. Waas (CWI), Image DBMS design considerations, algorithms and architecture
	11	J. Karlsson (CWI), Scalable Distributed Data Structures for Database Management
2001	1	S. Renooij (UU), Qualitative Approaches to Quantifying Probabilistic Networks
	2	K. Hindriks (UU), Agent Programming Languages: Programming with Mental Models
	3	M. van Someren (UVA), Learning as problem solving
	4	E. Smirnov (UM), Conjunctive and disjunctive version spaces with boundary sets
	5	J. van Ossenbruggen (VU), Processing Structured Hypermedia: A Matter of Style
	6	M. van Welie (VU), Task-based User Interface Design
	7	B. Schonhage (VU), Diva: Architectural Perspectives on Information Visualization
	8	P. van Eck (VU), A Compositional Semantic Structure for Multi-Agent Systems Dynamics
	9	P.J. 't Hoen (RUL), Towards distributed development of large object-oriented models
	10	M. Sierhuis (UVA), Modeling and Simulating Work Practice BRAHMS
	11	T.M. van Engers (VU), Knowledge management: mental models in business systems design
2002	1	N. Lassing (VU), Architecture-Level Modifiability Analysis
	2	R. van Zwol (UT), Modelling and Searching Web-based Document Collections
	3	H.E. Blok (UT), Database Optimization Aspects for Information Retrieval
	4	J.R. Castelo Valdueza (UU), The Discrete Acyclic Digraph Markov Model in Data Mining
	5	R. Serban (VU), The Private Cyberspace
	6	L. Mommers (UL), Applied legal epistemology
	7	P. Boncz (CWI), Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications
	8	J. Gordijn (VU), Value-based requirements engineering
	9	W-J. van den Heuvel (KUB), Integrating business applications with objectified legacy systems
	10	B. Sheppard (UM), Towards Perfect Play of Scrabble
	11	W.C.A. Wijngaards (VU), Agent based modelling of dynamics
	12	A. Schmidt (UVA), Processing XML in Database Systems
	13	H. Wu (TUE), A Reference Architecture for Adaptive Hypermedia Applications
	14	W. de Vries (UU), Agent Interaction
	15	R. Eshuis (UT), Semantics and Verification of UML Activity Diagrams for Workflow Modelling
	16	P. van Langen (VU), The Anatomy of Design: Foundations, Models and Applications

	17	S. Manegold (UVA), Understanding, modeling, and improving main-memory db performance
2003	1	H. Stuckenschmidt (VU), Ontology-based information sharing in weakly structd environments
	2	J. Broersen (VU), Modal Action Logics for Reasoning About Reactive Systems
	3	M. Schuemie (TUD), Human-computer interaction and presence in VR exposure therapy
	4	M. Petkovic (UT), Content-Based Video Retrieval Supported by Database Technology
	5	J. Lehmann (UVA), Causation in Artificial Intelligence and Law - A modelling approach
	6	B. van Schooten (UT), Development and Specification of Virtual Environments
	7	M. Jansen (UVA), Formal Explorations of Knowledge Intensive Tasks
	8	Y. Ran (UM), Repair Based Scheduling
	9	R. Kortmann (UM), The Resolution of Visually Guided Behaviour
	10	A. Lincke (UT), Some experimental studies on medium, innovation context and culture
	11	S. Keizer (UT), Reasoning under uncertainty in natural language dialogue
	12	R. Ordelman (UT), Dutch Speech Recognition in Multimedia Information Retrieval
	13	J. Donkers (UM), Nosce Hostem - Searching with Opponent Models
	14	S. Hoppenbrouwers (KUN), Freezing Language
	15	M. de Weerdt (TUD), Plan Merging in Multi-Agent Systems
	16	M. Windhouwer (CWI), Feature grammar systems
	17	D. Jansen (UT), Extensions of Statecharts with Probability, Time, and Stochastic Timing
	18	L. Kocsis (UM), Learning Search Decisions
2004	1	V. Dignum (UU), A Model for Organizational Interaction: Based on Agents, Founded in Logic
	2	L. Xu (UT), Monitoring Multi-party Contracts for E-business
	3	P. Groot (VU), A theoretical and empirical analysis of approx in symbolic problem solving
	4	C. van Aart (UVA), Organizational Principles for Multi-Agent Architectures
	5	V. Popova (EUR), Knowledge Discovery and Monotonicity
	6	B-J. Hommes (TUD), The Evaluation of Business Process Modeling Techniques
	7	E. Boltjes (UM), Voorbeeldig onderwijs
	8	J. Verbeek (UM), Politie en de Nieuwe Internationale Informatiemarkt
