Universiteit Antwerpen

Summarizing Data with Informative Patterns

Proefschrift

voorgelegd tot het behalen van de graad van
doctor in de wetenschappen: informatica
aan de Universiteit Antwerpen
te verdedigen door

Michael MAMPAEY

Promotor: Prof. dr. Bart Goethals

Antwerpen, 2011

Summarizing Data with Informative Patterns
Nederlandse titel: *Databanken Samenvatten met Informatieve Patronen*

The research in this thesis was supported by a Ph.D. grant of the Agency for Innovation by Science and Technology in Flanders (IWT).

Typeset in Palatino using LaTeX

ISBN 978-90-5728-348-2
D/2011/12.293/39

sum•ma•rize /ˈsʌməraɪz/ *v.* To make (or constitute) a summary of; to sum up; to state briefly or succinctly.

Oxford English Dictionary

Acknowledgements

THROUGHOUT THE PAST FIVE YEARS, many people have contributed either directly or indirectly to this dissertation, and I would like to take the opportunity to thank them. First of all, I am grateful to my advisor, Bart Goethals, who got me interested in doing a Ph.D. in Data Mining in the first place, for our discussions, scientific and otherwise. The organization of the *ECML PKDD* conference in Antwerp was also a memorable experience. Special thanks goes to the people that I have had the pleasure of collaborating with; Wim Le Page, Nikolaj Tatti, and Jilles Vreeken. You played no small part in this thesis coming to fruition. Thanks is due to the members of the doctoral jury: Geoff Webb, Arno Siebes, Toon Calders, Floris Geerts, Jan Paredaens, and Bart Goethals. Further, I would also like to thank all members—past and present—of the ADReM research group and the Department of Mathematics and Computer Science who made my stay at the University of Antwerp as pleasant as it was; Adriana, Álvaro, Antonio, Bart, Boris, Calin, Céline, Floris, Hai, Jan, Jan, Jeroen, Jeroen, Jilles, Joris, Juan, Koen, Kris, Mehmet, Nele, Nghia, Nikolaj, Olaf, Philippe, Roel, Sandy, Tayena, and Wim. Finally, I would like to thank my family and friends for their support.

Michael Mampaey
Antwerp, October 2011

Summary

A short summary of this dissertation is presented below. It was obtained by applying the MTV algorithm (Chapter 6) to the full text of the thesis, where each stemmed word was regarded as an item, and every sentence as a transaction. Stop words, numbers, mathematical symbols, and formulae were ignored. The resulting dataset consists of 2 231 sentences, and 23 855 words of which 1 861 are unique.

❧

description length
maximum entropy model
background knowledge
row column margin
search space
code table
log likelihood
lazarus count
frequency estimate
minimum description length principle
p value
attribute cluster
frequent itemset mining
itemset collection
take account

cluster description
MDL BIC
penalty term
Markov chain
redundant rule
mutual information
absolute error
non redundant
binary categorical
relative error
random swap
itemset support
high quality
itemset frequency
dependence rule
association rule
low entropy set
independence model
MDL principle
refined MDL
well known
pattern mining
compute probability
correlated attribute
maximum entropy distribution
model likelihood
section discuss
count statistic
code length
identify best
block size
attribute set

List of Publications

- T. Calders, B. Goethals, and M. Mampaey. Mining itemsets in the presence of missing values. In *Proceedings of the 22nd ACM Symposium on Applied Computing (ACM SAC), Seoul, Korea*, pages 404–408. ACM, 2007.

- B. Goethals, W. Le Page, and M. Mampaey. Mining interesting sets and rules in relational databases. In *Proceedings of the 25th ACM Symposium on Applied Computing (ACM SAC), Sierre, Switzerland*. ACM, 2010.

- N. Tatti and M. Mampaey. Using background knowledge to rank itemsets. *Data Mining and Knowledge Discovery*, 21(2):293–309, 2010.

- M. Mampaey and J. Vreeken. Summarising data by clustering items. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Barcelona, Spain*, pages 321–336. Springer-Verlag, 2010.

- M. Mampaey. Mining non-redundant information-theoretic dependencies between itemsets. In *Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery (DaWaK), Bilbao, Spain*, volume 6263 of *LNCS*, pages 130–141. Springer, 2010.

- M. Mampaey, N. Tatti, and J. Vreeken. Tell me what I need to know: Succinctly summarizing data with itemsets. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, pages 573–581. ACM, 2011. *(Best Student Paper)*

- M. Mampaey, N. Tatti, and J. Vreeken. Data summarization with informative itemsets. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC), Ghent, Belgium*, 2011.
- M. Mampaey and J. Vreeken. Summarizing categorical data by clustering attributes. Manuscript currently submitted to: *Data Mining and Knowledge Discovery*, 2011.
- M. Mampaey, J. Vreeken, and N. Tatti. Summarizing data succinctly with the most informative itemsets. Manuscript currently submitted to: *Transactions on Knowledge Discovery and Data Mining*, 2011.

Contents

Chapter 1

Introduction

During the past few decades, technological advances have made it possible to cheaply acquire and store vast quantities of data of various kinds; for instance, scientific observations, commercial records, medical data, government databases, multimedia, etc. Gathering such data is vital to gain insight into a problem, to study a certain phenomenon, or to be able to make informed decisions. Further, it is an integral part of the scientific process. Nowadays, however, we have such vast amounts of data at our disposal, that extracting information from it has become a nontrivial task. The analysis of such data by hand is certainly no longer tractable. This situation gives rise to the need for computational techniques that extract useful and meaningful information from databases.

Knowledge discovery from data (KDD) has therefore rapidly emerged as an important area of research. Particularly the field of data mining, a key component in the knowledge discovery process, has gained a lot of momentum. Located at the intersection of statistics, artificial intelligence, and database research, data mining is defined by Hand et al. [2001] as follows.

> Data mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner.

Several important (albeit vague) terms appear in this definition. The term *relationships* is very broad, and can refer to any regularity or structure of

the data in whatever form; this may include anything from local patterns to classification models or clusterings. The keyword *unsuspected* is a crucial one, as the aim of data mining is to discover relationships that are interesting, surprising, or previously unknown. This inherently assumes that the data analyst has some expectations about the data, which may or may not exist in an explicit form. Any deviation from this expectation is deemed interesting, and possibly worth a closer look; on the other hand, if a regularity in the data completely conforms to the user's expectations, it is indeed quite boring. We can regard this expectation as the background knowledge of the user.

Summarizing the data means to present its characteristics to the data miner in a succinct and concise manner, forming a small yet informative and manageable piece of information. Ideally it should contain just the information that the user needs, nothing more and nothing less. However, the results of existing data mining techniques can often be quite complex and detailed, or contain redundant information. For instance, in frequent itemset mining the user can be overwhelmed with countless patterns. Finally, the outcome of a data mining method should be *understandable*, meaning that it must be intuitive for an analyst to grasp what the result signifies, such that her or she is then able to take advantage of the acquired knowledge.

Data mining techniques can roughly be divided into two categories: *local* and *global* ones. Global techniques aim to construct a model of the data as a whole. Such models typically do not capture the data perfectly, but instead try to describe its overall tendencies in order to describe or understand it, or to be able to make predictions. Examples include clustering, classification and regression, graphical modeling, or fitting the parameters of a Gaussian distribution. Local techniques, on the other hand, describe particular aspects of the data that are interesting in some way. Such local structures (e.g., frequent itemsets), describe only a part of the data, rather than modeling it in its entirety. An advantage is that they are often easy to interpret and not difficult to mine. Typically, however, it is nearly impossible to consider *all* interesting patterns in a dataset, for the simple reason that there may be far too many of them to handle. It has been recognized that local pattern mining in practice should either be accompanied by a pruning, ranking, or filtering step, which drastically reduces the set of results while trying to retain informativeness; or one should directly mine sets of patterns as a whole, by considering how they relate to each other. This distinction between local and global data mining techniques is not always a clear-cut one, though.

The aim and subject of this thesis is to investigate how local patterns can be used to summarize data. To this end, we put forth some key concepts that a summary should adhere to.

- **Informativeness** It goes without saying that a summary should convey meaningful and useful information to a user, that is, a summary should communicate significant structures that are present in the data. As such, information theory plays an important theory in this dissertation to measure interestingness. Further, we will also focus on taking the data miner's background knowledge into account, since what is interesting to one person might not be so to another.

- **Conciseness** A summary must necessarily be succinct in order for a user to be able to oversee and manage it. This conciseness may simply refer the size of the summary, but it may also refer to its complexity.

- **Interpretability** Clearly, a concise black box method that can accurately capture a given dataset, does not allow us to understand the data any better. Therefore, we also list interpretability as a requirement. Local patterns generally tend to be quite easy to understand. In this work we mostly focus on two types of patterns: itemsets and attribute sets, and this for binary and categorical data.

In a sense, a summary can be seen as a model of the data, although it is not necessarily queryable or presented in a mathematical form. The summarization approaches presented in this dissertation should be categorized as *exploratory data mining*; our intent is to extend our knowledge of some given dataset that we do not yet know much about. This is an inherently imprecise problem with no clearly defined objective or a single correct solution. The goal of this thesis is therefore not to develop one all-embracing summarization method. Rather, we will examine several approaches, since in practice we may be confronted with different types of data and applications. Using different ways to look at data, with different patterns, criteria, etc, can be more insightful than using a single technique.

At this point, we remark the distinction between summarizing a given dataset, and summarizing a collection of patterns. While many techniques exist that start from a given collection of patterns, i.e., the *result* of a data mining method, often to be able to reconstruct the original pattern set, our aim is to directly summarize the data itself, making use of local patterns.

1.1 Thesis Outline

This thesis is organized in seven chapters. Chapters 3 through 6 are based on previously published work. The outline of the thesis is as follows.

Chapter 2 introduces preliminary definitions and notations regarding pattern mining, information theory, and model selection. These preliminaries form the basis for the subsequent chapters.

Chapter 3 presents an approach for discovering dependence rules between sets of attributes using information-theoretic measures. We investigate techniques to reduce the redundancy in the set of results, and present an efficient algorithm to mine these rules.

The content of this chapter is based on:
M. Mampaey. Mining non-redundant information-theoretic dependencies between itemsets. In *Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery (DaWaK), Bilbao, Spain*, volume 6263 of *LNCS*, pages 130–141. Springer, 2010.

Chapter 4 provides a technique to create a high-level overview of a dataset, by clustering its attributes into correlated groups. For each attribute cluster, a small distribution gives a description of the corresponding part of the data. These groupings can subsequently be used as an approximative model for the data. To find good attribute clusterings, we employ the Minimum Description Length principle.

The content of this chapter is based on:
M. Mampaey and J. Vreeken. Summarising data by clustering items. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Barcelona, Spain*, pages 321–336. Springer-Verlag, 2010.
M. Mampaey and J. Vreeken. Summarizing categorical data by clustering attributes. Manuscript currently submitted to: *Data Mining and Knowledge Discovery*, 2011.

Chapter 5 studies the usage of intuitive statistics to construct a background model for ranking itemsets. These statistics include row and column

margins, lazarus counts, and transaction bounds. Based on these statistics, a Maximum Entropy model is presented, which can be constructed and queried efficiently in polynomial time, and against which itemsets are ranked.

The content of this chapter is based on:
N. Tatti and M. Mampaey. Using background knowledge to rank itemsets. *Data Mining and Knowledge Discovery*, 21(2):293–309, 2010.

Chapter 6 introduces a novel algorithm to mine a collection of itemsets as a global model for data. The proposed algorithm models the data using a Maximum Entropy distribution which is built based on a succinct itemset collection. Techniques from Chapter 4 to ensure efficient computation, and from Chapter 5 to infuse background knowledge, are incorporated into the algorithm. To discover the best collection of itemsets that summarizes a given dataset, we employ the Bayesian Information Criterion and the Minimum Description Length principle.

The content of this chapter is based on:
M. Mampaey, N. Tatti, and J. Vreeken. Tell me what I need to know: Succinctly summarizing data with itemsets. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, pages 573–581. ACM, 2011. *(Best Student Paper)*
M. Mampaey, J. Vreeken, and N. Tatti. Summarizing data succinctly with the most informative itemsets. Manuscript currently submitted to: *Transactions on Knowledge Discovery and Data Mining*, 2011.

Chapter 7 summarizes the main contributions of the dissertation and ends with concluding remarks.

Chapter 2

Preliminaries

This chapter introduces some of the main concepts from pattern mining, information theory, and model selection techniques such as MDL. We establish preliminaries and notation as a basis for the subsequent chapters, and give some pointers for further reading.

2.1 Pattern Mining

One of the most prolific areas of research within the data mining community is without a doubt that of pattern mining. A pattern is any type of regularity that is observed in data that may or may not be of interest. The term pattern mining can generally refer to any type of pattern, and a variety of data formats; e.g., substructure mining in graph data, sequential pattern mining in temporal data, etc. Many interestingness measures have been defined for these patterns, and many efficient algorithms to discover them have been developed and applied in various practical situations.

The focus of this dissertation is on two particular types of patterns: itemsets and attribute sets. Besides this, we will also consider rules between them. Since the inception of frequent itemset mining (and association rule mining) by Agrawal et al. [1993], several hundreds of papers have been published on the subject, covering aspects ranging from the design of efficient algorithms, over the definition of measures of interestingness, to combatting redundancy and mining sets of patterns.

The by now classic textbook example of frequent itemset mining is that of supermarket basket analysis. In this setting we are given a product database consisting of *transactions* that represent the purchases made by the customers of a supermarket, and we wish to investigate their purchasing behavior. The problem of frequent itemset mining then, is to find all sets of items—or *itemsets*—that were frequently bought together. The management of the supermarket can subsequently use this acquired information to their advantage, for instance, by adjusting product pricing.

Definitions

Below we present two sets of notations and definitions that are commonly used in the itemset mining literature; the transactional notation for binary data, and the categorical notation. Both sets of notations are commonly used in the literature, and we will follow them in this thesis. Although we will occasionally abandon mathematical rigor in favor of notational convenience by using both notations interchangeably, in general it will be clear from the context what is meant.

Transactional notation

Let $\mathcal{I}$ be the set of all items. An *itemset* X is a subset of $\mathcal{I}$. For notational convenience, we shall denote an itemset $\{x, y, z\}$ briefly as xyz, and the union $X \cup Y$ of two itemsets as XY.

A *binary dataset* $\mathcal{D}$ is a set of *transactions*, where a transaction t is a pair $\langle tid, I \rangle$, consisting of a transaction identifier $tid \in \mathbb{N}$, and an itemset $I \subseteq \mathcal{I}$. The set of all possible transactions is denoted $\mathcal{T} = \mathbb{N} \times 2^{\mathcal{I}}$, or often just $\mathcal{T} = 2^{\mathcal{I}}$, disregarding the identifiers for simplicity. The *empirical distribution* over $\mathcal{T}$ defined by a dataset $\mathcal{D}$ is denoted $q_{\mathcal{D}}$. A transaction t is said to *contain* or *support* an itemset X, written as $X \subseteq t$, if for all items x in X it holds that $x \in t.I$. The *cover* of an itemset is defined as the set of *tid*s of the transactions that contain it,

$$cover(X, \mathcal{D}) = \{t.tid \mid t \in \mathcal{D} \text{ with } X \subseteq t\} .$$

The *support* of an itemset X in a dataset $\mathcal{D}$ is defined as the number of transactions in $\mathcal{D}$ containing X,

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

Similarly, the *frequency* of an itemset X in $\mathcal{D}$ is the relative number of transactions supporting X,

$$fr(X, \mathcal{D}) = \frac{supp(X, \mathcal{D})}{|\mathcal{D}|} .$$

An *association rule* between two itemsets X and Y is written as $X \Rightarrow Y$, where usually both sets are assumed to be either disjoint ($X \cap Y = \emptyset$) or inclusive ($X \subseteq Y$), depending on the context. The support of a rule $X \Rightarrow Y$ is defined as the support of the union of its antecedent and its consequent, $X \cup Y$. The *confidence* of an association rule $X \Rightarrow Y$ is defined as

$$conf(X \Rightarrow Y, \mathcal{D}) = \frac{supp(X \cup Y, \mathcal{D})}{supp(X, \mathcal{D})} .$$

In the definitions above, whenever the dataset $\mathcal{D}$ is clear from the context, we omit it from notation.

Categorical notation

Let $\mathcal{A} = \{a_1, \ldots, a_N\}$ be the set of attributes. Each attribute has a domain $dom(a)$, which is the set of possible values $\{v_1, v_2, \ldots\}$ it can assume. In this thesis, we only consider categorical attributes, i.e., attributes with a discrete and finite domain. Attributes whose domain equals $\{0, 1\}$ are called binary. The domain of a set of attributes $X \subseteq \mathcal{A}$ is simply the Cartesian product of the individual attributes' domains: $dom(X) = \prod_{a_i \in X} dom(a_i)$. A transaction t over $\mathcal{A}$ is a vector of length N. The i-th element of t is denoted as $t_i \in dom(a_i)$. An *item* is defined as an attribute-value pair $(a_i = v_i)$ where $v_i \in dom(a_i)$. For brevity, we write an itemset $\{(x_1 = v_1), \ldots, (x_k = v_k)\}$ as $\{(X = v)\}$. A transaction t is said to contain an itemset $\{(X = v)\}$, if for each $x_i \in X$ it holds that $t_i = v_i$. From here, we can define support, frequency, etc. as above.

Note that we can trivially transform a binary, transactional dataset into a categorical dataset, by constructing a binary attribute for each item, and setting it to 1 or 0 depending on its occurrence. In this case, frequent itemset mining is often restricted to *positive* items, i.e., $\mathcal{I} = \{(a_i = 1) \mid a_i \in \mathcal{A}\}$. We can consider *negative* items of the form $(a_i = 0)$ as well. An itemset also containing negative items is called a *generalized* itemset, and often written as, e.g., $\overline{x}y\overline{z}$ where x and z are negative, and y is positive.

Conversely, we can transform a categorical dataset into a transactional one, by constructing an item for each possible attribute-value pair in the categorical data.

Problem

The problem of frequent itemset mining (or FIM for short), is to find the collection $\mathcal{F}$ of all itemsets whose support is higher than a user-defined threshold $minsup \in (0, |\mathcal{D}|]$, or equivalently, all itemsets whose frequency is higher than a threshold $minfreq \in (0, 1]$. In other words, to find

$$\mathcal{F} = \{X \subseteq \mathcal{I} \mid supp(X, \mathcal{D}) \geq minsup\} .$$

The itemsets in $\mathcal{F}$ are called frequent. This problem is known to be **NP**-hard.

Choosing a value for the minimum support parameter mostly depends on the task and the dataset at hand. In general, however, it should be noted that choosing a value that is too high tends to produce few results, which often represent already known information. On the other hand, choosing a value that is too low results in far too many patterns for the user to handle. Finding a threshold value that achieves a proper balance can be tricky.

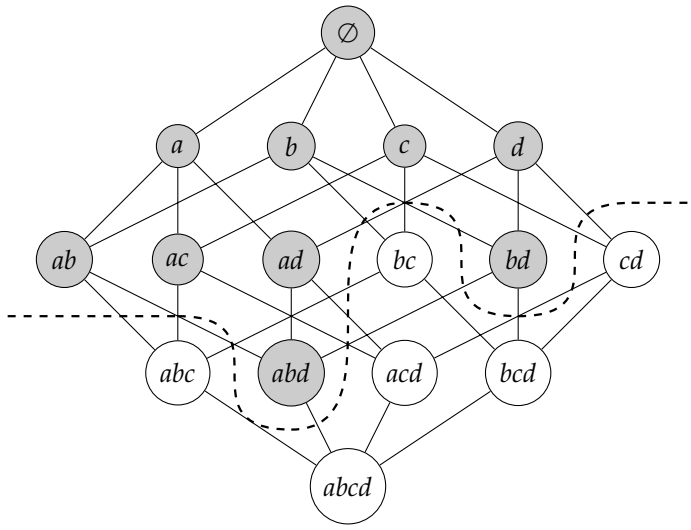
The problem of association rule mining is to find all rules whose confidence is higher than a user-defined threshold $minconf \in [0, 1]$, and for which the union of the antecedent and consequent is frequent. From the definition above, we see that the confidence of an association rule $X \Rightarrow Y$ is calculated from the supports of X and $X \cup Y$. Association rule mining is therefore usually implemented in two phases, by first mining all frequent itemsets, and then generating confident rules from them.

Besides support and confidence, many interestingness measures for itemsets and association rules have been proposed in the literature. For an overview, see, e.g., [Tan et al., 2004, Geng and Hamilton, 2006, Wu et al., 2007].

Frequent Itemset Mining Algorithms

Over the years, many frequent itemset mining algorithms have been proposed (see, e.g., [Goethals and Zaki, 2003, Bayardo et al., 2004]). These algorithms typically make use of the *monotonicity property*, which allows an efficient traversal of the search space.

Figure 2.1: A subset lattice over a set of four items $\{a, b, c, d\}$. In this example, the frequent itemsets are indicated in gray. The border, represented by the dashed line, separates the frequent from the infrequent itemsets.



Property 2.1 (Support monotonicity)**.** *Given two itemsets X and Y, it holds that*

$$X \subset Y \Rightarrow supp(X) \geq supp(Y) \ .$$

This property (also called the Apriori property) implies that if an itemset is infrequent, then all of its supersets are infrequent as well. Conversely, if an itemset is frequent, we know that all of its subsets are frequent as well.

The search space of all itemsets (the powerset of $\mathcal{I}$) forms a lattice [Davey and Priestley, 1990]. Figure 2.1 shows an example of a subset lattice for $\mathcal{I} = \{a, b, c, d\}$. The itemsets that are frequent (in a here unspecified dataset) are indicated in bold. As can be seen, the frequent itemsets all lie above the border represented by the dashed line. We can traverse this lattice efficiently using the Apriori property, in order to discover all frequent itemsets.

Below we present two famous frequent itemset mining algorithms, Apriori and Eclat. Some other well-known algorithms include fp-growth [Han et al., 2000] and lcm [Uno et al., 2004]. For in-depth surveys on frequent pattern mining, see, e.g., [Goethals, 2003, Han et al., 2007].

Apriori

The Apriori algorithm was independently proposed by Agrawal and Srikant [1994] and Mannila et al. [1994]. It is reproduced here are as Algorithm 2.1. Apriori is a breadth-first algorithm, which considers the search space in levels. First 1-itemsets (itemsets of size 1) are considered, then 2-itemsets, and so on. For each level k, a set C_k of candidate itemsets is constructed, based on the frequent itemsets from the previous level. That is, a particular itemset of size k is considered only if all of its subsets of size $k-1$ are frequent—otherwise we automatically know that it is infrequent because of the Apriori property (line 10). Each candidate is constructed by considering pairs of frequent k-itemsets with a common $(k-1)$-prefix (line 9). Then, for each candidate itemset, the data is scanned to determine its support, and the frequent ones are retained in the set $\mathcal{F}_k$ (line 5).

Algorithm 2.1: Apriori

input : a binary dataset $\mathcal{D}$;
a minimum support threshold *minsup*
output: the set of frequent itemsets in $\mathcal{D}$ w.r.t. *minsup*

1 $\mathcal{F} \leftarrow \emptyset$
2 $k \leftarrow 1$
3 $C_1 \leftarrow \{\{x\} \mid x \in \mathcal{I}\}$
4 **while** $C_k \neq \emptyset$ **do**
5 $\quad \mathcal{F}_k \leftarrow \{X \in C_k \mid supp(X, \mathcal{D}) \geq minsup\}$
6 $\quad \mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_k$
7 $\quad C_{k+1} \leftarrow \emptyset$
8 $\quad$ **for each** $X, Y \in \mathcal{F}_k$ such that $X[1 \ldots k-1] = Y[1 \ldots k-1]$ **and** $X[k] \leq Y[k]$ **do**
9 $\quad\quad Z \leftarrow X \cup Y[k]$
10 $\quad\quad$ **if** $\forall Z' \subset Z$ with $|Z'| = k$, it holds that $Z' \in \mathcal{F}_k$ **then**
11 $\quad\quad\quad C_{k+1} \leftarrow C_{k+1} \cup \{Z\}$
12 $\quad\quad$ **end**
13 $\quad$ **end**
14 $\quad k \leftarrow k+1$
15 **end**
16 **return** $\mathcal{F}$

While this pruning of candidates is very effective, the Apriori algorithm is quite demanding in terms of memory usage; a candidate set on a single level k can contain up to $\binom{N}{k}$ itemsets, which may exceed the available memory, especially for large and dense datasets. Several optimizations of Apriori have been proposed, e.g., by reducing the number of database scans, partitioning the data, or by sampling [Goethals, 2003].

Eclat

The Eclat algorithm (see Algorithm 2.2) was introduced by Zaki et al. [1997]. Eclat uses the concept of *tid lists*. For an itemset X, this is simply its cover, i.e., the *tid*s of the transactions in $\mathcal{D}$ that contain X. Given two itemsets X_1 and X_2, it is easy to see that the *tid* list of $X_1 \cup X_2$ equals $cover(X_1) \cap cover(X_2)$. A dataset $\mathcal{D}$ is represented in vertical form; rather than consisting of transactions containing items, the data consists of *tid* lists, where each item has a list associated with it, containing the *tid*s of the transactions it occurs in. Further, the algorithm works with conditional datasets, i.e., $\mathcal{D}^I$ is the restriction of $\mathcal{D}$ to all transactions containing I. The Eclat algorithm performs a depth-first traversal of the search space. As such, it does not fully utilize the Apriori property, i.e., it does not check whether *all* subsets of some candidate itemset are frequent. This has the disadvantage that the algorithm might consider more candidate itemsets than the Apriori algorithm. However, this also means that it spends less time checking whether subsets are frequent, and moreover, due to its depth-first approach, the algorithm requires far less memory. In practice, therefore, the algorithm tends to perform very well.

Heuristic improvements to Eclat include the use of diffsets [Zaki and Gouda, 2003], and the ordering of the items based on their support.

Reducing the Output

Due to the fact that the complete set of frequent itemsets can be unwieldy (often in the order of millions or billions), many approaches have proposed to reduce this set. To this end, several notions of *redundancy* have been introduced, in order to condense or summarize the full set of patterns. The idea is to report only a subset of the patterns, such that the omitted ones can be inferred. This can either be done losslessly or lossily. These techniques can potentially reduce the set of patterns by several orders of magnitude.