	9	M. Caminada (VU), For the Sake of the Argument
	10	S. Kabel (UVA), Knowledge-rich Indexing of Learning-objects
	11	M. Klein (VU), Change Management for Distributed Ontologies
	12	T. Duy Bui (UT), Creating Emotions and Facial Expressions for Embodied Agents
	13	W. Jamroga (UT), Using Multiple Models of Reality: On Agents who Know how to Play
	14	P. Harrenstein (UU), Logic in Conflict. Logical Explorations in Strategic Equilibrium
	15	A. Knobbe (UU), Multi-Relational Data Mining
	16	F. Divina (VU), Hybrid Genetic Relational Search for Inductive Learning
	17	M. Winands (UM), Informed Search in Complex Games
	18	V. Bessa Machado (UVA), Supporting the Construction of Qualitative Knowledge Models
	19	T. Westerveld (UT), Using Generative Probabilistic Models for Multimedia Retrieval
	20	M. Evers (Nyenrode), Learning from Design: Facilitating Multidisciplinary Design Teams
2005	1	F. Verdenius (UVA), Methodological Aspects of Designing Induction-Based Applications
	2	E. van der Werf (UM), AI techniques for the Game of Go
	3	F. Grootjen (RUN), A Pragmatic Approach to the Conceptualisation of Language
	4	N. Meratnia (UT), Towards Database Support for Moving Object data
	5	G. Infante-Lopez (UVA), Two-Level Probabilistic Grammars for Natural Language Parsing
	6	P. Spronck (UM), Adaptive Game AI
	7	F. Frasincar (TUE), Hypermedia presentation generation for semantic web systems
	8	R. Vdovjak (TUE), A model-driven approach for building ontology-based web applications
	9	J. Broekstra (VU), Storage, Querying and Inferencing for Semantic Web Languages
	10	A. Bouwer (UVA), Explaining behaviour: using qualitative simulation in interactive learning
	11	E. Ogston (VU), Agent Based Matchmaking and Clustering
	12	C. Boer (EUR), Distributed Simulation in Industry
	13	F. Hamburg (UL), Een Computermodel voor het ondersteunen van euthanasiebeslissingen
	14	B. Omelayenko (VU), Web-Service configuration on the Semantic Web
	15	T. Bosse (VU), Analysis of the Dynamics of Cognitive Processes
	16	J. Graaumans (UU), Usability of XML Query Languages
	17	B. Shishkov (TUD), Software Specification Based on Re-usable Business Components
	18	D. Sent (UU), Test-selection Strategies for Probabilistic Networks
	19	M. van Dartel (UM), Situated Representation
	20	C. Coteanu (UL), Cyber Consumer Law, State of the Art and Perspectives
	21	W. Derks (UT), Improving concurrency and recovery in DBMS by exploiting semantics
2006	1	S. Angelov (TUE), Foundations of B2B Electronic Contracting
	2	C. Chisalita (VU), Contextual issues in the design and use of IT in organizations
	3	N. Christoph (UVA), The role of Metacognitive Skills in Learning to Solve Problems
	4	M. Sabou (VU), Building Web Service Ontologies
	5	C. Pierik (UU), Validation Techniques for Object-Oriented Proof Outlines
	6	Z. Baida (VU), Software-aided service bundling
	7	M. Smiljanic (UT), XML schema matching - balancing efficiency and effectiveness
	8	E. Herder (UT), Forward, Back and Home Again - Analyzing User Behavior on the Web
	9	M. Wahdan (UM), Automatic Formulation of the Auditor's Opinion

	10	R. Siebes (VU), Semantic Routing in Peer-to-Peer Systems
	11	J. van Ruth (UT), Flattening Queries over Nested Data Types
	12	B. Bongers (VU), Interactivation - towards an e-cology of people, our t-environment, and arts
	13	H-J. Lebbink (UU), Dialogue and Decision Games for Information Exchanging Agents
	14	J. Hoorn (VU), Software requirements: update, upgrade, redesign
	15	R. Malik (UU), CONAN: Text Mining in the Biomedical Domain
	16	C. Riggelsen (UU), Approximation Methods for Efficient Learning of Bayesian Networks
	17	S. Nagata (UU), User Assistance for Multitasking with Interruptions on a Mobile Device
	18	V. Zhizhkun (UVA), Graph transformation for Natural Language Processing
	19	B. van Riemsdijk (UU), Cognitive Agent Programming: A Semantic Approach
	20	M. Velikova (UvT), Monotone models for prediction in data mining
	21	B. van Gils (RUN), Aptness on the Web
	22	P. de Vrieze (RUN), Fundaments of Adaptive Personalisation
	23	I. Juvina (UU), Development of Cognitive Model for Navigating on the Web
	24	L. Hollink (VU), Semantic Annotation for Retrieval of Visual Resources
	25	M. Drugan (UU), Conditional log-likelihood MDL and Evolutionary MCMC
	26	V. Mihajlovic (UT), Score region algebra: a flexible framework for structured IR
	27	S. Bocconi (CWI), Vox Populi: generating video documentaries from semantically annotated media repositories
	28	B. Sígurbjørnsson (UVA), Focused Information Access using XML Element Retrieval
2007	1	K. Leune (UvT), Access Control and Service-Oriented Architectures
	2	W. Teepe (RUG), Reconciling Information Exchange and Confidentiality: A Formal Approach
	3	P. Mika (VU), Social Networks and the Semantic Web
	4	J. van Diggelen (UU), Achieving Semantic Interoperability in Multi-agent Systems
	5	B. Schermer (UL), Software Agents, Surveillance, and the Right to Privacy
	6	G. Mishne (UVA), Applied Text Analytics for Blogs
	7	N. Jovanovic (UT), To whom it may concern - addressee identification in face-to-face meetings
	8	M. Hoogendoorn (VU), Modeling of Change in Multi-Agent Organizations
	9	D. Mobach (VU), Agent-Based Mediated Service Negotiation
	10	H. Aldewereld (UU), Autonomy vs. conformity: a perspective on norms and protocols
	11	N. Stash (TUE), Incorporating cognitive learning styles in a adaptive hypermedia system
	12	M. van Gerven (RUN), Bayesian Networks for Clinical Decision Support
	13	R. Rienks (UT), Meetings in Smart Environments; Implications of Progressing Technology
	14	N. Bergboer (UM), Context-Based Image Analysis
	15	J. Lacroix (UM), NIM: a Situated Computational Memory Model
	16	D. Grossi (UU), Designing Invisible Handcuffs
	17	T. Charitos (UU), Reasoning with Dynamic Networks in Practice
	18	B. Orriens (UvT), On the development an management of adaptive business collaborations
	19	D. Levy (UM), Intimate relationships with artificial partners
	20	S. Roijackers (UU), Customer configuration updating in a software supply network
	21	K. Vermaas (UU), Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005
	22	Z. Zlatev (UT), Goal-oriented design of value and process models from patterns
	23	P. Barna (TUE), Specification of Application Logic in Web Information Systems
	24	G. Ramírez Camps (CWI), Structural Features in XML Retrieval
	25	J. Schalken (VU), Empirical Investigations in Software Process Improvement
2008	1	K. Boer-Sorbán (EUR), Agent-Based Simulation of Financial Markets
	2	A. Sharpanskykh (VU), On computer-aided methods for modeling and analysis of organizations
	3	V. Hollink (UVA), Optimizing hierarchical menus: a usage-based approach
	4	A. de Keijzer (UT), Management of Uncertain Data - towards unattended integration
	5	B. Mutschler (UT), Modeling and simulating causal dependencies from a cost perspective
	6	A. Hommersom (RUN), On the Application of Formal Methods to Clinical Guidelines
	7	P. van Rosmalen (OU), Supporting the tutor in the design and support of adaptive e-learning
	8	J. Bolt (UU), Bayesian Networks: Aspects of Approximate Inference
	9	C. van Nimwegen (UU), The paradox of the guided user: assistance can be counter-effective
	10	W. Bosma (UT), Discourse oriented summarization
	11	V. Kartseva (VU), Designing Controls for Network Organizations: A Value-Based Approach
	12	J. Farkas (RUN), A Semiotically Oriented Cognitive Model of Knowledge Representation
	13	C. Carraciolo (UVA), Topic Driven Access to Scientific Handbooks
	14	A. van Bunningen (UT), Context-Aware Querying; Better Answers with Less Effort
	15	M. van Otterlo (UT), The Logic of Adaptive Behavior
	16	H. van Vugt (VU), Embodied agents from a user's perspective
	17	M. Op 't Land (TUD), Applying Architecture and Ontology to Enterprises
	18	G. de Croon (UM), Adaptive Active Vision
	19	H. Rode (UT), From Document to Entity Retrieval
	20	R. Arendsen (UVA), Geen bericht, goed bericht
	21	K. Balog (UVA), People Search in the Enterprise
	22	H. Koning (UU), Communication of IT-Architecture
	23	S. Visscher (UU), Bayesian network models for ventilator-associated pneumonia
	24	Z. Aleksovski (VU), Using background knowledge in ontology matching