Algorithm 2.2: Eclat

input : a binary database $\mathcal{D}$; a minimum support threshold *minsup*;
a set of items $I \subseteq \mathcal{I}$ (initially $I = \emptyset$)
output: the set of frequent itemsets $\mathcal{F}[I]$ with prefix I in $\mathcal{D}$ w.r.t. *minsup*

```
1  F[I] ← ∅
2  for each item i ∈ I occurring in D do
3  |  F[I] ← F[I] ∪ {I ∪ {i}}
4  |  D^i ← ∅
5  |  for each item j ∈ I occurring in D such that j > i do
6  |  |  C ← cover(i, D) ∩ cover(j, D)
7  |  |  if |C| ≥ minsup then
8  |  |  |  D^i ← D^i ∪ {⟨j, C⟩}
9  |  |  end
10 |  end
11 |  F[I] ← F[I] ∪ Eclat(D^I, minsup, I ∪ {i})
12 end
13 return F[I]
```

An itemset X is called *closed* [Pasquier et al., 1999] if its support is strictly higher than the support of all of its supersets, that is,

$$supp(X) > supp(Y) \text{ for all } Y \supset X \,.$$

A collection of closed frequent itemsets allows us to reconstruct the complete set of frequent itemsets; if a frequent itemset X does not occur in it (i.e., is not closed), then we can simply look up its smallest proper superset, whose support is equal to the support of X. If there is no such superset, then we know that X was infrequent to begin with.

Related to closedness, an itemset is called a *generator* or *free* [Boulicaut et al., 2003], if its support is strictly lower than the support of all of its subsets,

$$supp(X) < supp(Y) \text{ for all } Y \subset X \,.$$

Another example are non-derivable itemsets [Calders and Goethals, 2007]. An itemset is called *derivable* if its support can be inferred exactly from the supports of all its proper subsets. That is, using the Inclusion-Exclusion principle we can calculate tight lower and upper bounds on $supp(X)$ that are

based on $supp(X')$ for all $X' \subsetneq X$. If these bounds coincide, the support of X is known exactly, and hence it can be seen as redundant. For example, say we have an itemset $X = abc$. Then using Inclusion-Exclusion, we know that, e.g., $supp(abc) \geq supp(ab) + supp(ac) - supp(a)$. For an itemset of size 3, there are 2^3 such upper and lower bounds. The set of non-derivable frequent itemsets is downward closed; if a frequent itemset is non-derivable, then so are all of its subsets. From the set of all frequent non-derivable itemsets, we can construct the full set of frequent itemsets; if X is frequent and derivable, we can infer its support (possibly recursively) from its subsets.

Other examples of approaches to reduce a set of patterns include mining maximal frequent itemsets [Bayardo, 1998, Gouda and Zaki, 2001], and non-redundant association rules [Zaki, 2000].

Mining Sets of Patterns

The techniques mentioned in the previous section aim to reduce the output of a pattern mining algorithm somehow, but often still result in relatively large pattern sets. In recent years, methods have emerged that try to discover even smaller sets of patterns. The key realization is that in order to end up with a small, non-redundant pattern set, we must consider how the patterns relate to each other, rather than considering them individually. The general approach is to greedily add itemsets, either by considering them in some order and only adding an itemset if some criterion is satisfied, or maintaining a model of the data against which itemsets are evaluated, and updating it for each new itemset that is added. See, for instance, [Knobbe and Ho, 2006b, Gallo et al., 2007, Bringmann and Zimmermann, 2009, Kontonasios and De Bie, 2010, Vreeken et al., 2011].

2.2 Information Theory

To assess the interestingness of a pattern or set of patterns, this dissertation heavily relies on concepts from information theory. This allows us to quantify informativeness in a formal and well-founded way.

Central to the field of information theory is the notion of *entropy*, which was introduced by Shannon [1948]. Entropy measures the information content of a given random variable. Say we have a message over some alphabet that we want to transmit from a sender A to a receiver B, and we wish to do

so efficiently, i.e., in as few bits as possible (assuming without loss of generality that we are sending it in binary). To this end, A and B must agree on some code to transmit the message in. For instance, each symbol could be represented by a bitstring of a certain fixed length. However, if A and B have knowledge of the distribution over the symbols, they can use a code such that the total expected length of the message is minimized. Shannon showed that such a code is *optimal* if the length of a symbol's code is equal to $-\log p(x)$, where $p(x)$ is the probability of symbol x. The expected length of the message per symbol is then equal to $-\sum_x p(x) \log p(x)$. This is called the entropy of the source. It describes how much information is contained in X. Alternatively, entropy can be seen as a measure of the complexity or uncertainty of the source; the more uncertain we are, the more bits we need on average to transmit a message.

Formally, given a discrete random variable X with domain $\mathcal{X}$ and a probability distribution p, the entropy of X is defined as

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x) \ .$$

The base of the logarithm above is 2, and by convention $0 \log 0 = 0$.

For a dataset $\mathcal{D}$ and an attribute set X, we can define the entropy of X as

$$H(X, \mathcal{D}) = -\sum_{x \in \mathcal{X}} fr(X = x) \log fr(X = x) \ ,$$

where $\mathcal{X} = dom(X)$.

Very often we may want to compare the information content of two random variables, or express how close one is to the other. The *Kullback-Leibler divergence* [Cover and Thomas, 2006] between two random variables X and Y, with probability distributions p and q over a domain $\mathcal{X}$, is defined as

$$KL\,(X \| Y) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \ .$$

Informally, it tells us how different q is from p. The Kullback-Leibler divergence expresses how many additional bits are needed on average per symbol if we send a message that is distributed according to p, using a code that is optimal for q instead. It is nonnegative, equal to zero if and only if $p = q$, and asymmetric (hence it is not a distance metric).

Based on Kullback-Leibler, the *mutual information* between two random variables X and Y with respective domains $\mathcal{X}$ and $\mathcal{Y}$, is defined as the divergence of their joint distribution r from the product distribution $p \cdot q$,

$$\begin{aligned} I(X,Y) &= KL\left(r \| p\, q\right) \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} r(x,y) \log \frac{r(x,y)}{p(x)q(y)} \end{aligned}$$

where p and q are the marginal distributions of r with respect to $\mathcal{X}$ and $\mathcal{Y}$ respectively, i.e., $p = r_{\mathcal{X}}$ and $q = r_{\mathcal{Y}}$. If X and Y are independent then their mutual information is equal to zero and vice versa. Mutual information can conveniently be expressed in terms of entropy, as

$$\begin{aligned} I(X,Y) &= H(Y) - H(Y \mid X) \\ &= H(X) - H(X \mid Y) \\ &= H(X) + H(Y) - H(X,Y) \end{aligned}$$

where $H(Y \mid X)$ is the conditional entropy of Y given X, and $H(X,Y)$ is the joint entropy of X and Y.

Several authors have introduced methods to asses the quality of patterns based on information-theoretic concepts, see, e.g., [Jaroszewicz and Simovici, 2004, Heikinheimo et al., 2007].

2.3 Model Selection

The problem of finding a suitable model for a given dataset is far from trivial. Given a family of possible models, the task is to pick the model that somehow is best at capturing the data. Good model selection techniques generally not only take into account how well a certain model fits the data, but also consider the complexity of the model itself. Such techniques are guided by Occam's razor, which may be phrased as follows.

> Given two hypotheses of equal explanatory power, the simplest one is to be preferred.

Here, we briefly describe two model selection techniques used in this thesis.

BIC

Given a dataset $\mathcal{D}$ and a probabilistic model p, the Bayesian Information Criterion (BIC) [Schwarz, 1978] is defined as

$$-\log p(\mathcal{D}) + \frac{k}{2}\log|\mathcal{D}|$$

where k is the number of free parameters of p. The first term is the negative log-likelihood of the model, so the better the model fits the data, the lower this term gets. The second term is a penalty term, which is a function of the complexity of p. Therefore, the simpler the model, the lower this penalty.

Note that the first term scales linearly with the size of $\mathcal{D}$, whereas the second term grows logarithmically. Therefore, the more data (or evidence) that is available, the more the first term will outweigh the complexity penalty, i.e., the more complex our model is allowed to be.

MDL

The Minimum Description Length (MDL) principle [Rissanen, 1978, Grünwald, 2005], like its close cousin MML (Minimum Message Length) [Wallace, 2005], is a practical version of Kolmogorov Complexity [Li and Vitányi, 1993]. All three embrace the slogan *Induction by Compression*. The MDL principle can roughly be described as follows.

Given a dataset $\mathcal{D}$ and a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ is the one that minimizes

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

in which

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

This is called two-part MDL, or *crude* MDL. This stands opposed to *refined* MDL, where model and data are encoded together [Grünwald, 2007]. In this dissertation we use two-part MDL, because we are specifically interested in the model. Moreover, although refined MDL has stronger theoretical foundations, it cannot be computed except for some special cases.

The first term is the description length of the model. What this description looks like, depends on the *encoding* that is used. Each model gets a code such that the encoded message is uniquely decodable. In general, the length of this code should reflect the complexity of the model. In order to be decodable, the encoding should also satisfy Kraft's inequality, i.e.,

$$\sum_{M \in \mathcal{M}} 2^{-L(M)} \leq 1 \,.$$

Besides as a penalty on its complexity, the description length of the model can also be seen as an assertion of the prior expectation of that model. Namely, we can define (up to normalization) a probability distribution Pr, where $Pr(M) = 2^{-L(M)}$ is the prior probability of model M. In practice, however, it is often more natural to specify a description than a probability distribution. The second term describes the data itself, encoded with the model M. The better the model fits the data, the shorter this description should be. Generally this corresponds again to the negative log-likelihood of the data, that is, $L(\mathcal{D} \mid M) = -\log p(\mathcal{D} \mid M)$. Combining these two terms, we get a description length that aims to balance the fit of the model and its complexity.

From the point of view of communicating a message from a sender A to a receiver B, in contrast to Section 2.2, both sender and receiver now have no knowledge of the distribution of the data. Hence, they must first send a model of the data, such that they can subsequently send the data itself, using that model. (Note, however, that A and B still need to agree upon a code to describe models.) According to the MDL principle, the model that minimizes the total number of bits that A needs to send to B, is the best model.

Some approaches that use model selection techniques such as BIC or MDL in a pattern mining context include [Tatti and Heikinheimo, 2008, Tatti and Vreeken, 2008, Kontonasios and De Bie, 2010, Vreeken et al., 2011].

Chapter 3

Mining Non-redundant Information-Theoretic Dependencies between Attribute Sets

In this chapter we present an information-theoretic approach for mining strong dependencies between attribute sets in binary data. We define measures based on entropy and mutual information to asses the quality of such rules. The problem of local redundancy is theoretically investigated in this context, and we present lossless pruning techniques based on set closures, as well as lossy techniques based on rule augmentation. We introduce an efficient and scalable algorithm called μ-Miner, using Quick Inclusion-Exclusion to achieve fast entropy computation. The algorithm is empirically evaluated through experiments on synthetic and real-world data.

This chapter is based on work published as:
M. Mampaey. Mining non-redundant information-theoretic dependencies between itemsets. In *Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery (DaWaK), Bilbao, Spain*, volume 6263 of *LNCS*, pages 130–141. Springer, 2010.

3.1 Introduction

The discovery of rules from data is an important problem in data mining. Mining association rules in transactional datasets particularly has received a lot of attention [Agrawal et al., 1993, Zaki et al., 1997, Han et al., 2000]. The objective of association rule mining is to find highly confident rules between sets of items frequently occurring together. This has been generalized to, among others, relational tables with categorical or numerical attributes [Srikant and Agrawal, 1996]. In this context, much attention has gone to the discovery of (approximate) functional dependencies in relations [Kivinen and Mannila, 1995, Huhtala et al., 1999a]. A functional dependency $X \Rightarrow Y$ between two sets of attributes is said to hold, if any two tuples agreeing on the attributes of X also agree on the attributes of Y. Often it is desirable to also find rules that *almost* hold, for instance if the data is noisy. Typically, an error is associated with a functional dependency, which describes how well the relation satisfies the dependency, commonly this is the minimum relative number of tuples that need to be removed from the relation for the dependency to hold [Kivinen and Mannila, 1995]. These tuples can be thought of as being the exceptions to the rule. However, the fact that there are few tuples violating a dependency $X \Rightarrow Y$, does not necessarily also mean that Y strongly depends on X; in fact, X and Y might even be independent.

Therefore, we take an information-theoretic approach to mining dependencies between sets of attributes in binary data. We express the dependence of a rule based on the mutual information between the consequent and antecedent. Furthermore, we use the entropy of a rule or attribute set to describe its complexity. We present an algorithm called μ-Miner that mines rules with a high dependence and a low complexity, and does so efficiently by making use of the Inclusion-Exclusion principle.

Further, we investigate what kinds of redundancy can occur in the collection of all low entropy, high dependence rules. For association rules, several types of redundancy have been studied [Zaki, 2000, Balcázar, 2008]. In this chapter we examine two types of redundancy; one that is lossless and based on set closures, and one that is lossy and based on superfluous augmentation. These techniques are then integrated into the algorithm.

The structure of this chapter is as follows. In Section 3.2 we discuss related work. In Section 3.3 we define the measures we will use to assess dependence rules, and investigate some of their theoretical properties. In Section 3.4,

the issue of redundancy is treated. In Section 3.6 we present the μ-MINER algorithm, which is experimentally evaluated in Section 3.7. The chapter ends with conclusions in Section 3.8.

3.2 Related Work

The discovery of exact and approximate functional dependencies from relations has received a lot of attention in the literature. The TANE algorithm proposed by Huhtala et al. [1999a] finds exact and approximate functional dependencies in a relation, which have a low g_3 error, defined as follows,

$$g_3(X \Rightarrow Y) = \min\{|\mathcal{D}'| \mid \mathcal{D}' \subseteq \mathcal{D} \text{ and } \mathcal{D} \setminus \mathcal{D}' \text{ satisfies } X \Rightarrow Y\}/|\mathcal{D}| \,.$$

TANE is a breadth-first algorithm that works with transaction partitions induced by attribute sets, which can be constructed in linear time with respect to the size of the dataset. For two attribute sets X and Y, if the partition induced by XY does not refine the partition induced by X, then $X \Rightarrow Y$ is a functional dependency. The error of an approximate dependency can also be computed using these partitions. The main difference with our work is the way that the strength of a dependency is measured, but also that TANE only mines minimal rules, i.e., rules of the form $X \Rightarrow Y$ for which $|Y| = 1$ and there is no $X' \subsetneq X$ such that $X' \Rightarrow Y$ is an (approximate) functional dependency. Further, we also consider the complexity of dependencies.

Dalkilic and Robertson [2000] propose to use conditional entropy to determine the strength of dependencies in relational data. Their does not focus on rule discovery, but examines several properties and information inequalities from a theoretical point of view.

Jaroszewicz and Simovici [2002] use information-theoretic measures to assess the importance of itemsets or association rules on top of the traditional support/confidence-based mining framework. They use Kullback-Leibler divergence (of which mutual information is a special case), to determine the redundancy of confident association rules. Given the supports of some subsets of an association rule, its most likely confidence is computed, using a maximum entropy model. If the actual confidence of the rule in the data is close to the estimate, the rule is considered to be redundant.

Heikinheimo et al. [2007] mine all low entropy sets from binary data, as well as tree structures based on these sets. A breadth-first mining algorithm

is proposed that exploits the monotonicity of entropy, after which additional structure is imposed on these sets, in the form of Bayesian trees. The nodes of these trees correspond to attributes, and their directed edges express the conditional entropy between the attributes. The paper also discusses the use of high entropy sets, which are argued to be potentially interesting due to the lack of correlation among their attributes.

3.3 Strong Dependence Rules

In this section we introduce our interestingness measures for attribute sets and rules, and explore some of their theoretical properties.

Definitions

Definition 3.1 (Rule Entropy)**.** *Let X and Y be two disjoint attribute sets. The entropy h of the rule $X \Rightarrow Y$ is defined as the joint entropy of X and Y,*

$$h(X \Rightarrow Y) = H(X \cup Y) \ .$$

It is easy to see that $0 \leq h(X \Rightarrow Y) \leq \log |dom(XY)| = |X| + |Y|$.

Definition 3.2 (Rule Dependence)**.** *Let X and Y be two disjoint attribute sets. We define the dependence μ of the rule $X \Rightarrow Y$ as*

$$\mu(X \Rightarrow Y) = \frac{I(X,Y)}{H(Y)} \ .$$

By dividing the mutual information by $H(Y)$ we obtain a normalized, asymmetric measure, which ranges from 0 to 1. If X and Y are independent then $\mu(X \Rightarrow Y) = 0$, which means that X tells us nothing about Y. On the other hand, it holds that $\mu(X \Rightarrow Y) = 1$ if and only if X fully determines Y; in this case the rule is called *exact*.

Properties

Here we describe some useful properties of h and μ, which we exploit later to construct our algorithm.

Theorem 3.1 (Monotonicity of Entropy)**.** *Let X and X' be two attribute sets. If $X \subseteq X'$, then it holds that $h(X) \leq h(X')$.*

Proof. Let us write $X' = X \cup \{x\}$. Using the chain rule of entropy, we have

$$H(X') = H(X) + H(x \mid X) \ .$$

Since it holds that $H(x \mid X) \geq 0$, we can conclude that $H(X') \geq H(X)$. □

Using the monotonicity of h, it is possible to efficiently traverse the search space of all attribute sets to discover the ones with low entropy.

Theorem 3.2 (Antecedent Monotonicity)**.** *Let X, X' and Y be attribute sets with $X \subseteq X'$, then it holds that $\mu(X \Rightarrow Y) \leq \mu(X' \Rightarrow Y)$.*

Proof. Expanding the denominator in the definition of μ, we see that

$$\begin{aligned} I(X, Y) &= H(Y) + H(X \mid Y) \\ &\leq H(Y) + H(X' \mid Y) \\ &= I(X', Y) \ . \end{aligned}$$

The inequality above follows directly from Theorem 3.1. □

Theorem 3.2 implies that the more attributes we put in the antecedent of a rule, the higher its dependence μ gets. However, this will also increase the entropy of such the rule, and hence it will be pruned. Furthermore, some of the attributes might be independent of the other ones, and it is quite likely that such large rules display some sort of redundancy as described in the next section.

Theorem 3.3 (Partial Monotonicity)**.** *Let $X \Rightarrow Y$ be a rule, where X and Y are disjoint. There exists an attribute $y \in Y$ such that $\mu(X \Rightarrow Y) \leq \mu(Xy \Rightarrow Y \setminus y)$.*

Proof. Choose an attribute $y' \in Y$ that maximizes the term $\mu(Xy' \Rightarrow Y \setminus y')$. Then we have

$$\frac{I(Xy', Y \setminus y')}{H(Y \setminus y')} \geq \frac{\sum_{y \in Y} I(Xy, Y \setminus y)}{\sum_{y \in Y} H(Y \setminus y)} \ .$$

We prove that the second term is greater than or equal to $\mu(X \Rightarrow Y)$, i.e,

$$\frac{\sum_{y \in Y} I(Xy, Y \setminus y)}{\sum_{y \in Y} H(Y \setminus y)} \geq \frac{I(X, Y)}{H(Y)} \ ,$$

which is true if and only if

$$\frac{\sum_{y \in Y} H(Y \setminus y \mid Xy)}{\sum_{y \in Y} H(Y \setminus y)} \leq \frac{H(Y \mid X)}{H(Y)} .$$

This last inequality follows from combining the following two inequalities.

$$\frac{1}{k} \sum_{y \in Y} \frac{H(Y \setminus y)}{|Y \setminus y|} \geq \frac{H(Y)}{|Y|}$$
$$\frac{1}{k} \sum_{y \in Y} \frac{H(Y \setminus y \mid Xy)}{|Y \setminus y|} \leq \frac{H(Y \mid X)}{|Y|}$$

Here $k = |Y|$. For a proof of these inequalities, see Cover and Thomas [2006], Theorems 17.6.1 and 17.6.3 respectively. □

This last theorem allows us to systematically and efficiently construct all rules $X \Rightarrow Y$ with a high dependence based on a given low entropy set $Z = XY$, in a levelwise fashion. Hence, it is not necessary to consider all $2^{|Z|}$ possible rules that can possibly be constructed from that attribute set.

Closedness

Due to the explosion of patterns commonly encountered in classic frequent itemset mining, one often turns to mining a condensed representation of a collection of frequent itemsets. Such pattern collections are typically much smaller in magnitude, can be discovered faster, and it is possible to infer other patterns from them. One example are maximal frequent itemsets [Bayardo, 1998, Gouda and Zaki, 2001]. Two other popular condensed representations are closed and non-derivable frequent itemsets (see also Section 2.1), which we extend to our approach.

The concept of closedness is well-studied for frequent itemset mining [Pasquier et al., 1999]. An itemset is closed with respect to support if all of its proper supersets have a strictly smaller support. A closure operator can be defined that maps an itemset to its (unique) smallest closed superset, i.e., its closure. Similarly, we can consider attribute sets that are closed with respect to entropy. We formally do this as follows. The set inclusion relation ($\subseteq$) defines a partial order on the powerset $\mathcal{P}(\mathcal{A})$ of all attribute sets. Further, a partial order, i.e., refinement ($\sqsubseteq$), can be defined on the set $\mathcal{Q}(\mathcal{T})$

of all transaction partitions. A given attribute set $X \in \mathcal{P}(\mathcal{A})$ partitions $\mathcal{T}$ into equivalence classes according to the value that X obtains in the transactions, and conversely a partition in $\mathcal{Q}(\mathcal{T})$ corresponds to an attribute set in $\mathcal{P}(\mathcal{A})$. Note that the entropy of an attribute set is computed using the sizes of the equivalence classes in its corresponding partition. Let us call these two mapping functions i_1 and i_2. It can be shown that i_1 and i_2 form a Galois connection [Davey and Priestley, 1990] between $(\mathcal{P}(\mathcal{A}), \subseteq)$ and $(\mathcal{Q}(\mathcal{T}), \sqsubseteq)$. The composition $cl := i_2 \circ i_1$ now defines a closure operator on $\mathcal{P}(\mathcal{A})$, which satisfies the following properties for all attribute sets $X \subseteq \mathcal{A}$.

$$\begin{cases} X \subseteq cl(X) & \text{(extension)} \\ cl(X) = cl(cl(X)) & \text{(idempotency)} \\ X \subseteq X' \Rightarrow cl(X) \subseteq cl(X') & \text{(monotonicity)} \end{cases}$$

Definition 3.3. *We call an attribute set $X \subseteq \mathcal{I}$ closed if $X = cl(X)$. The set X is called a generator if for all $X' \subsetneq X$ it holds that $cl(X') \subsetneq X$.*

It holds that all proper supersets of a closed attribute set have a strictly higher entropy and $h(X) = h(cl(X))$. All proper subsets of a generator have strictly lower entropy. Note that a rule $X \Rightarrow Y$ is exact, i.e., $\mu(X \Rightarrow Y) = 1$, if and only if $X \subseteq XY \subseteq cl(X)$. Furthermore, if an exact rule $X \Rightarrow Y$ is minimal, i.e., there is no $X' \subset X$ such that $X' \Rightarrow Y$ is exact, then X is a generator.

Derivability

The notion of itemset derivability was introduced by Calders and Goethals [2007]. Given the supports of all proper subsets of an itemset $(X = 1)$, it is possible, using the inclusion-exclusion principle, to derive tight lower and upper bounds on its own support. If these bounds coincide we know $supp(X = 1)$ exactly, and $(X = 1)$ is called derivable. The set of all frequent itemsets can thus be derived from the set of all non-derivable frequent itemsets. Similarly, we can define the derivability of an attribute set.

Definition 3.4. *We call X h-derivable if its entropy can be determined exactly from the entropies of its proper subsets.*

Derivability (both for itemsets and attribute sets) is a monotonic property. Interestingly, an attribute set X is h-derivable if and only if the corresponding itemset $(X = 1)$ is derivable with respect to its support.

3.4 Rule Redundancy

Mining all low entropy, high dependence rules can yield a very large set of patterns, which is not desirable for a user who wants to analyze them. Typically, this collection contains a lot of redundant rules. In this section we investigate how we can characterize, and consequently prune such rules. We define two types of redundancy; one that is lossless, and based on the closure of attribute sets, and one that is lossy, and based on the unnecessary augmentation of the antecedent or consequent of a rule.

Closure-based Redundancy

As mentioned in the previous section, rules of the form $X \Rightarrow cl(X)$ are always exact. It should be clear that combining an arbitrary rule with an appropriate exact rule yields a new rule with identical entropy and dependence. For instance, if $A \Rightarrow B$ is exact, then $\mu(A \Rightarrow C) = \mu(AB \Rightarrow C)$.

Theorem 3.4. *Let $X \Rightarrow Y$ and $X' \Rightarrow Y'$ be two rules, where $X \subseteq X' \subseteq cl(X)$ and $Y \subseteq Y' \subseteq cl(Y)$. Then $h(X \Rightarrow Y) = h(X' \Rightarrow Y')$ and $\mu(X \Rightarrow Y) = \mu(X' \Rightarrow Y')$.*

Proof. Since $cl(X) = cl(X')$, it follows that $H(X) = H(X')$, and similarly $H(Y) = H(Y')$ and $H(XY) = H(X'Y')$. Hence the theorem holds. □

Since the entropy and dependence of such larger rules can be inferred using the smaller rule and the closure operator, we call them redundant. These minimal rules are constructed using generators.

Definition 3.5 (Closure-based Redundancy)**.** *A rule $X \Rightarrow Y$ is redundant with respect closure if*

$$\begin{cases} X \text{ is not a generator or } |Y| > 1 & \text{if } \mu(X \Rightarrow Y) = 1\,, \\ XY \text{ is not generator} & \text{if } \mu(X \Rightarrow Y) < 1\,. \end{cases}$$

Note that if XY is a generator, then X and Y are also generators, since the set of all generators is downward closed.

Augmentation-based Redundancy

Here we define a stricter kind of redundancy that prunes rules which have superfluous attributes added to their antecedents or consequents.

Antecedent Redundancy

Suppose we have two rules with a common consequent, $X \Rightarrow Y$ and $X' \Rightarrow Y$, where $X' = X \cup \{x\}$. Theorem 3.2 tells us that $\mu(X \Rightarrow Y) \leq \mu(X' \Rightarrow Y)$. Even though the latter rule has a higher dependence, it might be redundant if x does not make a real contribution to the rule. For instance, if X and x are independent, then $\mu(X' \Rightarrow Y)$ is simply the sum of $\mu(X \Rightarrow Y)$ and $\mu(x \Rightarrow Y)$. To characterize this type of redundancy we use the chain rule of mutual information,

$$I(X', Y) = I(X, Y) + I(x, Y \mid X) \,,$$

where the last term is the conditional mutual information (which can be written as $I(x, Y \mid X) = H(x \mid X) - H(x \mid XY)$). It is known that I does not behave monotonically with conditioning. In the aforementioned case where X and x are independent, we have $I(x, Y \mid X) = I(x, Y)$. If X already explains part of the dependency between x and Y, then $I(x, Y \mid X) < I(x, Y)$, meaning there is an "overlap" between X and x. Otherwise, if $I(x, Y \mid X) > I(x, Y)$, this means that under knowing X, the mutual information between x and Y increases. Intuitively, it means that X' tells us more about Y than the sum of X and x separately—the rule $X' \Rightarrow Y$ is more informative than the sum of its parts. This motivates the following definition.