	25	G. Jonker (UU), Efficient and Equitable Exchange in Air Traffic Management
	26	M. Huijbregts (UT), Segmentation, diarization and speech transcription
	27	H. Vogten (OU), Design and Implementation Strategies for IMS Learning Design
	28	I. Flesch (RUN), On the Use of Independence Relations in Bayesian Networks
	29	D. Reidsma (UT), Annotations and Subjective Machines
	30	W. van Atteveldt (VU), Semantic Network Analysis
	31	L. Braun (UM), Pro-Active Medical Information Retrieval
	32	T.H. Bui (UT), Toward affective dialogue management using markov decision processes
	33	F. Terpstra (UVA), Scientific Workflow Design; theoretical and practical issues
	34	J. De Knijf (UU), Studies in Frequent Tree Mining
	35	B.T. Nielsen (UvT), Dendritic morphologies: function shapes structure
2009	1	R. Jurgelenaite (RUN), Symmetric Causal Independence Models
	2	W.R. van Hage (VU), Evaluating Ontology-Alignment Techniques
	3	H. Stol (UvT), A Framework for Evidence-based Policy Making Using IT
	4	J. Nabukenya (RUN), Improving the Quality of Organisational Policy Making
	5	S. Overbeek (RUN), Bridging Supply and Demand for Knowledge Intensive Tasks
	6	M. Subianto (UU), Understanding Classification
	7	R. Poppe (UT), Discriminative Vision-Based Recovery and Recognition of Human Motion
	8	V. Nannen (VU), Evolutionary Agent-Based Policy Analysis in Dynamic Environments
	9	B. Kanagwa (RUN), Design, Discovery and Construction of Service-oriented Systems
	10	J.A.N. Wielemaker (UVA), Logic programming for knowledge-intensive applications
	11	A. Boer (UVA), Legal Theory, Sources of Law & the Semantic Web
	12	P. Massuthe (TUE, Humboldt-Universitaet zu Berlin), Operating Guidelines for Services
	13	S. de Jong (UM), Fairness in Multi-Agent Systems
	14	M. Korotkiy (VU), From ontology-enabled services to service-enabled ontologies
	15	R. Hoekstra (UVA), Ontology Representation - Ontologies that Make Sense
	16	F. Reul (UvT), New Architectures in Computer Chess
	17	L. van der Maaten (UvT), Feature Extraction from Visual Data
	18	F. Groffen (CWI), Armada, An Evolving Database System
	19	V. Robu (CWI), Modeling prefs, strategy and collaboration in agent-mediated e-markets
	20	B. van der Vecht (UU), Adjustable Autonomy: Controling Influences on Decision Making
	21	S. Vanderlooy (UM), Ranking and Reliable Classification
	22	P. Serdyukov (UT), Search For Expertise: Going beyond direct evidence
	23	P. Hofgesang (VU), Modelling Web Usage in a Changing Environment
	24	A. Heuvelink (VUA), Cognitive Models for Training Simulations
	25	A. van Ballegooij (CWI), RAM: Array Database Management through Relational Mapping
	26	F. Koch (UU), An Agent-Based Model for the Development of Intelligent Mobile Services
	27	C. Glahn (OU), Contextual Support of social Engagement and Reflection on the Web
	28	S. Evers (UT), Sensor Data Management with Probabilistic Models
	29	S. Pokraev (UT), Model-Driven Semantic Integration of Service-Oriented Applications
	30	M. Zukowski (CWI), Balancing vectorized query execution with bandwidth-optimized storage
	31	S. Katrenko (UVA), A Closer Look at Learning Relations from Text
	32	R. Farenhorst and R. de Boer (VU), Architectural Knowledge Management
	33	K. Truong (UT), How Does Real Affect^3 Recognition In Speech?
	34	I. van de Weerd (UU), Advancing in Software Product Management
	35	W. Koelewijn (UL), Privacy en Politiegegevens
	36	M. Kalz (OUN), Placement Support for Learners in Learning Networks
	37	H. Drachsler (OUN), Navigation Support for Learners in Informal Learning Networks
	38	R. Vuorikari (OU), Tags and self-organisation
	39	C. Stahl (TUE), Service Substitution – A Behavioral Approach Based on Petri Nets
	40	S. Raaijmakers (UvT), Multinomial Language Learning
	41	I. Berezhnyy (UvT), Digital Analysis of Paintings
	42	T. Bogers (UvT), Recommender Systems for Social Bookmarking
	43	V.N.L. Franqueira (UT), Finding Multi-step Attacks in Computer Networks
	44	R.S. Tapia (UT), Assessing Business-IT Alignment in Networked Organizations
	45	J. Vreeken (UU), Making Pattern Mining Useful
2010	1	M. van Leeuwen (UU), Patterns that Matter