Definition 3.6 (Antecedent Redundancy)**.** *A rule $X \Rightarrow Y$ is redundant with respect to antecedent augmentation, if there exists an attribute $x \in X$ such that*

$$\begin{cases} \mu(X \Rightarrow Y) \leq \mu(X \setminus x \Rightarrow Y) + \mu(x \Rightarrow Y), \text{ or} \\ X \setminus x \Rightarrow Y \text{ is redundant} \,. \end{cases}$$

From this definition it follows that $\mu(X \Rightarrow Y) > \sum_{x \in X} \mu(x \Rightarrow Y)$ whenever the rule $X \Rightarrow Y$ is non-redundant.

Consequent Redundancy

Consider the rule $X \Rightarrow Y$ and an attribute $y \notin XY$. Unlike in the previous section, μ is not monotonic with respect to augmentation of the consequent, so in general the dependence of $X \Rightarrow Yy$ can either be higher or lower than that of $X \Rightarrow Y$. An increase in μ would mean that adding y to Y increases the mutual information $I(X, Y)$ more than it increases the entropy $H(Y)$.

Put differently, the relative increase in uncertainty from $H(Y)$ to $H(Yy)$ is surpassed by the increase in the amount of information X gives about Y and Yy, i.e., X gives relatively more information about Yy than it does about Y.

Definition 3.7 (Consequent Redundancy)**.** *A rule $X \Rightarrow Y$ is redundant with respect to consequent augmentation, if there exists an attribute $y \in Y$ such that*

$$\begin{cases} \mu(X \Rightarrow Y) \leq \mu(X \Rightarrow Y \setminus y), \text{ or} \\ X \Rightarrow Y \setminus y \text{ is redundant .} \end{cases}$$

From this definition it follows that if the rule $X \Rightarrow Y$ is non-redundant, then it holds that $\forall Y' \subset Y : \mu(X \Rightarrow Y') < \mu(X \Rightarrow Y)$.

Relation to Closure-based Redundancy

It turns out that augmentation redundancy is strictly stronger than closure-based redundancy, as stated in the theorem below.

Theorem 3.5. *If a rule $X \Rightarrow Y$ is redundant with respect to closure, then it is also redundant with respect to antecedent augmentation or consequent augmentation.*

Proof. First, suppose $\mu(X \Rightarrow Y) = 1$. If X is not a generator, then there exists a generator $X' \subsetneq X$ such that $\mu(X \Rightarrow Y) \leq \mu(X' \Rightarrow Y) + \mu(X \setminus X' \Rightarrow Y)$, and hence the rule is antecedent-redundant. Otherwise, if Y is not a singleton, then $\mu(X \Rightarrow Y) = \mu(X \Rightarrow Y')$ for every non-empty $Y' \subset Y$, and hence the rule is consequent-redundant. Second, suppose that $\mu(X \Rightarrow Y) < 1$. If XY is not a generator, then there exists a generator $X'Y' \subset XY \subseteq cl(X'Y')$ with $X' \subseteq X$ and $Y' \subseteq Y$. If $X' \neq X$ then the rule is antecedent-redundant, if $Y' \neq Y$ then the rule is consequent-redundant. □

3.5 Problem Statement

Given a binary dataset $\mathcal{D}$, and user-defined thresholds $h_{\max}$ and $\mu_{\min}$, find all rules $X \Rightarrow Y$, where $X, Y \subseteq \mathcal{A}$, such that

1. $h(X \Rightarrow Y) \leq h_{\max}$,
2. $\mu(X \Rightarrow Y) \geq \mu_{\min}$,
3. $X \Rightarrow Y$ is non-redundant .

3.6 The μ-Miner Algorithm

In this section we present the μ-Miner algorithm (see Algorithm 3.1). As input it takes a dataset $\mathcal{D}$, a maximum entropy threshold $h_{\max}$, and a minimum dependence threshold $\mu_{\min}$. The algorithm efficiently mines low entropy attribute sets, and from these sets strong dependence rules are constructed. Further, μ-Miner prunes rules that are closure redundant or augmentation redundant. The calculation of entropy and dependence, and the checking of redundancy is done by performing some simple arithmetic operations and lookups, and only one scan of the database is required.

Algorithm 3.1: μ-Miner

input : a binary dataset $\mathcal{D}$; a maximum entropy threshold $h_{\max}$;
a minimum dependency threshold $\mu_{\min}$
output: the non-redundant sets and rules in $\mathcal{D}$ w.r.t. $h_{\max}$ and $\mu_{\min}$

1 $\mathcal{L} \leftarrow \{\{x\} \mid x \in \mathcal{A} \text{ and } h(x) \leq h_{\max}\}$
2 AttributeSetMine($\mathcal{L}$, $h_{\max}$, $\mu_{\min}$)

Mining Attribute Sets

In the AttributeSetMine function (Algorithm 3.2), attribute sets with a low entropy are mined. This recursive function takes a collection of attribute sets with a common prefix as input, initially this is the set of all low entropy singletons. The search space is traversed in a depth-first, right-most fashion. This is far less memory-intensive than a breadth-first approach, and the right-most order ensures that when an attribute set is considered, all of its subsets will already have been visited in the past (lines 1&3), a fact we exploit later. This implies that we need to impose an order on the attributes. In our implementation of μ-Miner we use a heuristic ordering based on the entropy of the attributes, such that large subtrees of the search space are rooted by sets which are expected to be have a high entropy, which allows us to potentially prune large parts of the subspace.

Algorithm 3.2: AttributeSetMine

```
input : an attribute set collection 𝓛; a maximum entropy threshold
        h_max; a minimum dependency threshold μ_min
output: the low entropy sets w.r.t. h_max
for X_1 in 𝓛 in descending order do
    𝓛' ← ∅
    for X_2 < X_1 in 𝓛 do
        X ← X_1 ∪ X_2
        compute and store fr(X = 1)
        h(X) ← EntropyQie(X)
        if X is not a generator then
            report the corresponding exact rule(s)
        else if h(X) ≤ h_max then
            𝓛' ← 𝓛' ∪ {X}
            DependenceRuleMine(X, μ_min)
        end
    end
    𝓛' ← 𝓛' ∪ AttributeSetMine(𝓛', h_max, μ_min)
end
return 𝓛'
```

Efficiently Computing Entropy

A straightforward method to compute $h(X)$, is to perform a scan over the database to obtain the frequencies of $(X = v)$ for all values $v \in \{0,1\}^{|X|}$. In total there are 2^k such frequencies for $k = |X|$, however, at most $|\mathcal{D}|$ of them are nonzero and hence this method requires $O(|\mathcal{D}|)$ time. If the database fits in memory this counting method is perfectly feasible, otherwise it becomes too expensive, since database access is required for each candidate itemset. Another option is to use the partitioning technique used by Tane [Huhtala et al., 1999a]. For each itemset a partition of the transactions is explicitly computed in $O(|\mathcal{D}|)$ time, and then the sizes of the sets in the partition can be used to compute $h(X)$.

μ-Miner uses a different entropy computation method that does not require direct database access, and has a lower complexity, which is beneficial especially for large datasets. We start from a simple right-most depth-

Algorithm 3.3: ENTROPYQIE

input : a binary dataset $\mathcal{D}$; an attribute set $X \subseteq \mathcal{A}$
output: $h(X)$, the entropy of X

```
1  for each X' ⊆ X do
2  |  p(X') ← fr(X' = 1, D)
3  end
4  for each x ∈ X do
5  |  for each X' ⊆ X where x ∈ X' do
6  |  |  p(X' \ x) ← p(X' \ x) − p(X')
7  |  end
8  end
9  h(X) ← −Σ_{X'⊆X} p(X') log p(X')
10 return h(X)
```

first itemset support mining algorithm similar to ECLAT [Zaki et al., 1997], and store the supports in a trie (line 5). When we have mined the support of $(X = 1)$, the frequencies of all $(X = v)$ are computed with the stored supports of all subsets, by using Quick Inclusion-Exclusion (Algorithm 3.3), which takes $O(k\,2^{k-1})$ in general [Calders and Goethals, 2005]. However, we can again use the fact that at most $|\mathcal{D}|$ frequencies are nonzero, hence this counting method is $O(\min(k\,2^{k-1}, |\mathcal{D}|))$. The advantage of our method is that it is fast and it does not require database access. The disadvantage is that the supports of all mined itemsets must be stored, which may be a problem if memory is scarce and $h_{\max}$ is set high. Note that if we were to restrict ourselves to mining only non-derivable itemsets, we know that $k \leq \lceil \log |\mathcal{D}| \rceil$ [Calders and Goethals, 2007]. In this case the total number of frequencies we need to store is $O(|\mathcal{A}|^{\log |\mathcal{D}|})$ in the worst case, which is polynomial in $|\mathcal{A}|$ for a fixed database size, and polynomial in $|\mathcal{D}|$ for a fixed number of attributes. At this point we can already prune many attribute sets that will not produce any non-redundant rules, since we can detect whether an attribute set is a generator. If it is not, we can simply output the corresponding functional dependencies, and prune all of the attribute set's supersets (line 7).

Algorithm 3.4: DependenceRuleMine

```
   input : a low entropy set X; a dependence threshold μ_min
   output: the non-redundant strong dependence rules based on X
1  K ← {X \ x ⇒ x | x ∈ X}
2  while K is not empty do
3  |  for each A ⇒ B in K do
4  |  |  compute μ(A ⇒ B)
5  |  |  if μ(A ⇒ B) ≥ μ_min then
6  |  |  |  if A ⇒ B is non-redundant then
7  |  |  |  |  report A ⇒ B
8  |  |  |  end
9  |  |  end
10 |  end
11 |  K ← {A \ a ⇒ Ba | A ⇒ B ∈ K, using Theorem 3.3}
12 end
```

Mining Non-redundant Dependence Rules

For each low entropy set, DependenceRuleMine (Algorithm 3.4) is called to generate high dependence rules. It starts from rules with a singleton consequent, and then moves attributes from the antecedent to the consequent. By using the partial monotonicity property from Theorem 3.3, not all 2^k possible rules need to be considered. Since we have the entropies of all subsets available, we can compute the dependence by performing just a few lookups. If a rule is found to have high dependence, it is checked whether the rule is redundant (line 6). Again, since we have the entropies of all subsets available, these redundancy checks can be performed quite efficiently.

3.7 Experiments

We run experiments to evaluate the efficiency of μ-Miner, the quality of the discovered patterns, and the effect of our pruning techniques. The algorithm is implemented in C++, and is made publicly available.[1] All experiments were executed on a Linux system with a 2.2GHz CPU and 2GB of memory.

[1] http://www.adrem.ua.ac.be/implementations

Table 3.1: Characteristics of the datasets used in the experiments. Shown are the number of attributes $|\mathcal{A}|$, and the number of transactions $|\mathcal{D}|$.

Dataset	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{D}\rvert$
Chess	75	3 196
Mushroom	52	8 124
Pumsb	100	49 046
Synthetic	16	1 000 000
Zoo	17	100

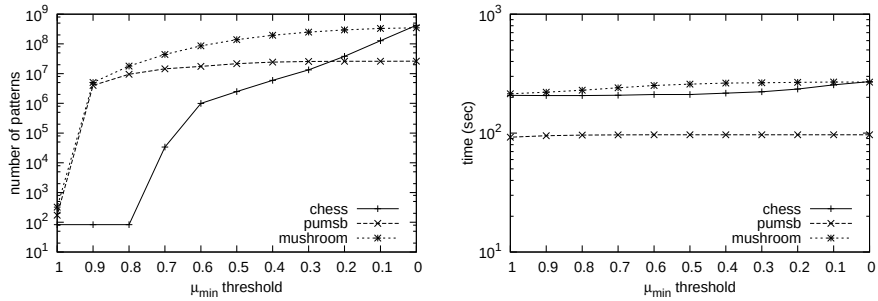
Datasets

We test our algorithm on datasets obtained from the UCI Machine Learning Repository [Frank and Asuncion, 2010] and the FIMI Dataset Repository [Goethals and Zaki, 2003], as well as on a synthetically generated one. The characteristics of these datasets can be found in Table 3.1.

First, we have some benchmark datasets taken from the FIMI repository: *Chess*, *Mushroom* and *Pumsb*. The original *Pumsb* dataset contains 2 112 attributes, in our experiments we used the top-100 high entropy attributes. *Mushroom* originally has 119 attributes, for our experiments we removed items with frequencies higher than 0.9 or lower than 0.1. These datasets are used to test the efficiency of μ-Miner.

Second, we generated a *Synthetic* dataset which contains an embedded pattern. We use it to evaluate the scalability of μ-Miner with respect to the size of the database. The dataset consists of 1 000 000 transactions and has 16 attributes. The 15 first attributes are independent and have random frequencies between 0.1 and 0.9. The last attribute equals the sum of the others modulo 2, i.e., the rule $\{1, \ldots, 15\} \Rightarrow \{16\}$ is a functional dependency. Note that this dependency is also minimal.

Finally, we use the *Zoo* dataset, taken from the UCI Machine Learning Repository, to asses the quality of the patterns that μ-Miner discovers. The dataset contains instances of animals and has 17 attributes that describe some of their properties such as *has-feathers*, *lays-eggs*, *is-predator*, etc. Two attributes are non-boolean: *nr-of-legs* which is integer-valued, and *type* whose value can be one of {*mammal*, *bird*, *reptile*, *fish*, *amphibian*, *insect*, *mollusk*}.



(a) Number of rules

(b) Execution time

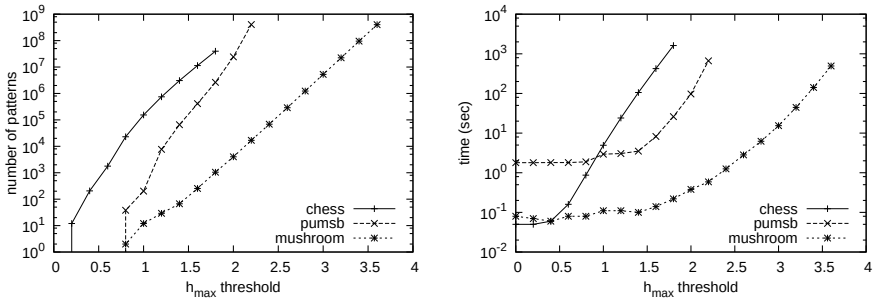
Figure 3.1: The number of returned rules and the execution time of μ-Miner for varying values of the $\mu_{\min}$ threshold parameter.

Experimental Evaluation

First, we perform some experiments on the benchmark datasets. To begin with, we set the value of $h_{\max}$ to a fixed value (1.5 for *Chess*, 2 for *Pumsb*, and 3.5 for *Mushroom*), and we vary the minimum dependence threshold $\mu_{\min}$ between 0 and 1. As can be seen in Figure 3.1a, the number of rules increases roughly exponentially when $\mu_{\min}$ is lowered. Noticeably, the execution times stay roughly constant as $\mu_{\min}$ decreases, as shown in Figure 3.1b. This is not surprising, since most computations are performed in the itemset mining phase, and the computation of μ involves just a few lookups.

Then, we set the value of $\mu_{\min}$ to a fixed value (in this case 0.4 for all datasets), and gradually increase the maximum entropy threshold $h_{\max}$ from zero upward. In Figure 3.2a we see that for very low values of $h_{\max}$ no rules are found. Then, the number of rules increases exponentially with $h_{\max}$, which is to be expected. In Figure 3.2b we see that this trend also translates to the execution times. For lower thresholds ($h_{\max} \leq 1$) the runtimes stay roughly constant, because they are dominated by I/O time.

Secondly, we evaluate the scalability of μ-Miner with respect to the size of the database using the *Synthetic* dataset. The aim is to discover the embedded functional dependency; in order to do this we set $\mu_{\min}$ to 1, and $h_{\max}$ sufficiently high (say, 16). We compare μ-Miner with the Tane and

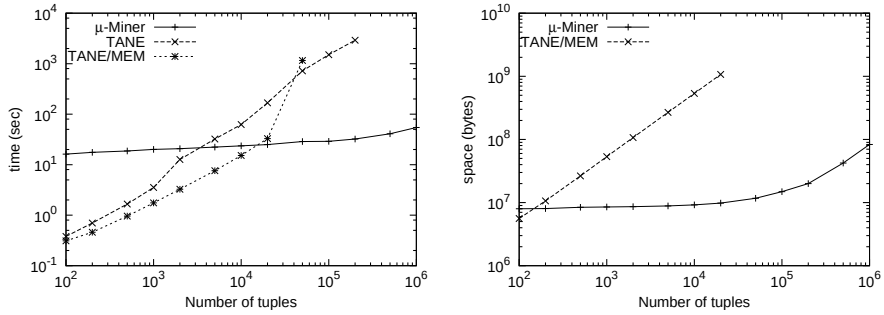


(a) Number of rules

(b) Execution time

Figure 3.2: The number of returned rules and the execution time of μ-Miner for varying values of the $h_{\max}$ threshold parameter.

Tane/mem implementations from [Huhtala et al., 1999b]. The main Tane algorithm stores partitions to disk level per level, while the Tane/mem variant entirely operates in main memory. The number of transactions is gradually increased from 10^2 to 10^6, and the runtimes are reported in Figure 3.3a. In Figure 3.3b the corresponding peak memory consumption of μ-Miner and Tane/mem are reported. The disk space usage of Tane is omitted since it is essentially the same as the memory usage of Tane/mem. We did not run Tane with more than $2 \cdot 10^5$ transactions, since this required more than 10GB of disk space for the largest level. Tane/mem could not be run with more than $5 \cdot 10^4$ transactions due to memory constraints. Meanwhile, even for the maximum number of transactions, μ-Miner requires less than 100MB of memory. Figure 3.3a shows that all algorithms scale linearly with $|\mathcal{D}|$, although the slope is much steeper for Tane and Tane/mem, while the execution time of μ-Miner increases very slowly. At around 3 000 transactions, μ-Miner overtakes Tane in speed, and at 10^5 transactions our algorithm is already two orders of magnitude faster. The Tane/mem algorithm is faster up to $\pm$10 000 transactions, but cannot handle any datasets much larger than that. At 50 000 transactions Tane/mem began swapping heavily. This observed difference in speed can be explained entirely by the counting method. Tane explicitly constructs a partition of size $O(|\mathcal{D}|)$ for each itemset (and stores these to disk or in memory level per level), while our algorithm com-



(a) Scalability (b) Peak memory usage

Figure 3.3: Scalability and peak memory usage of μ-Miner and Tane on the *Synthetic* dataset.

putes the required sizes of the partitions without actually constructing them. For the most part, the increase in the execution time of μ-Miner can be accounted for by the increase in time it takes to read the data file.

Next, let us investigate how pruning affects the size of the output. We experimented on the *Mushroom* and the *Pumsb* datasets for different values of $h_{\max}$ (3 for *Mushroom* and 1.5 for *Pumsb*) and $\mu_{\min}$ (0.2 and 0.8 for both datasets). The results are shown in Figure 3.4. For the *Mushroom* dataset pruning all non-minimal rules already reduces the output by roughly an order of magnitude. Augmentation pruning reduces the output by an additional two orders of magnitude. For the *Pumsb* dataset pruning non-minimal rules reduces the output by three orders of magnitude. Here augmentation pruning reduces the output in size even further, by roughly two orders of magnitude. This makes inspecting the resulting collection of rules far more manageable for a user.

Finally, we asses the quality of the patterns discovered by μ-Miner. The results of the *Zoo* dataset are used to demonstrate the usefulness of our algorithm. We chose some simple examples from the output, these results are easy to interpret since no expertise is required to understand the dataset. We give some examples of high dependence rules. The simplest and strongest rules are *type* $\Rightarrow$ *gives-milk*, *type* $\Rightarrow$ *has-backbone*, and *type* $\Rightarrow$ *has-feathers*. Each of these rules has an entropy of 2.36, and the dependency is 1, i.e., the

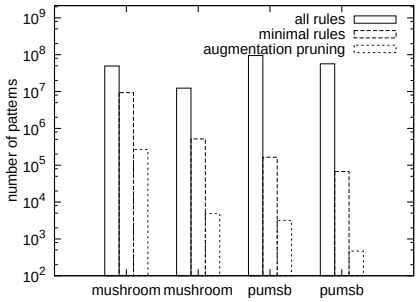


Figure 3.4: The number of returned rules with respect to the different types of redundancy pruning, for the *Mushroom* and *Pumsb* datasets.

type of animal (mammal, bird, insect, etc.) completely determines whether it gives milk, has a backbone, or has feathers. The rule *lays-eggs, is-venomous* $\Rightarrow$ *gives-milk* has an entropy of 1.41, and a dependence of 0.93. Thus, the set $\{$*lays-eggs, is-venomous*$\}$ almost completely determines $\{$*gives-milk*$\}$. The fact that the rule is not exact is entirely due to the transaction for *platypus*, the only animal that lays eggs, is venomous, and does give milk. In this case we can speak of an approximate functional dependency. The subrules *lays-eggs* $\Rightarrow$ *gives-milk* and *is-venomous* $\Rightarrow$ *gives-milk* separately have μ values 0.80 and 0.004. This shows that adding *is-venomous* (which is almost completely independent of *gives-milk*) to *lays-eggs* results in a rule that is more informative than both rules separately. Finally, the rule *has-hair, has-tail* $\Rightarrow$ *lays-eggs, has-backbone* has an entropy of 2.26, and a dependence of 0.70. The rules *has-hair, has-tail* $\Rightarrow$ *lays-eggs* and *has-hair, has-tail* $\Rightarrow$ *has-backbone* separately have lower μ values: 0.64 and 0.63.

3.8 Conclusions

We proposed the use of information-theoretic measures based on entropy and mutual information to mine dependencies between sets of attributes. This allows us to discover rules with a high dependence and a low complexity. We investigated the problem of redundancy in this context, and proposed two techniques to prune redundant rules. One is based on the closure of attribute sets and is lossless, while the other, shown to supersede the first,

is lossy and penalizes the augmentation of rules with superfluous attributes. We presented an algorithm called μ-Miner, which mines such dependencies and applies the pruning techniques above. Several experiments showed that μ-Miner is efficient and scalable: it can easily handle datasets with millions of transactions and does not require large amounts of memory. Furthermore, the proposed pruning techniques were shown to be very effective in reducing the size of the output by several orders of magnitude.

Chapter 4

Summarizing Categorical Data by Clustering Attributes

For a book, its title and abstract provide a good first impression of what to expect from it. For a database, however, obtaining a good first impression is typically not so straightforward. While low-order statistics only provide very limited insight, downright mining the data rapidly provides too much detail for such a quick glance. In this chapter we propose a middle ground, and introduce a parameter-free method for constructing high-quality descriptive summaries of binary and categorical data. Our approach builds a summary by clustering attributes that strongly correlate, and makes use of the Minimum Description Length principle to identify the best clustering—without requiring a distance measure between attributes.

Besides providing a practical overview of which attributes interact most strongly and in which value instantiations they typically occur, these summaries are also probabilistic models that can be used as surrogates for the data, and that can easily be queried. Extensive experimentation shows that our algorithm discovers high-quality results: correlated attributes are correctly grouped together, which is verified both objectively and subjectively.

This chapter is based on work published as:
M. Mampaey and J. Vreeken. Summarising data by clustering items. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Barcelona, Spain*, pages 321–336. Springer-Verlag, 2010.
M. Mampaey and J. Vreeken. Summarizing categorical data by clustering attributes. Manuscript currently submitted to: *Data Mining and Knowledge Discovery*, 2011.

The discovered models can also be employed as surrogates for the data; as an example of this we show that we can quickly and accurately estimate the supports of frequent itemsets.

4.1 Introduction

When handling a book, and wondering about its contents, we can simply start reading it from A to Z. In practice, however, to get a good first impression we usually first consult the summary. For a book, this can be anything from the title, abstract, or blurb, up to simply paging through it. The common denominator here is that a summary quickly provides high-quality and high-level information about the book. Occasionally a summary may already offer us exactly what we were looking for, but in general we just expect to get sufficient insight to determine what the book contains, and whether we need to read it further.

When handling a database, on the other hand, and wondering about its contents and whether (or how) we should analyze it, it is quite hard to get a good first impression. Naturally, we can inspect the schema of the database or look at the labels of the attributes. However, this does not tell use what is *in* the database. Basic statistics only help to a limited extent. For instance, first order statistics can tell us which values of an attribute occur, and how often. While sometimes this may be enough information, typically we would like to see in a bit more detail what the data looks like. However, for binary and categorical databases, further basic statistics are not readily available. Ironically, for these types of data this means that even if the goal is only to get a first impression, in order to acquire this we will have to analyze the data in much more detail. For non-trivially sized databases especially, this means investing more time and effort than we should at this stage of the analysis.

Having a good first impression of the data is very important when analyzing it, as data mining is an essentially iterative process [Hanhijärvi et al., 2009], where each step in the analysis is aimed at obtaining new insight. This insight in turn determines what other results we would find interesting, and hence, determines how to proceed in order to extract further knowledge from the data. As such, a good summary allows us to make an informed decision on what basic assumptions to make and how to start mining the data.

To this end, we here propose a simple and parameter-free method for providing high-quality summary overviews for categorical data, including binary transaction data. These summaries provide insight in which attributes are most correlated, as well as in what value configurations they occur. They are probabilistic models of the data that can be queried fast and accurately, allowing them to be used instead of the data. Further, by showing which

attributes interact most strongly, these summaries can aid in selecting or constructing features. In short, like a proper summary, they provide both a good first impression and can be used as a surrogate.

We obtain these summaries by clustering attributes that interact strongly, such that attributes from different clusters are more or less independent. We employ the Minimum Description Length (MDL) principle to identify the best clustering, thus eliminating the need for the user having to specify any parameters: the best summary is the attribute clustering that describes the data best. Furthermore, our clustering approach does not require a distance measure to be defined between attributes.

As an example of a summary and how it can be insightful, consider Figure 4.1, in which we depict the summary of a hypothetical large categorical database. A summary provides information at two levels of detail. First, it tells us which groups of attributes correlate most strongly. Here, we see that attributes A and J are grouped together, as are attributes B–H, and that I forms a singleton group. For these groups, by the MDL principle, we know that the attributes within them interact strongly, while between the groups the attributes can be considered virtually independent—if an attribute provides information about another one, we could have described the data in fewer bits by considering them together. As such, we learn that I does not interact much with any of the other attributes.

At the second level, summaries allow quick inspection of the attribute-value distributions within the groups. In the example, we see that A and J are each other's inverse, and that attributes B–H have two strongly prevalent value configurations, after which the frequencies drop off rapidly. While a summary contains the frequencies of all attribute-value combinations per group, typically one is most interested in the most prevalent, and hence we here only show the top-5.

We will investigate and analyze summaries obtained from real data in Section 4.6. Extensive experimentation shows that our method provides high-quality results: correlated attributes are correctly grouped together, representative features are identified, and the supports of frequent itemsets are closely approximated in very short time. Randomization further shows that our approach models statistically relevant structure in the data, providing information far beyond simple first order statistics.

To the best of our knowledge, there currently do not exist light-weight data analysis methods that can easily be used for summarization purposes.

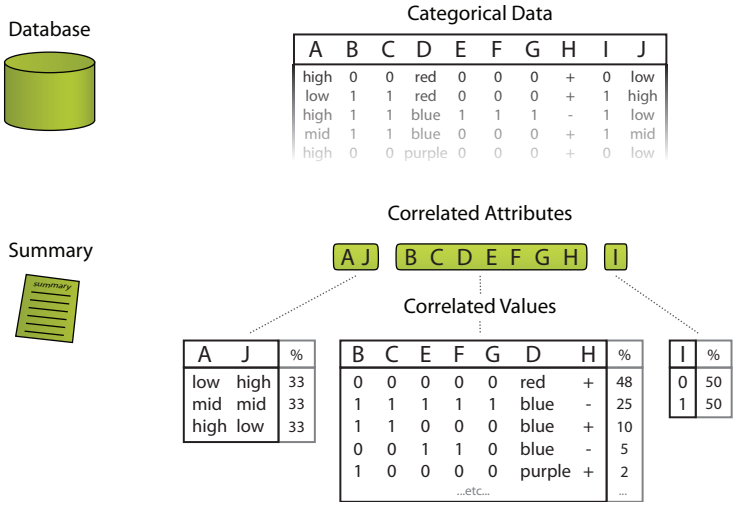


Figure 4.1: Example of a database summary. A summary shows which groups of attributes interact most strongly, as well as which value combinations occur in the database. For presentational clarity, we only show the top-5 most frequent value combinations for *B*–*H* here.

Instead, for binary and categorical data, a standard approach is to first mine for frequent itemsets, the result of which quickly grows to many times the size of the original database. Consequently, many proposals exist that focus on reducing or summarizing these sets of frequent patterns, i.e., they choose groups of representative itemsets such that the information in the complete pattern set is maintained as well as possible. In this work, we do not summarize the outcome of an analysis (i.e., a set of patterns), but instead provide a summary that can be used to decide how to analyze the data further.

Existing proposals for data summarization, such as Krimp [Siebes et al., 2006] and Summarization [Chandola and Kumar, 2005], provide highly detailed results. Although this has obvious merit, analyzing these summaries consequently also requires significant effort. Our method shares the approach of using compression to find a good summary. However, we do not aim to find a group of descriptive itemsets. Instead, we look at the data on the level of attributes, and aim to optimally group those attributes that inter-

act most strongly. In this regard, to a certain extent our approach is related to mining low-entropy sets [Heikinheimo et al., 2007], sets of attributes which identify strong interactions in the data. An existing proposal to summarize data by low-entropy sets, Less [Heikinheimo et al., 2009], requires a collection of low-entropy sets as input, and the resulting models are not probabilistic in nature, nor can they easily be queried. For a more complete discussion of related work, please refer to Section 4.5.

This chapter is structured as follows. Section 4.2 formalizes the problem and discusses how to distinguish good attribute clusterings. In Section 4.3 we present our method to discover good attribute clusterings. Section 4.4 discusses potential alternative search strategies. Related work is treated in Section 4.5. We experimentally evaluate our method in Section 4.6. Lastly, we round up with a discussion in Section 4.7 and conclude in Section 4.8.

4.2 Summarizing Data by Clustering Attributes

In this section we formally introduce our approach. We start with a short recap of MDL (see Section 2.3), and after defining what an attribute clustering is, we show how we can use MDL to identify the best clustering. We then formalize the problem, and discuss the search space. Finally, we also discuss a more refined encoding that takes cues from modern MDL theory.

Minimum Description Length

Given a dataset $\mathcal{D}$ and a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ is the one that minimizes

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

in which

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

To use MDL, we have to define what our set of models $\mathcal{M}$ is, how a model M describes a database, and how all of this is encoded in bits. Below, after giving some further intuition on MDL, we will first discuss a two-part MDL encoding for our summaries, as well as discuss a variant that draws from insights from refined MDL.

The reason we employ the MDL principle here, is that it gives us a well-founded approach for selecting the model that best balances model complexity and fit. That is, models that encode the data concisely, i.e., for which $L(\mathcal{D} \mid M)$ is low, are considered to be good models, because they are able to exploit, and ultimately describe, the structure of the data. In practice, this term often corresponds to the negative log-likelihood of the model.

However, it is possible to come up with an arbitrarily complex model that describes the data arbitrarily well—in the most extreme case one could use a 'model' that just states the data verbatim, which is clearly a form of overfitting. Therefore, a model that has a long description, i.e., for which $L(M)$ is large, is likely to overfit the data, and hence considered to be bad. Models that can be described succinctly, on the other hand, are less prone to overfitting, and additionally also tend to be easier to interpret. Of course, models that are *too* simple probably will not describe the data very well.

The MDL principle provides an appropriate balance between these extremes, by taking both model fit and model complexity into account. MDL is tightly linked to lossless data compression—the best model is the one that best compresses the data. It is also often formulated as a sender/receiver framework, where a sender wants to communicate the data to a receiver, in an as succinct as possible message. The sender encodes both the model and the data, and transmits them to the receiver, who decodes them to obtain the original data. The goal in this chapter is purely to model and summarize the data, not to actually compress or transmit it. However, it is useful to keep this analogy in mind.

Given this basic intuition, let us next formalize how to losslessly encode a model and how to encode the data given that model.

MDL for Attribute Clustering

The main idea for our data summaries are *attribute clusterings*. Therefore, we first formally define the concept of an attribute clustering.

Definition 4.1. *An attribute clustering $\mathcal{C} = \{A_1, \ldots, A_k\}$ of a set of attributes $\mathcal{A}$ is a partition of $\mathcal{A}$, that is,*

1. *every attribute belongs to a cluster:* $\bigcup_{i=1}^{k} A_i = \mathcal{A}$,
2. *all clusters are pairwise disjoint:* $\forall i \neq j : A_i \cap A_j = \emptyset$,
3. *there are no empty clusters:* $\forall A_i \in \mathcal{C} : A_i \neq \emptyset$.

Informally speaking, an attribute clustering is simply a grouping of the attributes $\mathcal{A}$. In order to give the user a bit more information, for each cluster we also include the distribution of the attribute-value occurrences in the data.

Given this definition, we can formalize how we actually encode our models and data, such that we can apply the MDL principle to distinguish good summaries from bad ones. We start by specifying how we encode our models, since then the encoding of the data follows straightforwardly. In our case, a model is a partition of the attributes, $\mathcal{A}$, together with descriptions of which value combinations occur how often, for each per partition, i.e., the distributions within the clusters.

Encoding the Partition Let us first describe how we can encode an attribute partition. Recall that the number of unique partitions for a set of n attributes is given by the well-known Bell number, denoted B_n, which can be computed using a simple recursive formula:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k \ ,$$

with base case $B_0 = 1$. Now, using a basic approach in information theory, we can fix a canonical order on the set of partitions, which allows us to enumerate all possible partitions. More importantly, this allows us to uniquely and efficiently identify a particular partition using $\log B_n$ bits. This is called a model-to-data code [Vereshchagin and Vitanyi, 2004], and is the encoding we use to identify the partition of $\mathcal{A}$ of a given attribute clustering.

Encoding the Code Tables Next, we formalize how to encode the joint distribution of clusters by using *code tables*. A code table is a simple two-column table that has an entry for each possible value v from the domain of a cluster A_i. The left-hand column contains the value, and the right-hand column the corresponding code. By making sure the used codes are uniquely decodable, we can straightforwardly and losslessly translate between a code and the associated values. We give an example of a code table in Figure 4.2. Note that the frequencies of the value occurrences are not explicitly included in the code table, we will show below that we can derive these exactly from the lengths of the codes in the right-hand column. Also, if a value combination does not occur in the data (when its frequency is zero), it is omitted from the code table, and it does not get a code.

Code Table					
a	b	c	$code(A_i = v)$	$L(code(v))$	$fr(A_i = v)$
1	1	1	0	1 bits	50.00%
1	1	0	10	2 bits	25.00%
1	0	1	110	3 bits	12.50%
0	1	0	1110	4 bits	6.25%
0	0	0	1111	4 bits	6.25%

Figure 4.2: Example of a code table CT_i for an attribute cluster $A_i = \{a, b, c\}$ containing three binary attributes. The (optimal) codes depicted here are included for illustration purposes, in practice we are only interested in their lengths, $L(code(v))$. Assuming the database contains 256 transactions, the description length of this code table is 44 bits.

We encode the entries in the left-hand column as follows. Again using a model-to-data code, we can identify one of the values of an attribute a in $\log |dom(a)|$ bits, given that the size of the domain of a is $|dom(a)|$. Therefore, the description length of a value v of a set of attributes A_i is equal to

$$\sum_{a \in A_i} \log |dom(a)| = \log |dom(A_i)| \ .$$

The entries in the right-hand column of a code table are the codes that will be used to encode the data. Clearly, these codes should be as efficient as possible, which is why we employ an optimal prefix code. A prefix code (or, confusingly also known as a prefix-free code) is a code that can be uniquely decoded without requiring a prefix. A well-known result from information theory [Shannon, 1948, Cover and Thomas, 2006] states that a prefix code is *optimal* when

$$L(code(A_i = v)) = -\log fr(A_i = v)$$

for all $v \in dom(A_i)$.

The choice of one particular optimal coding scheme over another (for instance, Huffman coding [Cover and Thomas, 2006], which we used to obtain the codes in Figure 4.2), is irrelevant: recall that in MDL we are not concerned about actual materialized codes, but that we only want to measure complexity. Hence, we are only interested in the *lengths* of the theoretically optimal code the equation above gives us.

So, basically, the encoded length of the right-hand side column of a code table is simply the sum of each of the code lengths. However, in order to be able to uniquely decode a code table, we need to take one further element into account: we need to know when a bit representation of a code ends. Therefore, before a code is given, we specify its length in a fixed number of bits; since the frequency of an itemset that occurs in the data is at least $1/|\mathcal{D}|$, its corresponding code length is at most $\log|D|$, and hence the length of the code can be stated in $\log\log|\mathcal{D}|$ bits.

Now, a binary string so-representing a distribution viz. code table can unambiguously be decoded: by the association between code lengths and frequencies we can derive the frequencies of the values on the left-hand side by regarding the code lengths found on the right-hand side. By adding these frequencies together, we know the distribution is fully specified when this sum is equal to one. Therefore, the description length of the code table CT_i of an attribute cluster A_i can be computed as

$$L(CT_i) = \sum_{\substack{v \in dom(A_i) \\ fr(A_i=v)\neq 0}} \log|dom(A_i)| + \log\log|\mathcal{D}| - \log fr(A_i = v) \ .$$

Using this encoding, clusters that have many value instantiations that are distributed irregularly will require many bits. Similarly, attribute clusters with few, evenly distributed values, are much cheaper. In other words, we favor simple distributions.

Encoding the Data Now that we know how to encode a clustering, we can discuss how to determine $L(\mathcal{D} \mid \mathcal{C})$, the length of the encoded description of $\mathcal{D}$ given a clustering $\mathcal{C}$. This is done straightforwardly. First, each transaction $t \in \mathcal{D}$ is partitioned according to $\mathcal{C}$. We then encode a tuple by replacing the value in each part by the corresponding code in the code tables. Since an itemset $(A = v)$ occurs $|\mathcal{D}|fr(A = v)$ times in the data, the encoded size of $\mathcal{D}$ restricted to a single cluster A is equal to

$$\begin{aligned} L(\mathcal{D}_A \mid \mathcal{C}) &= -\sum_{t \in \mathcal{D}} \log fr(A = t_A) \\ &= -|\mathcal{D}| \sum_{v \in dom(A)} fr(A = v) \log fr(A = v) \\ &= |\mathcal{D}| H(A) \ . \end{aligned}$$

That is, the description length of the data with respect to an attribute cluster is proportional to its entropy, which is a measure of complexity.

Putting all of the above together, the definition of the total encoded size $L(\mathcal{C}, \mathcal{D})$ is as follows.

Definition 4.2. *The description length of a categorical dataset $\mathcal{D}$ using an attribute clustering $\mathcal{C} = \{A_1, \ldots, A_k\}$ of size k is defined as*

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

where

$$\begin{cases} L(\mathcal{D} \mid \mathcal{C}) & = & |\mathcal{D}| \sum_{i=1}^{k} H(A_i) \\ L(\mathcal{C}) & = & \log B_n + \sum_{i=1}^{k} L(CT_i) \\ L(CT_i) & = & \sum\limits_{\substack{v \in dom(A_i) \\ fr(A_i = v) \neq 0}} \log |dom(A_i)| + \log\log |\mathcal{D}| - \log fr(A_i = v) \end{cases}$$

Note that in the middle line, we can simply take the sum over all code table lengths, i.e., we do not need to indicate how many code tables there are (this is captured in the $\log B_n$ term), nor do we need to separate them with additional bits, since the descriptions of the code tables are self-terminating.

Further, remark that we implicitly make a basic assumption. To use the above definition, we assume that the number of attributes, their domains (their sizes in particular), and the number of transactions in the database are known beforehand to both sender and receiver. The reason not to include these in our formalization is simple: we are aiming to summarize a single given dataset. Clearly, these properties are constant over all models we would consider for one dataset, and hence, including these would increase the total description length only by a constant term, independent of the actual data instance and model, which makes no difference when comparing different clusterings. If for one reason or another it would be required to explicitly include the cost of these values into the total description length, one could do so by using a Universal Code for integers [Rissanen, 2007].

Problem Statement

Our goal is to discover a summary of a binary or categorical dataset, in the form of a partitioning of the attributes of the data; separate attribute groups

should be relatively independent, while attributes within a cluster should exhibit strong interaction. Formally, the problem we address is the following. Given a database $\mathcal{D}$ over a set of categorical attributes $\mathcal{A}$, find the attribute clustering $\mathcal{C}^*$ that minimizes $L(\mathcal{C}, \mathcal{D})$,

$$\mathcal{C}^* = \arg\min_{\mathcal{C}} L(\mathcal{C}, \mathcal{D}) = \arg\min_{\mathcal{C}} L(\mathcal{C}) + L(\mathcal{D} \mid \mathcal{C}) \ .$$

By this problem statement, we identify the optimal clustering by MDL. Note that we do not require the user to specify any parameters, e.g., a predetermined number of clusters k. Essentially, this is done automatically, as k follows from the clustering that has the shortest description.

Search Space

The search space to be considered for our problem is rather large. The total number of possible partitions of a set of n attributes equals the Bell number B_n, which is at least $\Omega(2^n)$. Therefore we cannot simply enumerate and test all possible partitions, except for the most trivial cases. We must exploit the structure of the search space somehow, in order to efficiently arrive at a good clustering of the attributes.

The *refinement* relation of partitions naturally structures the search space into a lattice. A partition $\mathcal{C}$ is said to refine a partition $\mathcal{C}'$ if for all $A \in \mathcal{C}$ there exists an $A' \in \mathcal{C}'$ such that $A \subseteq A'$. The transitive reduction of the refinement relation corresponds to the merger of two clusters into one (or conversely, splitting a cluster into two nonempty parts). As such, the search space lattice can be traversed in a stepwise manner.

The maximal, most refined clustering contains a singleton cluster for each individual attribute. We call this the *independence clustering*, denoted $\mathcal{I}$, since it corresponds to the independence distribution. On the other hand, the least refined, most coarse partition consists of only one cluster containing all attributes, which corresponds to the full joint distribution.

Note that $L(\mathcal{C}, \mathcal{D})$ is not (strictly or weakly) monotonically increasing or decreasing with respect to refinement, which would make determining the optimum trivial—this would require that there be at least one monotonic path between $\mathcal{I}$ and $\{\mathcal{A}\}$. As far as we know, other useful properties that could be exploited to identify the optimal clustering efficiently, such as convertibility, also do not apply to this setting. Therefore, we resort to heuristics.

In this case, we will employ a greedy hierarchical clustering strategy, the details of which are discussed in Section 4.3. Another option could be to consider to use a branch-and-bound method, however, we choose the greedy approach due to its very low computational complexity, as stated below.

Measuring Similarities between Clusters

Based on the definition of $L(\mathcal{C}, \mathcal{D})$ above, we can derive a similarity measure between clusters that will turn out to be useful later on. Let $\mathcal{C}$ be an attribute clustering and let $\mathcal{C}'$ be the result of merging two clusters A_i and A_j in $\mathcal{C}$, the merged cluster being denoted $A_{ij} = A_i \cup A_j$. Hence, $\mathcal{C}$ is a refinement of $\mathcal{C}'$. Then the difference between the description lengths of $\mathcal{C}$ and $\mathcal{C}'$ defines a similarity measure between A_i and A_j.

Definition 4.3. *For a dataset $\mathcal{D}$, we define the similarity of two clusters A_i and A_j in a clustering $\mathcal{C}$ as*

$$CS_{\mathcal{D}}(A_i, A_j) = L(\mathcal{C}, \mathcal{D}) - L(\mathcal{C}', \mathcal{D}) \ ,$$

where $\mathcal{C}' = \mathcal{C} \setminus \{A_i, A_j\} \bigcup \{A_i \cup A_j\}$. Whenever $\mathcal{D}$ is clear from the context, we simply write $CS(A_i, A_j)$.

If A_i and A_j are highly correlated, then their merger results in a better clustering with a lower description length, and hence their similarity is positive. Otherwise, if the clusters are more or less independent, their merger increases the description length, and their similarity is negative. The fact that CS expresses a similarity is further illustrated by the following property, which allows us to calculate cluster similarity without having to compute the total description length of a clustering.

Property 4.1. *Let $\mathcal{C}$ be an attribute clustering of $\mathcal{A}$, with $A_i, A_j \in \mathcal{C}$, and let $\mathcal{D}$ be a categorical dataset. Then*

$$CS_{\mathcal{D}}(A_i, A_j) = |\mathcal{D}| I(A_i, A_j) + \Delta L(CT) \ ,$$

where

$$I(A_i, A_j) = H(A_i) + H(A_j) - H(A_{ij})$$

is the mutual information between A_i and A_j, and

$$\Delta L(CT) = L(CT_i) + L(CT_j) - L(CT_{ij}) \ .$$

Property 4.1 shows that we can decompose cluster similarity into a mutual information term, and a term expressing the difference in code table description length. Both of these values are high when A_i and A_j are highly correlated, and low when they are more or less independent. It is interesting to note that *CS* is a local measure, that is, it only depends on A_i and A_j, and is not influenced by the other clusters in $\mathcal{C}$.

Canonical Description Length

It can be useful to compare the description length $L(\mathcal{C}, \mathcal{D})$ of a clustering $\mathcal{C}$ to a baseline description of the data; a description that does not use a model, but simply communicates the data 'as is'. However, there are many different ways to encode a database, all with different description lengths. A very natural approach is to use an encoding assuming a uniform distribution of the values. In this encoding, every value of an attribute a have the same length, namely $\log |dom(a)|$.

Definition 4.4. *The* canonical description length *of a dataset $\mathcal{D}$ is defined as*

$$L_c(\mathcal{D}) = |\mathcal{D}| \sum_{a \in \mathcal{A}} \log |dom(a)| = |\mathcal{D}| \log |dom(\mathcal{A})| \ .$$

For example, if $\mathcal{D}$ is a binary dataset, its canonical description length is equal to $|\mathcal{D}||\mathcal{A}|$ bits, i.e., each entry requires exactly one bit.

It can be argued that if for some model $\mathcal{C}$, we observe that $L(\mathcal{C}, \mathcal{D}) \ll L_c(\mathcal{D})$, we can safely say that $\mathcal{C}$ captures or explains a true regularity of the data, and hence $\mathcal{C}$ is a good model. On the other hand, if $\mathcal{D}$ mostly just consists of random noise (that is, it does not exhibit any structure), then for any model $\mathcal{C}$ we will find that $L(\mathcal{C}, \mathcal{D}) \geq L_c(\mathcal{D})$; if such is the case, we should conclude that the data does not exhibit any structure that our model can fit.

Refining the Encoding

The description length of an attribute clustering for a certain dataset as described above, is often called *two-part* or *crude* MDL, since it separately encodes the model and the data. Refined MDL [Grünwald, 2007] is an improvement of two-part MDL that does not explicitly encode model and data separately. Rather, it uses a so-called Universal Code to describe the data. A universal code is a code that we can employ without having any prior information, and

for which the lengths of the code words are within a constant factor of the optimal codes. As such, using universal codes avoids the potential bias that can occur in two-part MDL.

Although refined MDL has stronger theoretical foundations, it cannot be computed except for some special cases; those for which we have a universal code. As only a very limited number of universal codes is known, for distributions of relatively simple objects, the practical application of refined MDL is limited. However, we can refine our encoding somewhat by using a notion from modern MDL theory: *prequential* coding [Grünwald, 2007].

By prequential coding, we are not explicitly transmitting the codes that will be used to encode the data—and so, we lose any potential bias that the explicit encoding of those codes might give. Rather than using explicit fixed code words, we encode the data using an implicitly assumed distribution on the data, and iteratively update this distribution (and hence, the codes) after every transmitted/received code word. As such, by each update the distribution changes slightly, and hence so does the underlying code table, and thus the encoding of the data that is to be transmitted.

It may seem that by not explicitly including a cost for using long code words, we do not penalize against this. However, the penalty is actually hidden in the description of the data, since there is always an overhead incurred by using a code table that does not exactly represent the distribution of the data, i.e., by using a code table that is suboptimal.

Before we discuss the size of this penalty, let us first specify how we can employ prequential coding in our setting. When we start transmitting the data, instead of using the actual attribute-value frequencies of the data, we start without such prior knowledge and use a uniform distribution over these frequencies, by assigning some constant occurrence count c to each possible value in the domain of an attribute set. Then, we transmit the first value by encoding it using an optimal code derived from this usage distribution, as defined earlier in this section. After this value is transmitted, we adjust the distribution by incrementing the occurrence count of that particular value by 1. Clearly, this changes the usage distribution, and hence the codes in the code table must be recomputed in order to remain optimal. We simply repeat this process iteratively for every transaction, until we have transmitted all of the data. After all the data has been sent, the codes in the code table are the same as when we would have transmitted them explicitly—that is, if we disregard the c constant.

The choice of the initial count c has an influence on the overhead. If c is taken very large, updating the code table with new data has little effect. For $c \to \infty$, the overhead tends to $KL(\mathbf{u} \| fr)$, where $\mathbf{u}$ is the uniform distribution. On the other hand, if c is extremely small, say some $\epsilon > 0$, adding data will wildly change the code table to extremes, leading to larger codes for certain values, especially when the data is not very large. In both cases, for $|\mathcal{D}| \to \infty$, the relative overhead converges to zero, however, in practice the amount of available data is limited, and therefore c does have an impact on the encoded length. A natural and often chosen value is $c = 0.5$ [Grünwald, 2007].

Let us consider the description length of the data for a single attribute cluster, using prequential coding. Let l be the number of distinct values that can occur, let m_i be the count of each such value v_i in the domain of the cluster (i.e., $m_i = supp(v_i)$), and let $m = \sum_{i=1}^{l} m_i$ be the total number of transactions (i.e., $m = |\mathcal{D}|$). If in the j-th encountered tuple, we observe a value v that has up till now occurred m_v times, then its current frequency in the code table is $(m_v + 0.5)/(j + 0.5\,l)$. Hence its code length is $-\log\left((m_v + 0.5)/(j + 0.5\,l)\right)$. Adding all code lengths together, we obtain

$$
\begin{aligned}
-\log \frac{\prod_{i=1}^{l} \prod_{j=0}^{m_i - 1} (j + 0.5)}{\prod_{j=0}^{m-1} (j + 0.5\,l)} \quad &= \quad \log \frac{\Gamma(m + 0.5\,l)/\Gamma(0.5\,l)}{\prod_{i=1}^{l} \Gamma(m_i + 0.5)/\Gamma(0.5)} \\
= \quad \log \Gamma(m + 0.5\,l) - \log \Gamma(0.5\,l) - \sum_{i=1}^{l} \left(\log\left((2m_i - 1)!!\right) - m_i \right) ,
\end{aligned}
$$

where Γ is the gamma function, which is an extension of the factorial function to the complex plane, that is, $\Gamma(x+1) = x\Gamma(x)$, with relevant base cases $\Gamma(1) = 1$ and $\Gamma(0.5) = \sqrt{\pi}$.

Interestingly, even though the distribution and hence the code lengths constantly change during the encoding of $\mathcal{D}$, the total description length does not depend on the order in which the tuples are processed.

The number of values l in the formula above can be extremely large if we take it to be the size of the domain of a cluster. However, in practice it will be a lot smaller than the full domain, especially for large attribute sets. Therefore, we can first specify the values that can occur. We do this using a model-to-data code for all nonempty subsets of $dom(A)$, which takes $\log(2^{|dom(A)|} - 1)$ bits.

Definition 4.5. *The refined description length of the data for a single cluster A, using prequential coding, is given by*

$$L_r(A, \mathcal{D}) = \log(2^{|dom(A)|} - 1) + \log\Gamma(m + 0.5\,l) - \log\Gamma(0.5\,l) - \sum_{i=1}^{l} \left(\log\left((2m_i - 1)!!\right) - m_i \right) ,$$

where $l = |\{v \in dom(A) \mid fr(A = v) \neq 0\}|$ is the number of values with non-zero support in the data. The refined description length of an attribute clustering $\mathcal{C} = \{A_1, \ldots, A_k\}$, using prequential coding, is then equal to

$$L_r(\mathcal{C}, \mathcal{D}) = \log(B_n) + \sum_{i=1}^{k} L_r(A_i, \mathcal{D}) .$$

Querying a Summary

Besides providing insight into which attributes interact most strongly, and which of their values typically co-occur, our summaries can also be used as surrogates for the data. That is, they form probabilistic models, being the product of independent distributions on clusters of attributes that can easily and efficiently be queried.

For categorical data, querying comes down to calculating marginal probabilities, i.e., determining itemset frequencies. The frequency of an itemset $(X = v)$ can simply be estimated from an attribute clustering by splitting up the itemset over the clusters, and calculating the product of the marginal frequencies of the subsets in each separate cluster.

This splitting up is justified by the fact that all relevant correlations between attributes are captured by a good clustering. Note that the more attributes are clustered together, the more detail the corresponding code table contains about their correlations, hence making better frequency estimations possible. In fact, for the trivial complete clustering, where all attributes are grouped together, we will obtain exact frequencies. As our summaries are constructed with the goal of capturing the key correlations as well as possible using as few and simple code tables as possible, we expect to obtain highly accurate, but not exact frequency estimations.

Definition 4.6. *Let $\mathcal{C} = \{A_1, \ldots, A_k\}$ be a clustering of $\mathcal{A}$, and let $(X = v)$ be an itemset, with $v \in dom(X)$. Then the frequency of $(X = v)$ is estimated as*

$$\hat{fr}(X = v) = \prod_{i=1}^{k} fr(X \cap A_i = v_i) \ ,$$

where v_i is the sub-vector of v for the corresponding attributes in $X \cap A_i$.

For instance, let $\mathcal{A} = \{a, b, c, d, e, f\}$ and $\mathcal{C} = \{abc, de, f\}$, and consider the itemset $(abef = 1)$, then $\hat{fr}(abef = 1) = fr(ab = 1) \cdot fr(e = 1) \cdot fr(f = 1)$.

Since the code table for each cluster A_i contains the frequencies of the values for A_i—because of the one-to-one mapping between code length and frequency—we can use our clustering model as a very efficient surrogate for the database. For a cluster A_i, let Ω_i be the subset of values in $dom(A_i)$ having a nonzero frequency in $\mathcal{D}$. The complexity of frequency estimation is then upper bounded by

$$O\left(\sum_{i=1}^{k} |A_i \cap X||\Omega_i|\right) \leq O\left(|X||\mathcal{D}|\right) \ .$$

If $|\Omega_i| \ll |\mathcal{D}|$, querying the summary is significantly faster than querying the data. Note that this automatically holds for small clusters for which $|dom(A)| < |\mathcal{D}|$.

4.3 Mining Attribute Clusterings

Now that we have defined how we can identify the best clustering, we need a way to *discover* it. In this section we present our algorithm, and investigate its properties and computational complexity.

Algorithm

As discussed in the previous section, the search space we have to consider is extremely large, making it is infeasible to examine the search space exhaustively, and thus we settle for heuristics. In this section we introduce our algorithm, which finds a good attribute clustering $\mathcal{C}$ for a dataset $\mathcal{D}$, with a low description length $L(\mathcal{C}, \mathcal{D})$.

Algorithm 4.1: AttributeClustering

input : a categorical dataset $\mathcal{D}$ over a set of attributes $\mathcal{A}$
output: an attribute clustering $\mathcal{C} = \{A_1, \ldots, A_k\}$ minimizing $L(\mathcal{C}, \mathcal{D})$

1 $\mathcal{C} \leftarrow \{\{a\} \mid a \in \mathcal{A}\}$
2 $\mathcal{C}_{\text{min}} \leftarrow \mathcal{C}$
3 compute and store $CS_{\mathcal{D}}(A_i, A_j)$ for all $i \neq j$
4 **while** $|\mathcal{C}| > 1$ **do**
5 $\quad A_i, A_j \leftarrow \arg\max_{i,j} CS_{\mathcal{D}}(A_i, A_j)$
6 $\quad \mathcal{C} \leftarrow \mathcal{C} \setminus \{A_i, A_j\} \bigcup \{A_i \cup A_j\}$
7 $\quad$ compute and store $CS_{\mathcal{D}}(A_{ij}, A_l)$ for all $l \neq ij$
8 $\quad$ **if** $L(\mathcal{C}, \mathcal{D}) < L(\mathcal{C}_{\text{min}}, \mathcal{D})$ **then**
9 $\quad\quad \mathcal{C}_{\text{min}} \leftarrow \mathcal{C}$
10 $\quad$ **end**
11 **end**
12 **return** $\mathcal{C}_{\text{min}}$

We use a greedy bottom-up hierarchical clustering algorithm that traverses the search space by iteratively merging clusters such that in each step the description length is minimized. The pseudo-code is given in Algorithm 4.1. We start by placing each attribute in its own cluster (line 1), which corresponds to the independence model. Then, we iteratively find the pair of clusters whose merger results in a clustering with the smallest description length. From Section 4.2 we know that this is the pair of clusters with the highest similarity, which can be computed locally (5). The clusters are merged (6), and the algorithm continues. We maintain the clustering with the shortest description (8–9), and finally return the best clustering (12).

This results in a hierarchy of clusters, which can be represented visually as a dendrogram, as shown in Figure 4.3. The clustering at the bottom corresponds to the independence distribution, while the clustering at the top represents the joint empirical distribution of the data. In the figure, merges that result in a lower description length are depicted with solid lines. Their height corresponds to the (cumulative) decrease in description length. An advantage of this approach is that it allows us to visualize how the clusters were formed, and how they are structured internally.

The graph in Figure 4.4 shows how the description length behaves as a function of k (the number of clusters), during a run of the algorithm on the binary *Connect* dataset. Starting at $k = n$, the description length $L(\mathcal{C}, \mathcal{D})$ gradually decreases as correlated clusters are merged. This indicates that there is structure present in the data, which is exploited to obtain a shorter description of it. Note that the description length of the data, $L(\mathcal{D} \mid \mathcal{C})$, decreases monotonically, since a less refined clustering uses more information, and hence fits the data better. The total description length continues to decrease until $k = 7$, which yields the best clustering found for this dataset. Afterwards, $L(\mathcal{C}, \mathcal{D})$ increases again, due to the fact that the decrease in $L(\mathcal{D} \mid \mathcal{C})$ does not outweigh the dramatic increase of $L(\mathcal{C})$, which means that the models are getting far too complex past this point.

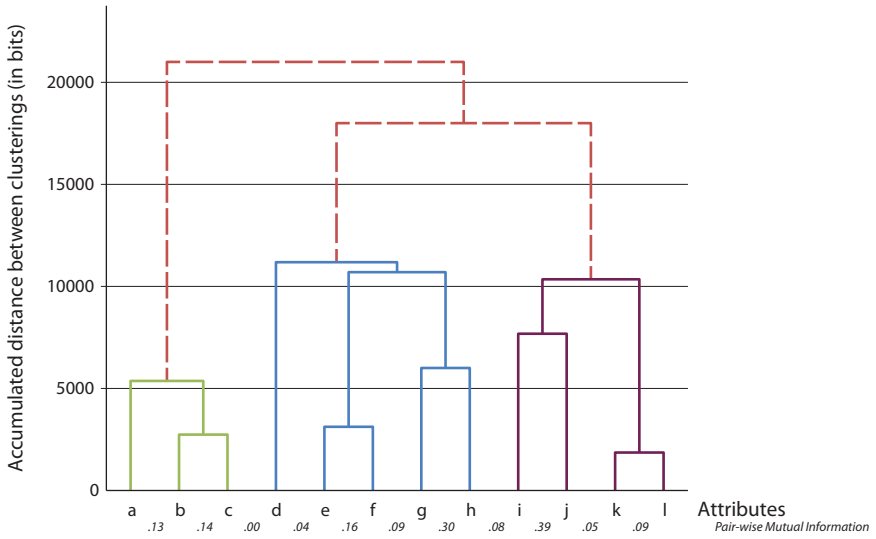


Figure 4.3: Example of a dendrogram depicting a hierarchy of attribute clusterings, for a subset of the attributes of the categorical *Markov* dataset. Merges that save bits are depicted with solid lines; their height corresponds to the cumulative drop in description length. There are three attribute clusters in this example, namely *a–c*, *d–h*, and *i–l*.

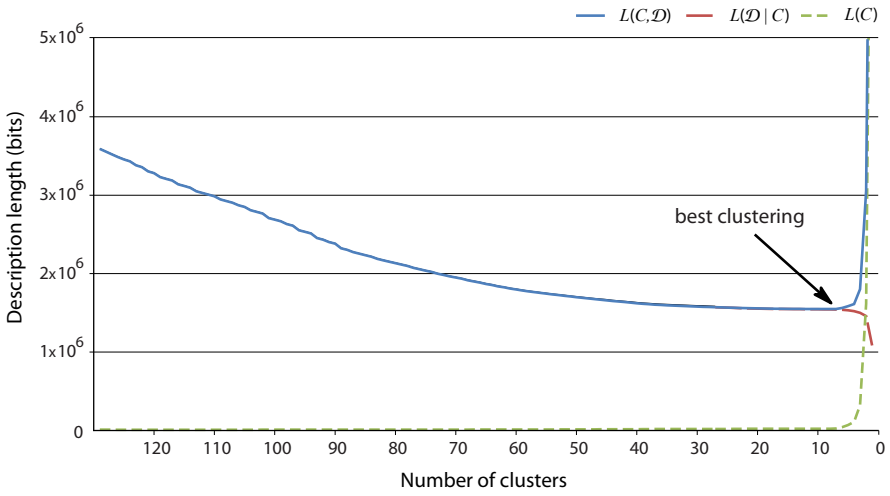


Figure 4.4: Evolution of the encoded length $L(\mathcal{C}, \mathcal{D})$ with respect to the number of clusters k, on the binary *Connect* dataset. The clustering with the lowest description length is located at $k = 7$.

Stopping Criterion

Figure 4.4 seems to suggest that $L(\mathcal{C}, \mathcal{D})$, as a function of k, has a single minimum. That is, first the description length decreases until a local optimum is reached, and afterwards the description length only increases. Naturally, the question arises whether this is the case in general; if so, the algorithm can terminate as soon as a local minimum is detected. Intuitively, we would expect that if the current best cluster merger increases the total description length, then any future merger is probably also a bad one. However, the following counterexample shows that this is not necessarily true.

Consider the dataset $\mathcal{D}$ in Figure 4.5 with four binary attributes $\{a, b, c, d\}$. Assume that a, b, and c are independent, and that for every transaction of $\mathcal{D}$ it holds that $d = a \oplus b \oplus c$, where $\oplus$ denotes exclusive or (XOR). Now, using this dependency, let $\mathcal{D}$ contain a transaction for every $v \in \{0, 1\}^3$ as values for abc, with multiplicity, say, 10. Then every pair of clusters whose union contains strictly less than four attributes (e.g., $A_i = ab$ and $A_j = d$) is independent. As the algorithm starts to merge clusters, the data description length $L(\mathcal{D} \mid \mathcal{C})$

a	b	c	d	multiplicity
0	0	0	0	$\times$ 10
0	0	1	1	$\times$ 10
0	1	0	1	$\times$ 10
0	1	1	0	$\times$ 10
1	0	0	1	$\times$ 10
1	0	1	0	$\times$ 10
1	1	0	0	$\times$ 10
1	1	1	1	$\times$ 10

Figure 4.5: An exclusive or (xor) dataset. The last attribute equals the xor of the first three. The rightmost column denotes the number of times each transaction occurs, i.e., there are 80 transactions.

remains constant (namely $4|\mathcal{D}|$), but the code tables become more complex, and thus $L(\mathcal{C})$ and $L(\mathcal{C}, \mathcal{D})$ increase. Only at the last step, when the two last clusters are merged, can the structure be captured and $L(\mathcal{D} \mid \mathcal{C})$ decreases to $3|\mathcal{D}|$, leading to a decrease of the total description length.

Note, however, that the drop in encoded length in the last step depends on the number of transactions in the database. In the above example, if the multiplicity of every unique transaction is 10, the complete clustering would be preferred over the independence clustering. However, if there are fewer transactions (say, every unique transaction occurs only once), then even though the dependencies are the same, the algorithm decides that the best clustering corresponds to the independence model, i.e., that there is no significant structure. Intuitively this can be explained by the fact that if there is only a small number of samples then the observed dependencies might be coincidental, but if many transactions exhibit it, the dependencies are truly present. This is one of the advantages of using mdl to select models.

This example shows that in general we should not stop the algorithm when the description length attains a local minimum. Nevertheless, it is a very synthetic example with a strong requirement on the number of transactions. For instance, if we generalize the xor example to 20 attributes, the minimum number of transactions for it to be detected already runs in the millions. Moreover, in none of our experiments with real data did we encounter a locally minimal description length that was not a global minimum.

Therefore, we can say that in practice it is acceptable to stop the algorithm when we encounter a clustering with a locally minimal description length.

Algorithmic Complexity

Naturally, a summarization method should be fast, because our aim is to get a quick overview of the data. Notwithstanding the search for efficient algorithms in data mining, in practice many exhibit an exponential runtime. Here we show that our algorithm is polynomial in the number of attributes.

In the first iteration, we compute the description length of each singleton $\{a\}$, and then determine which two clusters to merge. To do this, we must compute $O(N^2)$ cluster similarities, where $N = |\mathcal{A}|$. Since we might need some of these similarities later on, we store them in a heap, such that we can easily retrieve the maximum. Now say that in a subsequent iteration k we have just merged clusters A_i and A_j into A_{ij}. Then we can delete the $2k-3$ similarities from the heap that are no longer of use, and we need to compute and insert $k-2$ new similarities, namely those between A_{ij} and the remaining clusters. Since heap insertion and deletion is logarithmic in the size of a heap, maintaining the similarities in an iteration with k clusters, takes $O(k \log k)$ time. The initial filling of the heap takes $O(N^2 \log N)$. Since the algorithm performs at most N iterations, we see that the time complexity of maintaining the cluster similarities is $O(N^2 \log N) + \sum_{k=N}^{1} O(k \log k) = O(N^2 \log N)$.

Next we discuss the complexity of computing the similarities. Say that in an iteration k, we must compute the similarities $CS_{\mathcal{D}}(A_i, A_j)$ between the last merged cluster A_i and all other remaining clusters A_j. This requires collecting all nonzero frequencies $fr(A_{ij} = v)$, and we do this by simply iterating over all transactions t and computing $|t \cap A_{ij}|$, which takes $O(N|\mathcal{D}|)$ per pair in the worst case. Computing the initial similarities between all pairs of singletons takes $O(N^2|\mathcal{D}|)$. Hence, over all iterations, the similarity computations take $O(N^2|\mathcal{D}|) + \sum_{k=N}^{1} O(kN|\mathcal{D}|) = O(N^3|\mathcal{D}|)$.

Therefore, the total time complexity of AttributeClustering is

$$O(N^3|\mathcal{D}|)\,.$$

The dominant cost in terms of memory usage comes from the heap containing the cluster similarities. Since at any point there are at most N different clusters, the memory complexity of the algorithm is $O(N^2)$.

4.4 Alternative Approaches

Since we employ a greedy algorithm, we do not necessarily find the globally optimal attribute clustering. Here we discuss some potential alternative search strategies.

Vertical Clustering

A very straightforward approach to clustering attributes is to use any one of the wealth of existing clustering algorithms that cluster rows of data, and adapt it to our setting. By simply transposing the data, and applying say, k-means, we are done. However, the issue is not that trivial. While transposing data could conceptually make sense for binary data, this is arguably not the case for categorical data in general. Even so, most, if not all, clustering algorithms require a distance measure between tuples, which means we would need a distance measure for attributes. For categorical data, an often used measure is Hamming distance, the number of locations with unequal values. If two attributes have the same value in many transactions, clearly they are very similar. However, the converse is not true, e.g., two binary attributes that are each other's inverse are clearly correlated. Moreover, attributes with different domains would be incomparable. A similar argument can be made for Manhattan or Euclidean distances.

Turning to the area of information theory, a good choice is to use a distance measure based on mutual information. Specifically, let a_1 and a_2 be two attributes, then we could use the distance $d(a_1, a_2) = H(a_1, a_2) - I(a_1, a_2)$. This is a proper metric: it is symmetric, nonnegative, zero if and only if both attributes are a function of one another (i.e., there exists a bijection between them), and it obeys the triangle inequality. It reaches a maximum of $H(a_1, a_2)$ when a_1 and a_2 are independent. We could additionally choose to normalize d, e.g., by dividing by $H(a_1, a_2)$.

However, using pairwise distances on attributes can only get us so far, as the following small example shows. Consider a dataset with six binary attributes that is the cartesian product of two (independent) xor truth tables. Each pair of attributes in this dataset is independent, and has the same distance of 2. Consequently, no algorithm based on pairwise distances will be able to detect that there are two groups of three correlated attributes in this dataset, regardless of the algorithm that is used.

Finally, we might consider directly using the mutual information between clusters. Let A_1 and A_2 be two attribute clusters, then the distance between the clusters is $d(A_1, A_2) = H(A_1, A_2) - I(A_1, A_2)$. In this case, the best clustering is always the complete clustering. In the absence of a complexity penalty term, this requires the user to specify a parameter k for the number of clusters. Moreover, the choice of a specific search strategy is crucial here. The mutual information between a large and a small cluster will typically be larger than the mutual information between two small clusters. If one cluster becomes large, it will typically have a high similarity with other clusters. Hence, it will tend to 'eat up', as it were, the smaller clusters, rather than that the small clusters would merge together, resulting in a very heterogeneous, unbalanced clustering, which is not likely to be a desirable result.

Divisive Hierarchical Clustering

Instead of taking a bottom-up, agglomerative approach, a top-down divisive hierarchical clustering algorithm could be considered. Such an algorithm would start at the least refined clustering, and then iteratively split a cluster such that the description length decreases maximally. However, this approach is far from feasible, since already in the first step, the algorithm needs to consider $2^{|\mathcal{A}|-1} - 1$ possible splits (i.e., there are $2^{|\mathcal{A}|} - 2$ proper nonempty subsets; their complements describe the same splits). While a divisive approach does consider more clusterings than an agglomerative one, and hence could possibly discover a clustering with a lower description length, its exponential behavior makes it unsuitable for summarization purposes.

Beam Search

Beam search attempts to balance the level-wise, large-coverage approach of breadth-first search, with the memory efficiency of depth-first or greedy search. Starting with an initial state, a *beam* containing the b best solutions is maintained at each level, where b is called the beam width. The beam at the next level consists of the b best scoring neighbor states of the solutions in the current beam. Note that when the beam width is set to one, beam search is equivalent to the greedy algorithm. On the other hand, if we set b to infinity, the algorithm is simply breadth-first search. Since beam search considers a larger part of the search space, it tends to perform better for larger beams.

However, in practice the beam tends to exhibit little variability, and thus often does not lead to considerably better results. Further, both time and memory complexity are multiplied by a factor b. We empirically compare our greedy agglomerative approach to beam search in the experiments.

Simulated Annealing

Simulated annealing [Kirkpatrick, 1984] is a probabilistic optimization technique that tries to avoid finding a locally but not globally optimal solution, by not only greedily moving towards better neighbor solutions in the search space, but also heuristically moving to worse ones. The idea is borrowed from the physical process of annealing, where a material is heated and then gradually cooled down to form a crystal structure, which is a low energy state. The probability of transitioning to a neighboring state is dependent on the difference in energy of both states, and the current temperature of the system. As the temperature decreases, the probability of going to a higher energy state decreases. Many parameters influence the end result, such as the initial temperature, the cooling schedule, the material, etc.

In our setting, states are attribute clusterings, and the neighbors of a clustering $\mathcal{C}$ are all clusterings that are the result of either merging two of its clusters, or splitting one. The description length acts as the energy function. For the temperature, we employ a geometric cooling schedule: in iteration i, the temperature is decreased by a factor α: $t_{i+1} = \alpha t_i$. In this way, the number of iterations and the cool down rate are directly linked. We choose $t_0 = 1$, and set α such that $t_n = \alpha^n$ becomes equal to zero in double floating point precision. When in a certain state $\mathcal{C}$, we randomly choose to do either a split or a merge, and then uniformly pick a neighbor state $\mathcal{C}'$, and compute the difference in description length. The energy is normalized by estimating the maximal possible difference by sampling.

The probability of switching from a state $\mathcal{C}$ to a state $\mathcal{C}'$ in the i-th iteration is given by

$$\Pr(\mathcal{C} \to \mathcal{C}') = \min\left(\exp \frac{-CS(\mathcal{C},\mathcal{C}')/Z}{\lambda \alpha^i}, 1\right) .$$

The λ constant acts as a scaling parameter, and Z is a normalization factor. If CS is positive, i.e., if $\mathcal{C}'$ has a lower description length than $\mathcal{C}$, the transition is always made. On the other hand, if CS is negative, $\mathcal{C}'$ has a higher description length, and the transition is only made probabilistically based on

the magnitude of *CS* and the current temperature, which decreases over time. The algorithm is initiated in a random clustering.

Since simulated annealing has the potential to move away from a local minimum, unlike true greedy approaches, it has the potential to find better solutions. However, due to its inherently non-deterministic nature, a single run is not likely to give us a satisfactory answer—rather, the algorithm will have to be run many times, and even then we still have no guarantee that the best solution we encountered is anywhere close to the global optimum. Furthermore, by considering both groupings and splits of the current clustering, this approach is computationally heavy, as we have to calculate up to an exponential number of switching probabilities per step. We empirically compare our greedy agglomerative approach to simulated annealing in the experimental section.

4.5 Related Work

The main goal of our approach is to offer a good first impression of a categorical dataset. For numerical data, averages and correlations can easily be computed, and more importantly, are informative. For binary and categorical data, such informative statistics beyond simple counts are not readily available. As such, our work can be seen as to provide an informative 'average' for categorical data; for those attributes that interact strongly, it shows how often their value combinations occur.

Most existing techniques for summarization are aimed at giving a succinct representation of a given collection of itemsets. Well-known examples include closed itemsets [Pasquier et al., 1999] and non-derivable itemsets [Calders and Goethals, 2007], which both provide a lossless reduction of the complete pattern collection.

A lossy approach that provides a succinct summary of the patterns was proposed by Yan et al. [2005]. The authors cluster groups of itemsets and describe these groups using *profiles*, which can be characterized as conditional independence distributions on a subset of the items, which can subsequently be queried. Experiments show that our method provides better frequency estimates, while requiring fewer clusters than profiles. Wang and Karypis [2004] give a method for directly mining a summary of the frequent pattern collection for a given *minsup* threshold. Han et al. [2007] provide a more complete overview of pattern mining and summarization techniques.

For directly summarizing data rather than pattern sets, however, fewer proposals exist. Chandola and Kumar [2005] propose to induce k transaction templates such that the database can be reconstructed with minimal loss of information. Alternatively, the Krimp algorithm [Vreeken et al., 2011, Siebes et al., 2006] selects those itemsets that provide the best lossless compression of the database, i.e., the best description. While it only considers the 1s in the data, it provides high-quality and detailed results, which are consequently not as small and easily interpreted as our summaries. Though the Krimp code tables can generate data virtually indistinguishable from the original [Vreeken et al., 2007], they are not probabilistic models and cannot be queried directly, so they are not immediately suitable as data surrogates.

Wang and Parthasarathy [2006] build probabilistic Maximum Entropy models of the data by incrementally adding those itemsets into the model that deviate more than a given error threshold—considering the data as a bag of independent samples. The approach ranks and adds itemsets in level-wise batches, i.e., first itemsets of size 1, then of size 2, and so on. In Chapter 6 we improve over this method by avoiding the level-wise approach, and instead iteratively incorporate the itemset that increases the likelihood of the model most. Furthermore, by employing the Bayesian Information Criterion and mdl, the set of most informative itemsets can be identified without requiring any parameters. De Bie [2011b] proposes to use the Maximum Entropy principle to instead model the data as a whole—considering it as a monolithic sample. In [Kontonasios and De Bie, 2010] it is shown that this model can be used very effectively to rank and select itemsets with regard to the information they provide, while also taking their complexity into account.

All the above-mentioned techniques differ from our approach in that they model the data in relatively high detail using itemsets, whereas we provide a more high level summary that identifies the most strongly interacting categorical attributes, and their most prevalent attribute-value combinations.

Somewhat related to our method are low-entropy sets [Heikinheimo et al., 2007], attribute sets for which the entropy lies below a given threshold. As entropy is strongly monotonically increasing, typically many low-entropy sets are discovered even for low thresholds. This pattern explosion is often even worse than in frequent itemset mining. Heikinheimo et al. [2009] introduced a filtering proposal called Less, to select those low-entropy sets that together describe the data well. In our approach, instead of filtering, we discover attribute sets with low entropy directly on the data.

Orthogonal to our approach, the maximally informative k-itemsets (miki for short) by Knobbe and Ho [2006a] are k items (or patterns) that together split the data optimally, found through exhaustive search. Bringmann and Zimmermann [2007, 2009] propose a greedy alternative to this exhaustive method that can consider larger sets of items. Our approach groups attributes together that correlate strongly, so the correlations between groups are weak.

Since our approach employs clustering, the work in this field is not unrelated. However, clustering is foremost concerned with grouping rows together, typically requiring a distance measure between objects. Co-clustering [Chakrabarti et al., 2004], or bi-clustering [Pensa et al., 2005] is a type of clustering in which clusters are simultaneously detected over both attributes and rows. Chakrabarti et al. [2004] also employ the MDL principle to identify the best clustering, however, whereas our approach is to identify groups of categorical attributes for which the attribute-value combinations show strong correlation, their approach identifies locally dense areas in sparse binary matrices, and is not trivially extendable for categorical data.

Au et al. [2005] present an algorithm that clusters features in gene expression data, taking into account a target attribute, in order to do classification. To this end the authors introduce a distance measure between attributes, based on their mutual information. However, as argued in Section 4.4, this may not always be a good choice. The algorithm is a variant of the k-means algorithm—using mode attributes rather than means. Since the number of clusters k is a parameter of the algorithm, the authors propose to simply run the algorithm for all possible values of k, and select the clustering minimizing a defined score. The application of the paper is classification of gene expression data, which often suffers from the $n \ll p$ problem, i.e., the number of variables (p) is far greater than the number of samples (n). This has implications for the significance of the correlation (or mutual information) between attributes, and might lead to overfitting. Our algorithm takes this into account by using the MDL principle.

Dhillon et al. [2003] provide an algorithm that clusters words in text data for classification. For each word cluster, a feature is constructed as a weighted average of the distributions of the separate words in the cluster. While this inherently discards some information, it also reduces the number of features, making classification easier, and does not discard as much information as feature selection would. To find the clusters, an algorithm similar to k-means is presented. The word clusters are then used in a naive Bayes classifier. The

number of clusters k is a parameter of the algorithm, however, in this case it arguably is not a goal to find an optimal clustering, but to reduce the number of features purely for performance reasons, and hence it is probably desirable to be able to control this parameter.

Our formalization can also be regarded as a distance measure between categorical attributes (or, categorical data in general). As such, the method by Das et al. [1997] is both interesting and relevant. There, the authors take an orthogonal approach by measuring the similarity of a set of binary attributes not by regarding the similarity over the selected attributes, but by considering the marginal frequencies of a set of *other* attributes, called probes. Although experimental results show that some true similarities between attributes are captured, the measure and its results do lack an intuitive interpretation, and the selection of probes is manual, requiring further development in order to be used in practice.

4.6 Experiments

In this section we experimentally evaluate our method and validate the quality of the discovered attribute clusterings.

Setup

We implemented our algorithm in C++, and make the source code available for research purposes.[1] All experiments were executed on 2.67GHz (six-core) Intel Xeon X5650 machines with 12GB of memory, running Linux. All reported runtimes were obtained using a single-threaded version of the implementation. Unless stated otherwise, empirical p-values were calculated against the scores of 1 000 randomized models or datasets, providing a resolution of at most 0.1%. For all experiments we recorded memory usage during summary construction, which amounted to at most a few megabytes excluding the database itself.

Datasets

We evaluate our method on nineteen different datasets, covering a wide range of different data characteristics. We use five synthetic datasets and fourteen

[1] http://www.adrem.ua.ac.be/implementations

Table 4.1: The basic characteristics of the datasets used in the experiments. Shown are the number of attributes $|\mathcal{A}|$, the number of transactions $|\mathcal{D}|$, the canonical description length $L_c(\mathcal{D})$, and the description length of the independence clustering $L(\mathcal{I}, \mathcal{D})$.

Binary Data	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{D}\rvert$	$L(\mathcal{I}, \mathcal{D})$	$L_c(\mathcal{D})$
Independent	50	20 000	895 079	1 000 000
Markov	50	20 000	999 159	1 000 000
DAG	50	20 000	970 485	1 000 000
Accidents	468	340 183	25 991 622	159 205 644
BMS-Webview-1	497	59 602	1 201 522	29 622 194
Chess	75	3 196	142 812	239 700
Connect	129	67 557	3 593 260	8 714 853
DNA Amplification	391	4 590	191 428	1 794 690
Mammals	121	2 183	121 572	264 143
MCADD	198	31 924	2 844 465	6 320 952
Mushroom	119	8 124	443 247	966 756
Pen Digits	86	10 992	605 413	945 312
Categorical Data				
Independent	50	20 000	2 037 016	2 109 825
Markov	50	20 000	2 036 871	2 109 824
Chess	37	3 196	71 651	120 122
Connect	43	67 557	2 013 066	4 604 239
MCADD	22	31 924	1 966 658	2 168 968
Mushroom	23	8 124	267 334	388 268
Pen Digits	17	10 992	377 801	430 723

real-world and benchmark datasets, all of which, save one, are publicly available. Their basic characteristics are presented in Table 4.1.

The binary and categorical *Independent* datasets have independent attributes with randomly drawn frequencies. The attributes of the *Markov* datasets form a Markov chain. In the binary version each attribute is a copy of the previous one with a random copy probability, the first attribute having a 50% probability of being one. Similarly, the categorical version has attributes with up to eight values per attribute, and each attribute depends on the previous

one according to a randomly generated contingency table. The categorical *Independent* and *Markov* data are generated such that they have the same column margins. The *DAG* dataset is generated according to a directed acyclic graph among its binary attributes. Each attribute depends on at most four of its preceding attributes, according to randomly generated contingency tables.

Next, we use fourteen real-world and benchmark datasets. The well-known categorical *Chess*, *Connect*, and *Mushroom* datasets were obtained from the UCI Machine Learning Repository [Frank and Asuncion, 2010]. Their binary counterparts were obtained from the FIMI Dataset Repository [Goethals and Zaki, 2003], and simply contain one binary attribute for each attribute-value pair in the categorical versions. The *Accidents* and *BMS-Webview-1* datasets were also obtained from the FIMI Dataset Repository.

The *DNA Amplification* database contains data on DNA copy number amplifications. Such copies activate oncogenes and are hallmarks of nearly all advanced tumors [Myllykangas et al., 2006]. Amplified genes represent targets for therapy, diagnostics and prognostics.

The *Mammals* data[2] consists of presence records of European mammals within geographical areas of 50×50 km^2 [Mitchell-Jones et al., 1999].

The categorical *MCADD* data was obtained from the Antwerp University Hospital. MCADD (Medium-Chain Acyl-coenzyme A Dehydrogenase Deficiency) [Baumgartner et al., 2005, Van den Bulcke et al., 2011] is a deficiency newborn babies are screened for during a Guthrie test on a heel prick blood sample. The instances are represented by a set of 21 features: twelve different acylcarnitine concentrations measured by tandem mass spectrometry (TMS), together with four of their calculated ratios and five other biochemical parameters, each of which we discretized using k-means clustering with a maximum of ten clusters per feature.

Finally, the *Pen Digits* data was obtained from the LUCS-KDD data library [Coenen, 2003], and contains handwritten samples of the digits 0–9. The attributes correspond to eight x and y coordinates on a grid, describing the trace of a certain digit, which is indicated by the class label.

Evaluation

Table 4.2 presents an overview of the results of our algorithm for the used datasets. We show the number of clusters k in the clustering that our algo-

[2] http://www.european-mammals.org/

Table 4.2: Results of our attribute clustering algorithm. Shown are the number of identified groups of attributes k, the description length $L(\mathcal{C}, \mathcal{D})$ and the wall clock time used to discover the clusterings. Further, we show the relative description lengths $\frac{L(\mathcal{C},\mathcal{D})}{L(\mathcal{I},\mathcal{D})}$ and $\frac{L(\mathcal{C},\mathcal{D})}{L_c(\mathcal{D})}$ with respect to the description length of the independence clustering and canonical description length

				Compression ratio	
Binary Data	k	$L(\mathcal{C}, \mathcal{D})$	*time*	$L(\mathcal{I}, \mathcal{D})$	$L_c(\mathcal{D})$
Independent	49	895 078	1 s	99.9%	89.5%
Markov	15	884 943	4 s	88.6%	88.5%
DAG	12	797 588	5 s	82.2%	79.8%
Accidents	116	16 893 671	120 m	65.0%	10.6%
BMS-Webview-1	140	1 078 905	16 m	89.8%	3.6%
Chess	11	58 457	1 s	40.9%	24.4%
Connect	7	1 559 773	88 s	43.4%	17.9%
DNA Amplification	56	82 209	33 s	42.9%	4.6%
Mammals	28	98 515	2 s	81.0%	37.3%
MCADD	12	1 816 628	146 s	63.9%	28.7%
Mushroom	9	169 425	13 s	38.2%	17.5%
Pen Digits	5	333 032	10 s	55.0%	35.2%
Categorical Data					
Independent	50	2 037 016	4 s	100.0%	96.5%
Markov	20	1 932 611	12 s	94.9%	91.6%
Chess	9	57 353	1 s	80.0%	47.7%
Connect	7	1 554 827	11 s	77.2%	33.8%
MCADD	11	1 785 850	6 s	90.8%	82.3%
Mushroom	3	150 012	1 s	56.1%	38.6%
Pen Digits	5	309 788	1 s	82.0%	71.9%

rithm finds, its description length $L(\mathcal{C}, \mathcal{D})$—both absolute and relative to the description length of the independence clustering $L(\mathcal{I}, \mathcal{D})$ and of the canonical description length $L_c(\mathcal{D})$—and the wall clock time the algorithm took to complete. A low number of clusters and a short description length indicate that our algorithm successfully models structure that is present in the data.

For most datasets we see that the number of clusters k is much lower than the number of attributes $|\mathcal{A}|$. In fact, these numbers are such that it is indeed feasible to inspect the clusters by hand. Many of the datasets are highly structured, which can be seen from the strong compression ratios the clusterings achieve with respect to their canonical description lengths. Lastly, the table also shows that our algorithm usually needs just a handful of seconds to discover the clustering.

Below, we investigate the discovered clusterings in closer detail.

For the categorical *Independent* data, we see that the algorithm identifies 50 clusters of single attributes, which correctly corresponds to the generating independence distribution. For the binary *Independent* data, though, the algorithm concludes that there are 49 clusters. Upon further inspection, it turns out that the two attributes that are grouped together are not really independent in the data. In fact, their joint distribution differs significantly from their product distribution (p-value 0.04). This actually makes an interesting point, because MDL does not specifically attempt to discover the 'true' underlying model (which for the synthetic data is available to us), but instead simply tries to describe the data best. For the data instance under consideration here, where by chance two attributes happen to be somewhat correlated, the algorithm cannot but conclude that they should be clustered together.

The attributes in both *Markov* datasets form a Markov chain, so we expect that nearby attributes are clustered together. This is indeed what happens: each cluster consists of consecutive attributes. Figure 4.3 shows the resulting dendrogram for a subset of the attributes in the categorical dataset. Further, if we look at the mutual information between pairs of adjacent attributes, we see that the pairs with high mutual information tend to be grouped together, whereas pairs with low mutual information are not.

Likewise, in the *DAG* dataset, which has attribute dependencies forming a directed acyclic graph, the clusters contain attributes which form tightly linked subgraphs. Figure 4.6 depicts the dependency graph between the attributes of the *DAG* dataset. The width of the edges is proportional to the mutual information between the corresponding attributes. Note that this is done for illustration purposes only, since pairwise mutual information does not capture all information, as pointed out in Section 4.4. The discovered clusters are drawn in rectangles. As the figure shows, the edges inside clusters are generally bold, while the few inter-cluster edges are usually thin. This is a good indication of the quality of the discovered clustering.

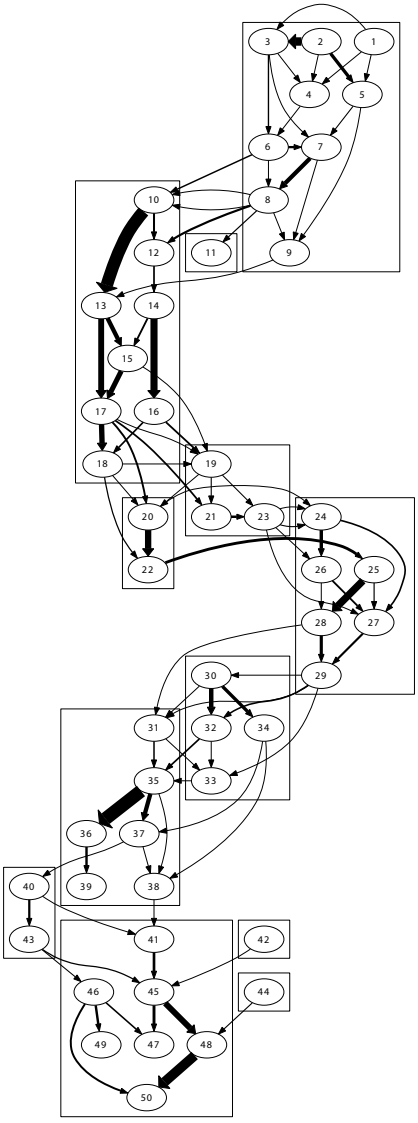


Figure 4.6: Dependency graph of the attributes of the *DAG* dataset, along with the discovered attribute clusters drawn in rectangles. The width of an edge between two nodes is proportional to the mutual information between the corresponding attributes.

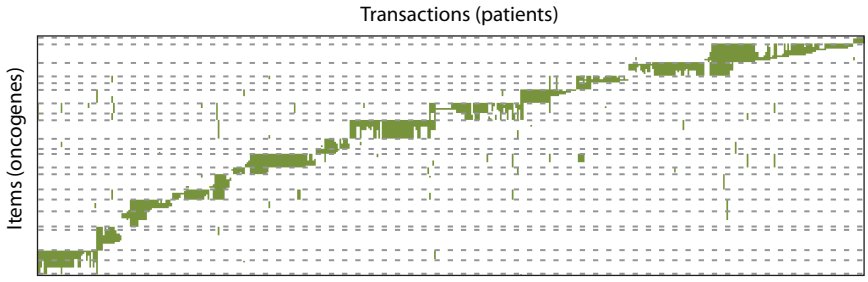


Figure 4.7: A selected sub-matrix of the *DNA Amplification* data and the corresponding discovered attribute clusters, which are separated by the dotted lines. For presentation purposes the figure is rotated sideways

The *DNA Amplification* dataset is an approximately *banded* dataset [Garriga et al., 2008]: the majority of the ones form a staircase pattern, and are located in blocks along the diagonal. In Figure 4.7, a selected subset of the rows and columns of the data is plotted, along with some of the attribute clusters that our algorithm finds. The 1s in the data are drawn dark, the 0s are white. For presentation purposes the figure has been rotated. The clustering clearly distinguishes the blocks that form the staircase pattern, even though the data is quite noisy. These blocks correspond to related oncogenes that are often amplified in conjunction. Further, the clusters contain genes located close to each other on their respective chromosomes. Inspecting the code tables, we see that the attributes within them are strongly correlated: for all code tables, the value with the highest frequency consists of zeroes (well over 90%, note this is a sparse dataset), and is followed by the value consisting of ones, whereas the remaining values have considerably lower frequencies. This shows that the attributes in a single cluster are usually either all present or all absent, far more than expected under independence.

The *Connect* dataset contains all legal grid configurations with eight discs, of the well-known Connect Four game. The game grid has 7 columns and 6 rows, and for each of the 42 locations there is an attribute describing whether it is empty, or which one of the two players has positioned a disc there. Furthermore, a class label describes which player can win or whether the game will result in a draw. Note that as we are mining the data exploratorily,

we consider the class label as 'just another attribute'. For both versions of the data, the algorithm discovers 7 clusters. Upon inspection, it turns out that each of these clusters corresponds to a single column in the game, i.e., the structure found by the algorithm reflects the physical structure of the game, even though our algorithm has no knowledge of this. Whereas different columns are more or less independent, attributes within a column are dependent: an empty location cannot have a nonempty location above it. This is also observed in the code tables: all occurring values correspond to configurations with zero or more filled locations at the bottom, with empty locations above that. Since the full grid contains only eight discs, the highest frequency values in the code tables are the relatively sparse ones. Furthermore, the class label is placed in the cluster of the middle column; this is a very plausible finding, since any horizontal or diagonal row of four must necessarily pass through the middle column, making it key to winning the game. Additionally, we note that in the binary dataset items originating from the same categorical attribute are grouped together, rather than being spread over different clusters.

For both the *Chess* and *Mushroom* datasets, we observe that the discovered clusterings for the categorical and binary datasets are not exactly the same. The algorithm makes a few more merges in the categorical case, but otherwise the discovered clusterings are comparable. For these datasets we also see in the binary variants that items originating from the same attribute in general tend to be clustered together. Interestingly, the (absolute) description lengths are very similar for both variants of the datasets, being slightly smaller in the categorical case. The explanation lies in the fact that for the categorical data we already know which values a given attribute may assume, while for the binary versions we do not know which items belong to the same attributes, which indirectly increases the cost to describe the code tables.

The attributes of the *MCADD* dataset consist of 12 different acylcarnitine concentrations measured by tandem mass spectrometry (TMS), together with 4 of their calculated ratios and 5 other biochemical parameters, and the class label which indicates whether MCADD is present. This recessive metabolic disease affects about one in 10 000 people while around one in 65 is a carrier of the responsible mutated gene. If left undiagnosed, this rare disease is fatal in 20% to 25% of the cases and many survivors are left with severe brain damage after a severe crisis. In the results, we see that acylcarnitines and corresponding calculated ratios are detected and grouped together. For

instance, the attributes for $C8$ and $C10$ are in a cluster together with the ratio $\frac{C8}{C10}$. Further, the class label cluster also contains the ratio $\frac{C8}{C12}$, which is one of the features commonly used in diagnostic criteria by experts and was also discovered in previous in-depth studies [Baumgartner et al., 2005].

Finally, for the *Pen Digits* datasets, we discover five attribute clusters. Each cluster consists entirely of either x or y coordinates. Besides spatially, the coordinate attributes are also temporally clustered: coordinates appearing first along the trace of a digit are grouped together, as are coordinates drawn later. More specifically, the x coordinates are split in three groups: beginning, middle, and end. The y coordinates are split in a beginning, and a middle–end cluster. This last cluster also contains the class label, which means that the class label is correlated to the vertical coordinates of the pen when drawing the latter half of a digit, a statement the authors deem plausible. The binary variant of this datasets results in a very similar summary, i.e., barring a few exceptions, items originating from the same attributes are grouped together in the same way as with the categorical data.

The above examples show that the attribute clusterings discovered by our algorithm are of high quality. We find structure between correlated attributes, which can be seen from the strong compression ratios and relatively small number of clusters. When subjectively investigating the resulting clusters themselves, they are easily interpretable and highly insightful.

Having established that we can achieve good results, we investigate what the contribution of our encoding is. We compare our defined description length $L(\mathcal{C}, \mathcal{D})$ with the description length $L_r(\mathcal{C}, \mathcal{D})$, which uses prequential coding (see Section 4.2). Table 4.3 shows the results for the discovered clusterings using $L_r(\mathcal{C}, \mathcal{D})$ as the description length. We note that the numbers of clusters discovered are comparable, usually slightly higher. Prequential encoding tends to be a bit more conservative in merging clusters, whereas our encoding tends to prefer clusterings with a lower number of clusters; for our purpose, this is a positive thing. Furthermore, the description lengths for both encodings are in the same ballpark.

It is important to emphasize, however, that we cannot simply compare the description lengths of two different clusterings under two different encodings. Nonetheless, the fact that the returned best clusterings for both encodings tend to be alike, and have a similar description length, is a good indication that our encoding is close to the theoretically clean refined MDL. Upon inspection, the discovered clusters are often comparable to the ones

Table 4.3: Results of our algorithm using the description length $L_r(\mathcal{C}, \mathcal{D})$, which uses prequential coding. Shown are the number of discovered clusters k, and the absolute and relative description length with respect to the description length of the independence and canonical clusterings.

Binary Data	k	$L_r(\mathcal{C}, \mathcal{D})$	$\frac{L_r(\mathcal{C},\mathcal{D})}{L_r(\mathcal{I},\mathcal{D})}$	$\frac{L_r(\mathcal{C},\mathcal{D})}{L_c(\mathcal{D})}$
Independent	48	894 855	99.9%	89.5%
Markov	11	880 567	88.1%	88.1%
DAG	8	774 276	82.5%	77.4%
Accidents	171	17 253 821	66.4%	10.8%
BMS-Webview-1	97	1 045 076	87.3%	3.5%
Chess	15	60 892	42.7%	25.4%
Connect	15	1 632 662	45.4%	18.7%
DNA Amplification	76	77 598	41.2%	4.3%
Mammals	29	92 990	76.9%	35.2%
MCADD	23	2 050 896	72.1%	32.4%
Mushroom	18	248 494	56.1%	25.7%
Pen Digits	12	379 027	62.6%	40.1%
Categorical Data				
Independent	50	2 036 444	100.0%	96.5%
Markov	15	1 908 282	93.7%	90.4%
Chess	7	56 107	78.5%	46.7%
Connect	7	1 552 621	77.1%	33.7%
MCADD	7	1 756 006	89.3%	81.0%
Mushroom	6	188 540	70.7%	48.6%
Pen Digits	4	303 282	80.3%	70.4%

discovered using our $L(\mathcal{C}, \mathcal{D})$ description length, however, for many datasets the clusters discovered using $L_r(\mathcal{C}, \mathcal{D})$ tend to be of a slightly lower subjective quality. For instance, for the binary *Connect* data we do not obtain the seven column clusters that were obtained before. Consequently, since the results using our encoding are comparable or better than those using prequential coding, and additionally since our encoding is more intuitive, from this point on we will only be using our description length $L(\mathcal{C}, \mathcal{D})$.

Randomization

Next, we investigate whether our algorithm is generally capable of discovering structure beyond what can be straightforwardly explained by simpler statistics. As such, for the binary datasets, we here consider the row and column margins, i.e., the number of ones per row and column. The idea is that if for some data the discovered attribute clustering simply follows from the margins of the data, then we would obtain the same result on any other dataset with the same margins. By considering random data with the same margins as the real datasets, we can so check whether our models are indeed able to model structure of the data at hand beyond what follows from its margins (assuming these datasets contain more structure than that). For random data we expect our approach to return the independence clustering.

To obtain the random data samples needed for these experiments, we use swap randomization, which is the process of randomizing data to obscure the internal dependencies, while preserving the row and column margins of the data [Gionis et al., 2007]. This is achieved by applying individual swap operations that maintain these margins. That is, one randomly finds two items a and b, and two transactions, such that a occurs in the first transaction but not in the second, and vice versa. Then, a swap operation simply swaps these items. This is a Markov Chain Monte Carlo process, which has to be repeated many times, as often as is required to break down the significant structure of the data, i.e., the mixing time of the chain; as suggested by Gionis et al. [2007], we use five times the number of ones in the data.

For each dataset we create 1 000 swap randomized datasets. We then calculate the average number of clusters our algorithm finds, and the average description length. Table 4.4 shows the results. We see that for all datasets the number of clusters is very close to the number of attributes, indicating that the structure that was present in the original datasets, was not simply a consequence of the row and column margins. Furthermore, the description lengths are much higher than those for the clusterings discovered in the original data. In fact, the third to last column shows that the average description length is almost exactly equal to the description length of the independence clustering. The last column shows the *empirical* p-value of our results,

$$p = \frac{|\{\mathcal{D}' \mid L(\mathcal{C}', \mathcal{D}') \leq L(\mathcal{C}, \mathcal{D})\}| + 1}{|\{\mathcal{D}'\}| + 1} ,$$

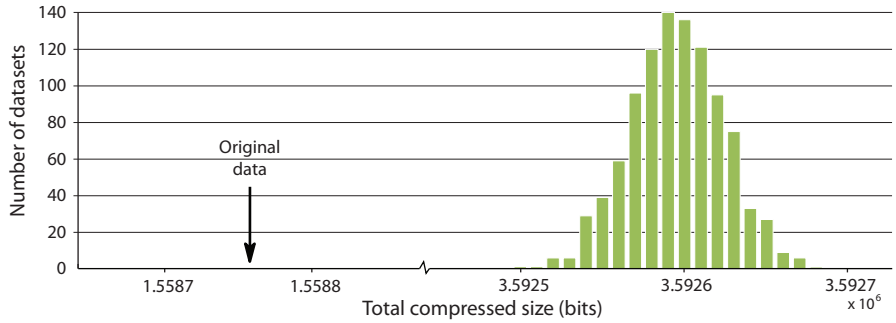


Figure 4.8: Distribution of the description lengths of 1 000 swap randomized instances of the binary version of the *Connect* dataset. The description length of the original data is indicated by the arrow.

where $\{\mathcal{D}'\}$ is the set of 1 000 swap randomized datasets. This is the empirical probability of observing a description length at least as low as $L(\mathcal{C}, \mathcal{D})$, Figure 4.8 shows the description length distribution of the swap randomized versions of the *Connect* dataset. We see that the description length of the original dataset is significantly lower than that of any of the swap randomized datasets. The p-values show that this holds for all the binary datasets, except for *Independent*; the structure of this dataset can of course be completely described by the column margins only.

Our algorithm is greedy in nature, and therefore does not necessarily discover the optimal clustering, that is, the clustering $\mathcal{C}^*$ which globally minimizes $L(\mathcal{C}, \mathcal{D})$. To get an idea of how good the discovered clustering is, we compare it to random clusterings. We uniformly sampled 1 000 random k-attribute clusterings for each dataset (where k is the size of the discovered clustering), and compute their description lengths. Table 4.5 shows for each dataset the absolute and relative average description length and standard deviation. For most datasets the relative description length is still less than 100%, indicating that even random clusterings can capture *some* of the structure, especially for strongly structured datasets. However, we also see that for all datasets (except trivially for the categorical *Independent* data) the description length for random clusterings is much worse than that of the original discovered clustering (see Table 4.2). From the empirical p-values in the last column of Table 4.5, we can see that our algorithm significantly outperforms

Table 4.4: Swap randomization experiments on the binary data, for 1 000 swap randomized datasets. Shown are the average number of clusters k, the average absolute description length $L(\mathcal{C}', \mathcal{D}')$, the description length $L(\mathcal{C}, \mathcal{D})$ of the original data, the average relative description length with respect to both the independence clustering description length $L(\mathcal{I}, \mathcal{D}')$ and the canonical description length $L_c(\mathcal{D}')$, and the empirical p-value for the original data.

Binary Data	k	$L(\mathcal{C}', \mathcal{D}')$	$L(\mathcal{C}, \mathcal{D})$	$\frac{L(\mathcal{C}', \mathcal{D}')}{L(\mathcal{I}, \mathcal{D}')}$	$\frac{L(\mathcal{C}', \mathcal{D}')}{L_c(\mathcal{D}')}$	*empirical p-value*
Independent	49.2	895 078	895 078	99.9%	89.5%	51.5%
Markov	48.9	999 157	884 943	99.9%	99.9%	$< 0.1\%$
DAG	48.9	970 481	797 588	99.9%	97.0%	$< 0.1\%$
Accidents	198.2	25 987 733	16 893 671	99.9%	16.3%	$< 0.1\%$
BMS-Webview-1	221.1	1 195 928	1 078 905	99.5%	4.0%	$< 0.1\%$
Chess	58.8	142 748	58 457	99.9%	59.5%	$< 0.1\%$
Connect	81.0	3 592 592	1 559 773	99.9%	41.2%	$< 0.1\%$
DNA Amplification	194.1	190 422	82 209	99.5%	10.6%	$< 0.1\%$
Mammals	62.3	121 048	98 515	99.5%	45.8%	$< 0.1\%$
MCADD	163.7	2 844 348	1 816 628	99.9%	45.0%	$< 0.1\%$
Mushroom	77.5	443 042	169 425	99.9%	45.8%	$< 0.1\%$
Pen Digits	64.6	605 330	333 032	99.9%	64.0%	$< 0.1\%$

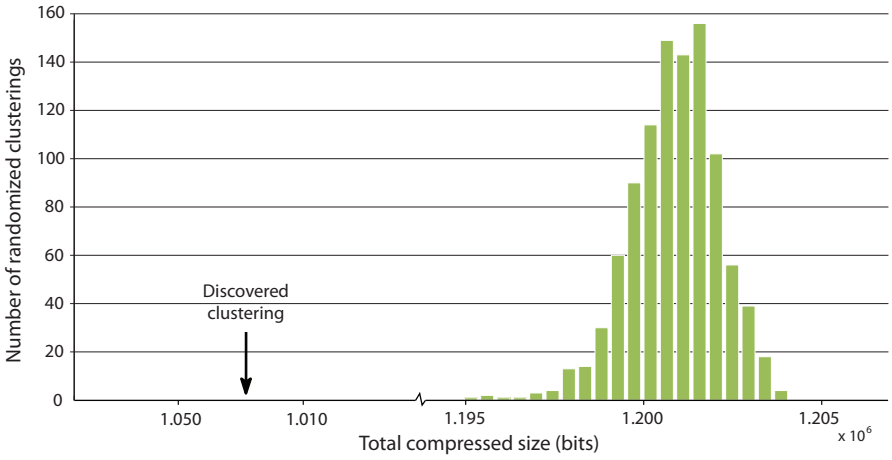


Figure 4.9: Distribution of the description lengths of 1 000 random k-clusterings, for the *BMS-Webview-1* dataset. The description length of the clustering discovered by our algorithm is indicated by the arrow.

randomly generated clusterings. In fact, none of the random clusterings had a lower description length than the one discovered by our algorithm (again, except for *Independent*). Figure 4.9 depicts the description length distribution for the *BMS-Webview-1* dataset, of the 1 000 randomly generated random attribute clusterings, together with the description length of the clustering found by our algorithm.

Beam Search and Simulated Annealing

In this subsection we investigate whether we can improve our algorithm by employing different search strategies. We ran experiments using beam search for a range of beam widths. Table 4.6 shows the results for b equal to 2, 5, 10, and 15. Note that using a beam width of 1 corresponds to our original algorithm. For most datasets and parameter settings, the discovered summaries are exactly the same as for a beam with of 1. In the other cases, some summaries with a lower description length were discovered, however, this relative decrease is barely noticeable (i.e., 10^{-3} or less). Meanwhile, it is clear that both runtime and memory consumption grow as the beam size is in-

Table 4.5: Average description length over 1 000 randomly generated k-partitions, relative to the canonical description length of the datasets. Shown are the average absolute description length $L(\mathcal{C}', \mathcal{D})$, the description length $L(\mathcal{C}, \mathcal{D})$ of the original discovered clustering, the relative description length with standard deviation, and the empirical p-value of the original discovered clustering.

Binary Data	$L(\mathcal{C}', \mathcal{D})$	$L(\mathcal{C}, \mathcal{D})$	$\frac{L(\mathcal{C}', \mathcal{D})}{L_c(\mathcal{D})}$	*standard deviation*	*empirical p-value*
Independent	930 878	895 078	93.1%	0.3%	$< 0.1\%$
Markov	994 857	884 943	99.5%	0.7%	$< 0.1\%$
DAG	976 662	797 588	93.2%	1.1%	$< 0.1\%$
Accidents	25 830 095	16 893 671	16.2%	0.1%	$< 0.1\%$
BMS-Webview-1	1 200 869	1 078 905	4.1%	0.0%	$< 0.1\%$
Chess	135 785	58 457	56.6%	1.9%	$< 0.1\%$
Connect	3 274 147	1 559 773	37.6%	0.9%	$< 0.1\%$
DNA Amplification	199 770	82 209	11.1%	0.1%	$< 0.1\%$
Mammals	117 597	98 515	44.5%	0.3%	$< 0.1\%$
MCADD	3 575 766	1 816 628	56.6%	1.9%	$< 0.1\%$
Mushroom	341 035	169 425	35.3%	0.7%	$< 0.1\%$
Pen Digits	654 217	333 032	69.2%	2.4%	$< 0.1\%$
Categorical Data					
Independent	2 037 016	2 037 016	96.5%	0.0%	100.0%
Markov	2 240 147	1 932 611	115.9%	9.2%	$< 0.1\%$
Chess	70 424	57 353	59.6%	1.0%	$< 0.1\%$
Connect	1 916 691	1 554 827	41.6%	0.6%	$< 0.1\%$
MCADD	2 169 650	1 785 850	100.0%	10.9%	$< 0.1\%$
Mushroom	179 259	150 012	46.2%	3.7%	$< 0.1\%$
Pen Digits	399 095	309 788	92.7%	7.8%	$< 0.1\%$

Table 4.6: Results for the beam search experiments for various beam widths. The table gives the description length $L(\mathcal{C}, \mathcal{D})$ of the best discovered clustering, relative to the canonical description length $L_c(\mathcal{D})$. The first column repeats the results from Table 4.2, and is equivalent to the case $b = 1$. The model that compresses best is depicted in boldface at the smallest beam width for that score.

Binary Data	$\frac{L(\mathcal{C},\mathcal{D})}{L_c(\mathcal{D})}$	$b = 2$	$b = 5$	$b = 10$	$b = 15$
Independent	**89.5%**	89.5%	89.5%	89.5%	89.5%
Markov	**88.5%**	88.5%	88.5%	88.5%	88.5%
DAG	79.8%	**79.7%**	79.7%	79.7%	79.7%
Accidents	**10.6%**	10.6%	10.6%	10.6%	10.6%
BMS-Webview-1	**3.6%**	3.6%	3.6%	3.6%	3.6%
Chess	**24.4%**	24.4%	24.4%	24.4%	24.4%
Connect	**17.9%**	17.9%	17.9%	17.9%	17.9%
DNA Amplification	**4.6%**	4.6%	4.6%	4.6%	4.6%
Mammals	**37.3%**	37.3%	37.3%	37.3%	37.3%
MCADD	**28.7%**	28.7%	28.7%	28.7%	28.7%
Mushroom	17.5%	**17.4%**	17.4%	17.4%	17.4%
Pen Digits	**35.2%**	35.2%	35.2%	35.2%	35.2%
Categorical Data					
Independent	**96.5%**	96.5%	96.5%	96.5%	96.5%
Markov	91.6%	**91.4%**	91.4%	91.4%	91.4%
Chess	**47.7%**	47.7%	47.7%	47.7%	47.7%
Connect	**33.8%**	33.8%	33.8%	33.8%	33.8%
MCADD	82.3%	82.3%	**82.0%**	82.0%	82.0%
Mushroom	38.6%	38.6%	**38.3%**	38.3%	38.3%
Pen Digits	**71.9%**	71.9%	71.9%	71.9%	71.9%

creased, namely by a factor b. Therefore, it does not seem to be favorable to add a beam to our algorithm, since neither the results nor the performance can notably be improved upon. The reason for this is that although beam search considers more search paths, the beam tends to exhibit very little variability and hence usually ends up with the same result.

Table 4.7 gives the results of similar experiments using simulated anneal-

ing. For each dataset, we execute 100 runs, where a single run consists of 1 000 iterations. The best clustering over those 100 runs is then reported. We repeat the experiments for varying settings of the λ scaling parameter. For the synthetic datasets, we see that simulated annealing performs approximately the same as our algorithm, i.e., the compression ratios are comparable. For most of the other datasets, however, the results are noticeably worse than our algorithm. Only for *MCADD* does simulated annealing provide a marginally better result. The influence of the λ parameter seems to be that the compression ratios decrease slightly for larger λ, although not much. This is because the λ parameter controls the decrease of the probability threshold to move away from a local optimum. Due to the inherently nondeterministic nature of the simulated annealing algorithm, we must perform many runs, but we do not know how many in advance. Furthermore, in practice the simulated annealing algorithm has many more parameters than we used in our experiments, e.g., the cooling schedule, and tuning these parameters can prove to be difficult. Hence, simulated annealing does not seem to provide any improvement over our algorithm.

Frequency Estimation

Finally, in this subsection we investigate how well our summaries can be used as surrogates of the data. We do this by using the code tables to estimate itemset frequencies, as described in Section 4.2. For each dataset we first mine the top-10 000 closed frequent itemsets.[3] Then, for each itemset in this collection, we estimate its frequency using the discovered attribute clustering, and compute both its absolute and relative error. For comparison, we generate 1 000 random k-clusterings, estimate their frequencies, and average the mean absolute and relative errors. The results are shown in Table 4.8.

Although frequency estimation is not the main goal of our approach, the results we obtain are very good. For most datasets, the average absolute error is less than 1%. Furthermore, the average relative error is usually also just a few percentage points. For the datasets where the relative error is larger, we see that the cause for this lies with the fact that the itemsets in those datasets have a very low average frequency. Compared to the random k-

[3]More precisely, we use the largest minimum support threshold such that the number of frequent closed itemsets is at least 10 000; therefore the total number of itemsets may be slightly larger. For the *DNA Amplification* data, there are only 1 946 closed itemsets with nonzero support.

Table 4.7: Results for the simulated annealing experiments for various settings of the λ parameter. The table gives the description length $L(\mathcal{C}, \mathcal{D})$ of the best discovered clustering, relative to the canonical description length $L_c(\mathcal{D})$. The first column repeats the results from Table 4.2. The best performing model is depicted in boldface.

Binary Data	$\frac{L(\mathcal{C},\mathcal{D})}{L_c(\mathcal{D})}$	$\lambda = 10^0$	$\lambda = 10^1$	$\lambda = 10^2$	$\lambda = 10^3$
Independent	**89.5%**	89.5%	89.5%	89.5%	89.5%
Markov	**88.5%**	88.6%	88.6%	88.6%	88.5%
DAG	**79.8%**	80.3%	80.5%	80.0%	80.3%
Accidents	**10.6%**	15.8%	15.8%	15.6%	15.7%
BMS-Webview-1	**3.6%**	4.0%	4.0%	4.0%	4.0%
Chess	**24.4%**	30.9%	28.8%	30.9%	28.1%
Connect	**17.9%**	28.2%	26.3%	28.0%	26.9%
DNA Amplification	**4.6%**	10.0%	10.0%	10.1%	10.0%
Mammals	**37.3%**	40.4%	40.4%	40.6%	40.2%
MCADD	**28.7%**	43.2%	43.1%	43.3%	43.1%
Mushroom	**17.5%**	25.8%	24.0%	24.6%	24.7%
Pen Digits	**35.2%**	47.1%	48.2%	48.7%	48.1%
Categorical Data					
Independent	**96.5%**	96.5%	96.5%	96.5%	96.5%
Markov	**91.6%**	92.4%	92.0%	92.2%	91.8%
Chess	**47.7%**	48.0%	48.0%	48.1%	48.0%
Connect	**33.8%**	34.2%	34.3%	34.2%	34.2%
MCADD	82.3%	**82.1%**	82.1%	82.1%	82.1%
Mushroom	**38.6%**	39.7%	39.5%	39.7%	39.7%
Pen Digits	**71.9%**	73.8%	72.3%	72.5%	72.1%

clusterings, we see that our algorithm always performs better on average, and this difference is significant, as can be seen from the empirical p-values in the last column. Further, as the code tables corresponding to a k-clustering (with $k < n$) inherently contain more information on how the attributes interact than for the independence clustering—which is an n-clustering, and hence by definition contains no correlations between attributes—our algorithm also performs better than the independence model in estimating count queries.

Table 4.8: Frequency estimation of the top-10 000 closed frequent itemsets. Depicted are the average frequency $\overline{fr}$ of the itemsets in the original data, the average absolute and relative errors of the frequency estimates using our model, the average absolute and relative errors for 1 000 random k-partitions, and the empirical p-values of the result of our algorithm for the relative estimation error

		Attribute Clustering		*Random k-partition*		
Binary Data	$\overline{fr}$	$\lvert fr - \hat{fr}\rvert$	$\frac{\lvert fr-\hat{fr}\rvert}{fr}$	$\lvert fr - \hat{fr}\rvert$	$\frac{\lvert fr-\hat{fr}\rvert}{fr}$	*empirical p-value*
Independent	29.0%	0.1%	0.5%	1.3%	4.5%	$< 0.1\%$
Markov	15.7%	0.3%	2.0%	1.3%	7.9%	$< 0.1\%$
DAG	25.5%	0.6%	2.4%	2.1%	8.7%	$< 0.1\%$
Accidents	55.8%	1.4%	2.7%	2.9%	5.3%	$< 0.1\%$
BMS-Webview-1	0.1%	0.1%	83.8%	0.1%	91.2%	$< 0.1\%$
Chess	81.2%	1.0%	1.2%	1.4%	1.7%	0.2%
Connect	88.8%	0.4%	0.4%	2.2%	2.5%	$< 0.1\%$
DNA Amplification	0.7%	0.1%	53.3%	4.2%	86.6%	$< 0.1\%$
Mammals	43.0%	13.3%	31.7%	19.8%	46.8%	$< 0.1\%$
MCADD	2.6%	0.2%	8.7%	0.3%	13.1%	$< 0.1\%$
Mushroom	12.5%	1.3%	13.6%	5.1%	44.9%	$< 0.1\%$
Pen Digits	6.1%	2.9%	52.1%	3.3%	58.5%	$< 0.1\%$
Categorical Data						
Independent	10.2%	0.1%	1.3%	0.1%	1.3%	100.0%
Markov	11.7%	0.3%	2.7%	0.6%	5.1%	$< 0.1\%$
Chess	81.2%	0.5%	0.6%	1.3%	1.6%	$< 0.1\%$
Connect	88.8%	0.4%	0.4%	2.2%	2.5%	$< 0.1\%$
MCADD	2.6%	0.2%	8.5%	0.3%	13.1%	$< 0.1\%$
Mushroom	12.5%	2.8%	2.5%	38.8%	380.7%	$< 0.1\%$
Pen Digits	6.1%	3.0%	53.8%	3.3%	58.9%	1.3%

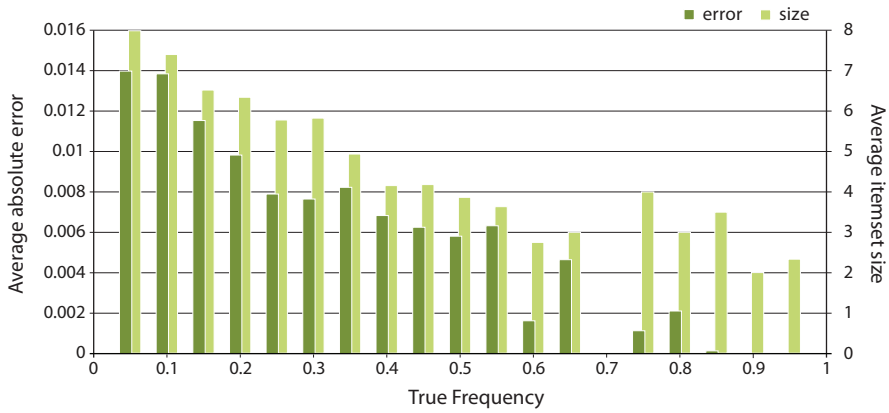


Figure 4.10: The average estimation error per frequency, and the average itemset size per frequency for the *Mushroom* dataset.

Figure 4.10 shows the average estimation error in function of the frequency of the top-10 000 closed itemsets for the *Mushroom* dataset, in bins with a width of 5%. We see that the error tends to grow as the frequency decreases; the reason for this lies in the fact that low-frequency itemsets tend to be larger (as also depicted in the figure), and hence we have to combine information from more code tables, which makes the estimate less precise, since in doing so we make more independence assumptions.

In Figure 4.11 we plot the cumulative probability of the absolute estimation error for the *Connect* and *Mushroom* datasets. For every $\epsilon \in [0,1]$ we determine the probability δ that the absolute error $|\, fr(X) - \hat{fr}(X) \,|$ is greater than ϵ. For both datasets we see that the discovered clustering outperforms the random k-clusterings, which in turn only marginally improve upon the independence model. For instance, in the *Mushroom* dataset we see that probability of an absolute estimation error larger than 5% is about 40% for the random models, whereas for our discovered clustering this is only 1%.

In Table 4.9 the speed of querying code tables is demonstrated. For each dataset, we generate 100 000 itemsets uniformly, with sizes uniformly picked between 2 and the size of the largest transaction. First, we measure the query speed of a straightforward counting algorithm, which iterates over all transactions, and increases a counter whenever the queried itemset is contained in

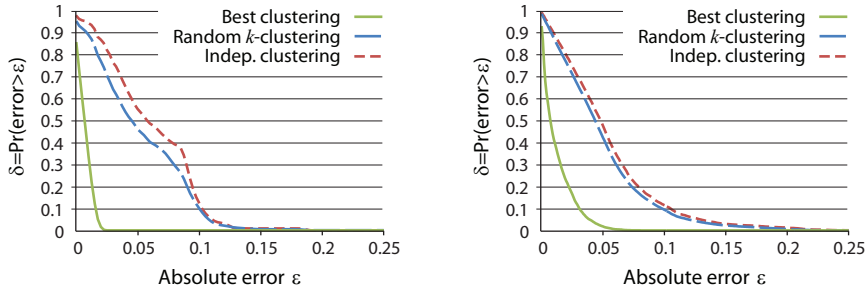


Figure 4.11: Probability of a frequency estimation error larger than ϵ for *Connect* (left) and *Mushroom* (right) on the top-10 000 closed frequent itemsets.

a transaction. Second, we measure the speed of querying by using code tables rather than the data itself. Both methods were implemented in C++, compiled with the same compilation flags, and executed on the same machine. The first and third column show the average query time in milliseconds per itemset. For all datasets, using approximate querying is considerably faster, often an order of magnitude or more. Constructing the code tables, however, also takes time, and hence there is a point up to which querying the data is still cheaper. The rightmost column shows the number of queries such that the time taken to construct the model plus the time to perform the queries, is equal to querying the data (measured in a resolution of ten itemsets). For most datasets, the takeover point lies at a few thousand queries.

Lastly, we compare the frequency estimation capabilities of our attribute clusterings with the profile-based summarization approach by Yan et al. [2005]. A set of (overlapping) profiles summarizes a given set of patterns—rather than the data itself. A profile can be seen as a conditional independence distribution on a subset of the items, which can subsequently be queried. Although summarization with profiles is different from our clustering approach, we can compare the quality of the frequency estimates. We mimic the experiments by Yan et al. [2005] on the *Mushroom* and *BMS-Webview-1* datasets, by comparing the average relative error, also called *restoration error* in the paper. The collection of itemsets contains all frequent closed itemsets for a minsup threshold of 25% and 0.1% respectively. For *Mushroom* we attain a restoration error of 2.31%, which is lower than the results reported by Yan et al. [2005] for any number of profiles. For *BMS-Webview-1*

Table 4.9: The speed of direct querying versus frequency estimation, averaged over 100 000 randomly sampled itemsets, and the point at which the aggregate model construction and approximate querying time overtakes exact querying. Values in italics are extrapolated.

		Summary		
Binary Data	query	construction	estimation	takeover
Independent	2.37 ms	1 s	0.02 ms	740
Markov	2.27 ms	4 s	0.04 ms	1 400
DAG	2.31 ms	5 s	0.11 ms	2 450
Accidents	45.18 ms	120 m	2.07 ms	*166 968*
BMS-Webview-1	1.91 ms	13 m	0.29 ms	*596 913*
Chess	0.32 ms	1 s	0.03 ms	1 790
Connect	7.52 ms	88 s	0.06 ms	15 220
DNA Amplification	0.24 ms	33 s	0.16 ms	*412 500*
Mammals	0.20 ms	2 s	0.05 ms	5 390
MCADD	2.49 ms	146 s	0.08 ms	60 380
Mushroom	0.50 ms	13 s	0.05 ms	28 140
Pen Digits	0.58 ms	10 s	0.10 ms	31 660
Categorical Data				
Independent	0.53 ms	4 s	0.05 ms	6 170
Markov	0.53 ms	12 s	0.11 ms	28 510
Chess	0.09 ms	1 s	0.04 ms	5 380
Connect	2.52 ms	11 s	0.07 ms	5 810
MCADD	0.59 ms	6 s	0.06 ms	9 030
Mushroom	0.17 ms	1 s	0.05 ms	2 850
Pen Digits	0.21 ms	1 s	0.07 ms	3 270

our restoration error is 70.4%, which is on par with Yan et al.'s results when using about 100 profiles. Their results improve when increasing the number of profiles. However, the best scores require over a thousand profiles, arguably not a feasible number of profiles to inspect by hand.

4.7 Discussion

The experiments show that our method discovers high-quality attribute clusterings. The description lengths of the discovered summaries are noticeably low compared to the canonical description lengths, which indicates that our algorithm successfully exploits the structure of the data. Moreover, inspection of the returned clusterings shows that on synthetic data, the discovered attribute clusterings are in accordance with their respective generative processes, while on real data correlated attributes are correctly grouped, and the discovered clusterings are intuitive and logical. Through inspection of the code tables of clusters, we showed that the user is given clear insight in the distribution of the data over the grouped attributes, and can easily identify the prevalent attribute-value combinations.

The comparison between encodings shows that our basic two-part MDL encoding provides performance similar to or even slightly better than the theoretically more refined encoding, which tends to be more conservative. Additionally, our basic encoding has the advantage of being more interpretable. While our algorithm is greedy, and therefore does not necessarily produce an optimal result, experiments showed that the discovered summaries obtain significantly lower description lengths than for random clusterings. The swap randomization experiments validate that our approach identifies structure that can generally not be explained through simple statistics, i.e., the row and column margins of the data. As such, we meet our goal that our summaries provide basic insight in the data that goes beyond first order statistics.

The experiments on alternative search strategies show that beam search and simulated annealing do not provide any noticeable improvement over our algorithm, which indicates that our greedy search indeed exploits the structure of the search space well. Finally, we demonstrated that besides the fact that attribute clusterings are a simple and intuitive way to gain insight into a dataset, they can also be used as queryable surrogates for the data; itemset frequencies of the top-10 000 closed frequent itemsets are approximated quickly and with very high precision.

As such, we have shown that the summaries we discover provide useful insight into the data beyond simple first order statistics, and can hence be used as a quick first inspection of the basic structure of the data, for instance in order to decide whether and how the data should be analyzed in detail. Moreover, as they are simply groupings of attributes, and counts of the value

combinations thereof, they can be straightforwardly included explicitly as background knowledge in other mining algorithms, in order to prevent discovery of results that can be explained by what we already know [Hanhijärvi et al., 2009, Mampaey et al., 2011].

While much of the focus in data mining research is on detailed data analysis, we point out the importance of lightweight methods providing basic insight in data, to the end of making an informed decision on whether and how to mine the data at hand. Although such methods clearly should be fast, it is important to stress that the ease of interpreting their results is key.

Besides basic insight, other possible uses for our summaries include feature selection and feature construction. While not the aim of this work, and hence we do not go into detail, one could consider only clustering attributes with the target label, in order to identify by MDL the set of features that together best describe the target. Alternatively, one could consider each cluster as a new feature, choosing either all, or only the most prevalent, value combinations as its values. A related interesting future use for our summaries would be fast approximate frequent itemset mining, as the possible combinations of attribute-values can be effectively enumerated by a-priori.

Although the experiments show that high-quality summaries are discovered, by the greedy nature of our approach we have no guarantees on how well we approximate the optimum. While this is an often-encountered problem in MDL, it may be worthwhile to investigate whether a connection can be made to well-studied optimization problems, such as SETCOVER, for which optimality bounds are known.

By using MDL to discover a summary, our method is parameter-free: the amount and structure in the data determines what model is chosen. In general, MDL can be regarded as data hungry. That is, the more data is available, the better more complex correlations can be detected. In general, our method is best applied to data of at least 100s of rows.

Even though the experiments show that our algorithm is fast in practice, we see room for improvements. For instance, the algorithm can trivially be parallelized, as well as optimized by using *tid*-lists. The main bottleneck of our approach is the first step of the clustering, where all pair-wise distances have to be computed; it would be worthwhile to develop bounds or heuristics to postpone calculation of distances between attributes that will not be considered for joining. However, we especially regard the development of fast *approximate* summarization techniques for databases with many transac-

tions and/or attributes as an important topic for future research, in particular as many data mining techniques cannot consider such datasets directly, but could be made to consider the summary surrogate.

In this work we only consider categorical data, for which it is difficult to construct a sensible distance measure. As such, it would be interesting to investigate whether our approach can be extended toward ordinal data, which would require non-trivial extension of the encoding. Another, related, as well as important, open problem is the generation of summaries for heterogeneous data, e.g., consisting of both numeric and categorical attributes.

4.8 Conclusions

In this chapter we introduced a method for getting a good first impression of a dataset with categorical attributes. Our parameter-free method builds summaries by clustering attributes that strongly correlate, and uses the Minimum Description Length principle to identify the best clustering, without requiring a distance measure between attributes. The result offers an overview of which attributes interact most strongly, and in what value instantiations they typically occur. Furthermore, since they form probabilistic models of the data, these summaries are good surrogates for the data that can be queried efficiently and accurately. Experiments showed that our method provides high-quality results that correctly identify groups of correlated attributes, and can be used to obtain close approximations of itemset frequencies.

Chapter 5

Using Background Knowledge to Rank Itemsets

Assessing the quality of discovered results is an important open problem in data mining. Such assessment is particularly vital when mining itemsets, since commonly many of the discovered patterns can be easily explained by background knowledge. The simplest approach to screen uninteresting patterns is to compare the observed frequency against the independence model. Since the parameters for the independence model are the column margins, we can view such screening as a way of using the column margins as background knowledge.

In this chapter we study techniques for more flexible approaches to infusing background knowledge. Namely, we show that we can efficiently use additional easy-to-interpret statistics such as row margins, lazarus counts, and transaction bounds. We demonstrate that these statistics describe forms of data that occur in practice and have been studied in data mining.

To infuse the information efficiently we use a maximum entropy approach. In general, solving a maximum entropy model is infeasible, but we demonstrate that for our setting it can be solved in polynomial time. Experiments show that more sophisticated models fit the data better and that using more information improves the frequency prediction of itemsets.

This chapter is based on work published as:
N. Tatti and M. Mampaey. Using background knowledge to rank itemsets. *Data Mining and Knowledge Discovery*, 21(2):293–309, 2010.

5.1 Introduction

Discovering interesting itemsets from binary data is one of the most studied branches in pattern mining. The most common way of defining the interestingness of an itemset is by its frequency, the fraction of transactions of the data in which all items co-occur. This measure has a significant computational advantage since frequent itemsets can be discovered using level-wise or depth-first search strategies (see Section 2.1). These itemsets can then be presented to the user, in a ranking starting from the most interesting ones. The drawback of frequency is that we cannot infuse any background knowledge into the ranking. For example, if we know that the items a and b occur often, then we should expect that ab also occurs relatively often.

Many approaches have been suggested to infuse background knowledge into ranking itemsets. The most common approach is to compare the observed frequency against the independence model. This approach has the advantage that we can easily compute the estimate and that the background knowledge is easily understandable. The downside of the independence model is that it contains relatively little information. For example, if we know that most of the data points contain a small number of ones, then we should infuse this information for ranking patterns. For a more detailed discussion see the related work in Section 5.8.

Assessing the quality of patterns can be seen as a part of the general idea where we are required to test whether a data mining result is statistically significant with respect to some background knowledge (see [Hanhijärvi et al., 2009] as an example of such a framework). However, the need for such assessment is especially important in pattern mining due to two major problems. Firstly, the number of discovered patterns is usually extremely large, so a screening/ranking process is needed. Secondly, many of the discovered patterns reflect already known information, so we need to incorporate this information such that we can remove trivial results.

The aim of this chapter is to study how we can infuse background knowledge into pattern mining efficiently. In our approach we will build a global statistical model based on the given knowledge. We set the following goals.

1. The background knowledge should be simple to understand.
2. We should be able to infer the model from the data efficiently.
3. We should be able to efficiently compute expected itemset frequencies.

While these goals seem simple, they are in fact quite strict. For example, consider modeling attribute dependencies by using Bayesian networks. First of all, inferring the expected frequency of an itemset from the Bayesian network is done using a message passing algorithm, and is not guaranteed to be computationally feasible [Cowell et al., 1999]. Further, understanding parent-child relationships in a Bayesian network can be discombobulating.

We will consider the following simple statistics: column margins, row margins, number of zeroes between ones (lazarus counts), and the boundaries of ones. We will use these statistics individually, but also consider different combinations of them. While these are simple statistics, we will show that they describe many specific types of dataset structures, such as banded datasets or nested datasets.

We will use these statistics and the maximum entropy principle to build a global model. In a general setting, inferring such a model is an infeasible problem. However, we will demonstrate that for our statistics, inferring the model can be done in polynomial time. Once this model is discovered we can use it to assess the quality of an itemset by comparing the observed frequency against the frequency expected by the model. The more the observed frequency differs from the expected value, the more significant is the pattern.

We should point out that while our main motivation is to assess itemsets, the discovered background model is a true statistical global model and is useful for other purposes as well, such as model selection or data generation.

Example 5.1. *Consider a dataset $\mathcal{D}$ with 3 items, a, b, and c, and 12 transactions, represented here in binary form.*

$$\mathcal{D} = \left\{ \begin{array}{l} (0,0,1), (0,0,1), (0,1,0), (0,1,0), \\ (1,0,0), (1,0,0), (1,0,1), (0,1,1), \\ (1,1,0), (1,1,1), (1,1,1), (1,1,1) \end{array} \right\}$$

Item a occurs in 7 out of 12 transactions. The same holds for items b and c. If $\mathcal{D}$ obeyed the independence model, then the number of transactions with two items active should be around $12 \times 3 \times (7/12)^2 \times 5/12 \approx 5$. However, we have in $\mathcal{D}$ only 3 out of 12 transactions with two items active. Hence, row margins, the fractions of transactions containing k ones, for $k = 0, \ldots, 3$, cannot be explained by the independence model in this case. Thus, we are motivated to build a model which incorporates the row margins. Our expectation is that such model will have a better likelihood.

5.2 Statistics as Background Knowledge

In this section we introduce several count statistics for transactional data, and indicate for which types of datasets they are useful. A *statistic* is a function $S : \mathcal{T} \to \mathbb{N}$ mapping a transaction t to an integer. All statistics are of the form $q_{\mathcal{D}}(S(\mathcal{A}) = k)$, where $q_{\mathcal{D}}$ is the empirical distribution, i.e., our background knowledge will be the fraction of transactions in the data for which $S(t) = k$.

Column Margins

The simplest of statistics one can consider are the column margins or item probabilities. These probabilities can be used to define an independence model. This model has been used before in other works to estimate itemset frequencies [Brin et al., 1997, Aggarwal and Yu, 1998]. It has the advantage of being computationally fast and easy to interpret. Typically, however, it is a rather simplistic model and few datasets actually satisfy the independence assumption. Due to its simplicity and widespread use, we include the item probabilities in all of our models.

Row Margins

We define $ones(\mathcal{A})$ to be a random variable representing the number of ones in a random transaction. We immediately see that $ones(\mathcal{A})$ obtains integer values between 0 and N. Consequently, $p(ones(\mathcal{A}) = k)$ is the probability of a random transaction having k ones. Given a transaction $t \in \mathcal{T}$, we will also denote by $ones(t)$ the number of ones in t, that is,

$$ones(t) = \sum_{i=1}^{N} t_i \ .$$

The use of row margins (in conjunction with column margins) has been proposed before by Gionis et al. [2007], to asses the significance of (among others) frequent itemsets. However, their approach is different from ours (see Section 5.8). In said article, it is shown that for datasets with very skewed row margin distribution, most frequent itemsets, clusterings, correlations, etc. can be explained entirely by row and column margins alone. Supermarket basket datasets fall into this category, since the transactions in such data typically contain only a handful of items.

Lazarus Counts

A *lazarus event* in a transaction is defined as the occurrence of a zero within a string of ones.[1] This requires that a total order is specified on the attributes in $\mathcal{A}$. For simplicity, we assume that this order is $a_1 < \cdots < a_N$. Let $t \in \mathcal{T}$ be a transaction, then the lazarus count of t is defined as

$$laz(t) = |\{t_i = 0 \mid \text{there exist } a, b \text{ s.t. } a < i < b \text{ and } t_a = t_b = 1\}| \,.$$

The lazarus count of a transaction ranges from 0 to $N - 2$. If the lazarus count of t is 0, then t is said to satisfy the *consecutive-ones* property.

A specific case of datasets with the consecutive-ones property are banded datasets. These are datasets whose rows and columns can be permuted such that the non-zeroes form a staircase pattern. The properties of such banded binary matrices have been studied by Garriga et al. [2008], who presented algorithms to determine the minimum number of item perturbations it would take for a dataset to be fully banded. A banded or approximately banded dataset can be characterized as follows: the majority of the transactions will have a low lazarus count, and typically the row margins are low as well.

Transaction Bounds

For certain types of data it can be useful to examine at which positions the ones in the transactions of a dataset begin and end; or, what the lowest and highest items in the transaction are. Given a transaction $t \in \mathcal{T}$, we define the *first* and *last* statistics as

$$\begin{aligned} first(t) &= \min\{i \mid t_i = 1\} \,, \\ last(t) &= \max\{i \mid t_i = 1\} \,. \end{aligned}$$

If t contains only zeroes, then we define $first(t) = last(t) = 0$.

A dataset is called *nested* if for each pair of rows, one is always a subset of the other [Mannila and Terzi, 2007]. For such data, the rows and columns can be permuted such that the rows have consecutive ones starting from the first column. Thus, assuming the permutation is done, transactions in nested

[1] The term *lazarus event* hails from the paleontology literature, when a lack of evidence of occurrence would suggest that a species seems to have become extinct at a certain time period, only to reappear later on. The name refers to a biblical character that was raised form the dead. In Dutch, however, the phrase *being lazarus* means to be really, really drunk.

datasets will have a low lazarus count and low left bound $first(t)$. Nestedness has been studied extensively in the field of ecology for absence/presence data of species (see, for example, [Mannila and Terzi, 2007] for more details).

5.3 Maximum Entropy Model

The independence model can be seen as the simplest model that we can infer from binary data. This model has many nice computational properties: learning the model is trivial and querying it is a simple and fast procedure. Moreover, a fundamental theorem shows that the independence model is the distribution maximizing the entropy among all the distributions that have the same frequencies for the individual items. Our goal is to define a more flexible model. We require that we should be able to infer such model efficiently and we should be able to make queries. In order to do that we will use the Maximum Entropy principle [Csiszár, 1975, Jaynes, 1982].

Definition of the Maximum Entropy Model

Say that we have calculated a certain set of statistics from a dataset and we wish to build a distribution, such that this distribution satisfies the same statistics. The maximum entropy approach gives us the means to do that.

More formally, assume that we are given a function $S : \mathcal{T} \to \mathbb{N}$ mapping a transaction t to an integer value. For example, this function can be $ones(t)$, the number of ones in a transaction. Assume that we are given a dataset $\mathcal{D}$ with N attributes and let $q_{\mathcal{D}}$ be its empirical distribution. We associate a statistic n_k with S for any $k = 1, \ldots, K$ as the proportion of transactions in the data for which $S(t) = k$, that is, $n_k = q_{\mathcal{D}}(S(\mathcal{A}) = k)$, where K is the maximum value that S can attain. Since $\mathcal{T}$ is finite, we know that $K < \infty$.

In addition to the statistics $\{n_k\}$ we also wish to use the margins of the individual attributes, that is, the probability of an attribute having a value of 1 in a random transaction. We denote these column margins by $m_i = q_{\mathcal{D}}(a_i = 1)$, where $i = 1, \ldots, N$.

The distribution p^* is the unique distribution maximizing the entropy among the distributions having the statistics m_i and n_k. To be more precise, we define $\mathcal{P}$ to be the set of distributions satisfying the statistics m_i and n_k,

$$\mathcal{P} = \{p \mid p(a_i = 1) = m_i,\ p(S(\mathcal{A}) = k) = n_k \text{ for } i = 1, \ldots, N, k = 1, \ldots, K\} \ .$$

The maximum entropy distribution maximizes the entropy in $\mathcal{P}$,

$$p^* = \arg\max_{p \in \mathcal{P}} \; H(p) \; . \tag{5.1}$$

Note that $\mathcal{P}$ and consequently p^* depend on the statistics m_i and n_k, yet we have omitted them from the notation for the sake of clarity.

Our next step is to demonstrate a classic result that the maximum entropy distribution has a specific form. In order to do so, we start by defining *indicator functions* $M_i : \mathcal{T} \to \{0,1\}$ such that $M_i(t) = t_i$, that is, M_i indicates whether the attribute a_i has a value of 1 in a transaction t. We also define $T_k : \mathcal{T} \to \{0,1\}$ such that $T_k(t) = 1$ if and only if $S(t) = k$.

Theorem 5.1 (Theorem 3.1 in [Csiszár, 1975])**.** *The maximum entropy distribution given in Equation 5.1 has the form*

$$p^*(\mathcal{A} = t) = \begin{cases} u_0 \prod_{i=1}^{N} u_i^{M_i(t)} \prod_{k=1}^{K} v_k^{T_k(t)} & \text{if } t \notin Z \\ 0 & \text{if } t \in Z \end{cases} \tag{5.2}$$

for a set $Z \subseteq \mathcal{T}$, and real-valued parameters u_i, v_k, where u_0 acts as a normalization factor. Moreover, a transaction t is in Z if and only if $p(\mathcal{A} = t) = 0$ for all distributions p in $\mathcal{P}$.

Equation 5.2 is guaranteed to hold only if a certain technical assumption is made about $\mathcal{P}$. We need to assume that for each transaction t there is a distribution p such that $p(\mathcal{A} = t) > 0$. Generally, this will not be true. In our setting, the transactions in Z are the ones that are inconsistent with the statistics m_i and n_k. For example, a transaction t where $M_i(t) = 0$, but the corresponding row margin $m_i = 1$; or, $T_k(t) = 1$, but the corresponding statistic $n_k = 0$. However, these special cases can easily be taken into account, and hence this technicality is not a problem in practice.

Our goal is now to discover the parameters u_i and v_k, and provide an algorithm for making queries from this model.

Example 5.2. *The margins for the toy example given in Example 5.1 are $m_1 = m_2 = m_3 = 7/12$, and $n_0 = 0$, $n_1 = 1/2$, $n_2 = 1/4$, $n_3 = 1/4$. The parameters for the maximum entropy model for these margins are $u_0 = 1/8$, $u_1 = u_2 = u_3 = 1$, and $v_0 = 0$, $v_1 = 4/3$, $v_2 = 2/3$, and $v_3 = 2$. We can verify that these are correct by simply computing that p^* has the desired margins. For example,*

$$p^*(\mathit{ones}(\mathcal{A}) = 1) = 4/3(1/8 + 1/8 + 1/8) = 1/2 \; .$$

It turns out that we can represent our maximum entropy model as a mixture model. Such a representation will be fruitful when we are solving and querying the model. To see this, let u_i and v_k be the parameters in Eq. 5.2. We define a distribution q for which we have

$$q(\mathcal{A} = t) = Z_q^{-1} \prod_{i=1}^{N} u_i^{M_i(t)} ,$$

where Z_q is a normalization constant so that q is a proper distribution. Since M_i depends only on a_i we see that q is actually an independence distribution. This allows us to replace the parameters u_i with more natural parameters $q_i = q(a_i = 1)$. A simple calculation reveals that

$$u_i = \frac{q(a_i = 1, \mathcal{A} - a_i = 0)}{q(\mathcal{A} = 0)} = \frac{q_i \prod_{j \neq i}(1 - q_j)}{\prod_j (1 - q_j)} = \frac{q_i}{1 - q_i} .$$

Hence it is easy to compute q_i from u_i and vice versa. It should be stressed that q_i is not necessarily equal to m_i, the column margin of a_i in the data.

Example 5.3. *Consider the parameters u_i in Example 5.2. We see that $q_i = 1/2$, since $u_i = 1 = 0.5/(1 - 0.5) = q_i/(1 - q_i)$. Hence in this case the distribution q is actually the uniform distribution.*

Our next step is to consider the parameters v_k for the statistic S. First, we define a distribution

$$r(S(\mathcal{A}) = k) = Z_r^{-1} v_k q(S(\mathcal{A}) = k) ,$$

where Z_r is a normalization constant such that r is a proper distribution. We can now express the maximum entropy model p^* using r and q. By rearranging the terms in Eq. 5.2 we have

$$p^*(\mathcal{A} = t) = \frac{v_k}{Z_r} q(\mathcal{A} = t) = r(S(\mathcal{A}) = k) \frac{q(\mathcal{A} = t)}{q(S(\mathcal{A}) = k)} , \tag{5.3}$$

where $k = S(t)$. The right-hand side of the equation is a mixture model. According to this model, we first sample an integer, say $1 \leq k \leq K$, from r. Once k is selected we sample the actual transaction from $q(\mathcal{A} = t \mid S(\mathcal{A}) = k)$.

We should point out that we have some redundancy in the definition of r. Namely, we can divide each v_k by Z_r and consequently remove Z_r from the equation. However, keeping Z_r in the equation proves to be handy later on.

Example 5.4. *In our toy example, assume that the parameters of q are $q_i = 1/2$. Consequently, q is the uniform distribution, and $q(S(t) = 0) = q(S(t) = 3) = 1/8$ and $q(S(t) = 1) = q(S(t) = 2) = 3/8$, where we take $S(t) = ones(t)$. Thus, the values for v_k are then $v_0 = 0/(1/8) = 0$, $v_1 = (1/2)/(3/8) = 4/3$, $v_2 = (1/4)/(3/8) = 2/3$, and $v_3 = (1/4)/(1/8) = 2$. Note that these are exactly the parameters give in Example 5.2. The reason for this is that we started with the correct parameters q_i.*

Combining Multiple Statistics

We can generalize our model by allowing multiple statistics together. By combining statistics, we can construct more detailed models that are based on relatively simple background information.

We distinguish two alternatives to combine count statistics. Assume, for the sake of exposition, we have two statistics $S_1 : \mathcal{T} \to \mathbb{N}$ and $S_2 : \mathcal{T} \to \mathbb{N}$, with $S_1(t) \leq K_1$ and $S_2(t) \leq K_2$ for all $t \in \mathcal{T}$. Then we can either consider using the joint probabilities $q_{\mathcal{D}}(S_1(\mathcal{A}) = k_1, S_2(\mathcal{A}) = k_2)$ for $k_1 \leq K_1$ and $k_2 \leq K_2$; or, we may use the marginal probabilities $q_{\mathcal{D}}(S_1(\mathcal{A}) = k_1)$ and $q_{\mathcal{D}}(S_2(\mathcal{A}) = k_2)$ separately. The joint case can be easily reduced to the case of a single statistic by constructing a new statistic $S' = S_1 + (K_1 + 1)S_2$. Solving and querying the model in the marginal case can be done using the same techniques and the same time complexity as with the joint case; we simply marginalize this joint statistic.

5.4 Solving the Maximum Entropy Model

In this section we introduce an algorithm for finding the correct distribution. We will use the classic Iterative Scaling procedure. For more details on this algorithm and the proof of correctness we refer to the original paper by Darroch and Ratcliff [1972].

The generic Iterative Scaling algorithm is a framework that can be used for discovering the maximum entropy distribution given any set of linear constraints. The idea is to search the parameters and update them in an iterative fashion so that the statistics we wish to constrain will converge towards the desired values. In our case, we update u_i and v_k iteratively so that the statistics of the distribution will converge into m_i and n_k. The sketch of the algorithm is given in Algorithm 5.1.

Algorithm 5.1: IterScale

input : a set of column margins m_i; a set of statistics n_k
output: the maximum entropy distribution p^* satisfying m_i and n_k

1 **repeat**
2 **for each** $i = 1, \ldots, N$ **do**
   // update q_i such that $p^*(a_i = 1) = m_i$
3 $d \leftarrow p^*(a_i = 1)$
4 $c \leftarrow m_i(1 - d)/((1 - m_i)d)$
5 $q_i \leftarrow q_i c/(1 - (1 - q_i)c)$
6 $Z_r \leftarrow \sum_{k=1}^{K} v_k q(S(\mathcal{A}) = k)$
7 **end**
   // update v_k such that $p^*(S(\mathcal{A}) = k) = n_k$
8 **for each** $k = 1, \ldots K$ **do** $p_k \leftarrow p^*(S(\mathcal{A}) = k)$
9 **for each** $k = 1, \ldots K$ **do** $v_k \leftarrow v_k n_k / p_k$
10 $Z_r \leftarrow 1$
11 **until** p^* converges
12 **return** p^*

In order to use the Iterative Scaling algorithm we need techniques for computing the probabilities $p^*(a_i = 1)$ and $p^*(S(\mathcal{A}) = k)$, for a statistic S (lines 3 and 8). The former is a special case of computing the frequency of an itemset $p^*(X = 1)$ and is detailed in Section 5.6. For the latter, assume that we have an algorithm, say ComputeStatProb, which given a set of probabilities $p_1, \ldots, p_N$ will return the probabilities $p(S(\mathcal{A}) = k)$ for each $k = 1, \ldots, K$, where p is the independence model parameterized by p_i, i.e., $p(a_i = 1) = p_i$. Using only ComputeStatProb we are able the obtain the required probabilities. Note that Equation 5.3 implies that we can compute $p^*(S(\mathcal{A}) = k)$ from $q(S(\mathcal{A}) = k)$. To compute the latter we call ComputeStatProb with $p_i = q_i$ (for $i = 1, \ldots, N$) as parameters.

5.5 Computing Statistics

In order to use the Iterative Scaling algorithm we need an implementation of ComputeStatProb, a routine that returns the probabilities of statistic S with respect to the independence model. In this section we describe for several

Table 5.1: Time and memory complexity of the ComputeStatProb algorithm for various count statistics. N is the number of attributes, and K is the number of distinct values the statistic can assume.

Statistic	*memory*	*time*
row margins (from scratch)	$O(K)$	$O(KN)$
row margins (update method)	$O(N)$	$O(N)$
lazarus counts	$O(K)$	$O(KN)$
transaction bounds	$O(N^2)$	$O(N^2)$
joint row margins and lazarus counts	$O(N^2)$	$O(K^2N)$
joint row margins and transaction bounds	$O(N^3)$	$O(KN^2)$

statistics how they can be computed efficiently—namely for row margins, lazarus counts, and transaction bounds. Since ComputeStatProb is called multiple times, its runtime consumption is pivotal. Naturally, memory and time requirements depend on the statistic at hand. In Table 5.1 the complexities of ComputeStatProb are listed for several statistics and joint statistics. All statistics are computed using the same general dynamic programming idea: to compute the statistics for items $\{a_1, \ldots, a_i\}$, we first solve the problem for items $\{a_1, \ldots, a_{i-1}\}$, and using that result we will be able to compute efficiently the statistics for $\{a_1, \ldots, a_i\}$. To simplify notation, we define $\mathcal{A}_i = \{a_1, \ldots, a_i\}$ to be the set of the first i items.

Row Margins

The first statistic we consider is the number of ones in a transaction $ones(t)$, ranging from 0 to N. We must calculate the probabilities $p(ones(\mathcal{A}) = k)$, where p is the independence model. Let us write $p_i = p(a_i = 1)$, the probability of attribute a_i attaining the value of 1. We first introduce a way of computing the probabilities from scratch. In order to do that, note the following identity for $k > 0$.

$$p(ones(\mathcal{A}_i) = k) = p_i p(ones(\mathcal{A}_{i-1}) = k-1) + (1-p_i)p(ones(\mathcal{A}_{i-1}) = k) \tag{5.4}$$

This identity holds since p is the independence model. Hence, to compute $p(ones(\mathcal{A}) = k)$ we start with $p(ones(a_1) = k)$, add a_2, a_3, and so on, until

we have processed all variables and reach $\mathcal{A}$. Algorithm 5.2 gives the pseudo code of ComputeRowMarginsProb. Note that we are simultaneously computing the probabilities for all k. We can perform the computation in $O(N^2)$ steps and $O(N)$ memory slots.

We can improve this further by analyzing the flow of the Iterative Scaling algorithm. The algorithm calls ComputeStatProb either using parameters $p_1 = q_1, \ldots, p_N = q_N$, or $p_1 = q_1, \ldots, p_i = 1, \ldots, p_N = q_N$ for some i.

Assume that we have $q(ones(\mathcal{A}) = k)$ from the previous computation. These probabilities will change only when we are updating q_i. To achieve more efficient computation we will first compute $q(ones(\mathcal{A}) = k \mid a_i = 1)$ from $q(ones(\mathcal{A}) = k)$, update q_i and then update $q(ones(\mathcal{A}) = k)$. To do that we can reverse the identity given in Eq. 5.4 into

$$p(ones(\mathcal{A} - a_i) = k) = \frac{1}{1 - p_i} \left(p(ones(\mathcal{A}) = k) - p_i p(ones(\mathcal{A} - a_i) = k - 1) \right) ,$$

if $k = 1, \ldots, N$. When $k = 0$ we have

$$p(ones(\mathcal{A} - a_i) = 0) = p(ones(\mathcal{A}) = 0) / (1 - p_i) .$$

Using these identities we can take a step back and remove a_i from $\mathcal{A}$. To compute $q(ones(\mathcal{A}) = k \mid a_i = 1)$ we can apply the identity in Eq. 5.4 with $p_i = 1$. Once q_i is updated we can again use Eq. 5.4 to update $q(ones(\mathcal{A}) = k)$. The UpdateRowMarginProb algorithm is given as Algorithm 5.3. All these computations can be done in $O(N)$ time, and $O(N)$ space.

Often, we are only interested in small transactions (for instance when dealing with supermarket data, which is typically very sparse), especially when N is large. Therefore, we can consider a truncated (and more general) version of $ones(t)$ by defining

$$ones(t; K) = \min(ones(t) , K) .$$

Hence, $p(ones(\mathcal{A}; K) = k)$ is the probability of a random transaction having k ones, if $k < K$. On the other hand $p(ones(\mathcal{A}; K) = K) = p(ones(\mathcal{A}) \geq K)$ is the probability of a random transaction having at least K ones. The range of $ones(\mathcal{A}; K)$ is $0, \ldots, K$. We can exploit the fact that $p(ones(\mathcal{A}) \geq K) = 1 - \sum_{k=0}^{K-1} p(ones(\mathcal{A}) = k)$ to reduce computation time. More specifically, the runtime of ComputeRowMarginProb is reduced from to $O(N^2)$ to $O(NK)$, and its memory usage from $O(N)$ to $O(K)$. The time and memory complexity of ComputeRowMarginProb, on the other hand, both remain $O(N)$.

Algorithm 5.2: ComputeRowMarginProb

input : an independence distribution p with parameters $p_1, \ldots, p_N$
output: the probabilities $c_k = p(ones(\mathcal{A}) = k)$ for $k = 0, \ldots, N$

$c_0 \leftarrow 1$
for $k = 1, \ldots, N$ **do**
  $c_k \leftarrow 0$
end
for $i = 1, \ldots, N$ **do**
  for $k = i, \ldots, 1$ **do**
    $c_k \leftarrow p_i \cdot c_{k-1} + (1 - p_i) \cdot c_k$
  end
  $c_0 \leftarrow (1 - p_i) \cdot c_0$
end
return all c_k

Lazarus Events

Our aim is to efficiently compute the probabilities $p(laz(\mathcal{A}) = k)$ where p is an independence distribution. Again, let us write $p_i = p(a_i = 1)$. The desired probabilities are computed incrementally in N steps, starting from $p(laz(a_1) = k)$, up to $p(laz(\mathcal{A}) = k)$. In order to determine the lazarus count probabilities, we use an auxiliary statistic, the last occurrence $last(t)$.

We will compute the probabilities $p(laz(\mathcal{A}) = k, last(\mathcal{A}) = j)$ and then marginalize them to obtain $p(laz(\mathcal{A}) = k)$. To simplify the notation below, we define the probability

$$p^{(i)}(k, j) = p\left(laz(\mathcal{A}_i) = k, last(\mathcal{A}_i) = j\right) \ .$$

First, assume that $last(\mathcal{A}_i) = i$, which implies that $a_i = 1$. In this case the lazarus count increases by $i - last(\mathcal{A}_{i-1}) - 1$. We have the following identity

$$p^{(i)}(k, i) = p_i \sum_{l=0}^{k} p^{(i-1)}(l, i + l - k - 1) \ , \tag{5.5}$$

for $k = 1, \ldots, i - 2$. For the boundary cases, for $k = 0$ and $i > 1$ we have

$$p^{(i)}(0, i) = p_i p^{i-1}(0, i - 1) + p_i p^{(i-1)}(0, 0) \ ,$$

Algorithm 5.3: UpdateRowMarginProb

input : the probabilities $c_k = p(ones(\mathcal{A}) = k)$ for an independence distribution p with parameters $p_1, \ldots, p_N$;
an updated probability p'_j for item a_j
output: the updated probabilities $c_k = p(ones(\mathcal{A}) = k)$ for $k = 0, \ldots, N$

// remove item a_j
$c_0 \leftarrow c_0/(1 - p_j)$
for $k = 1, \ldots, N$ **do**
  $c_k \leftarrow \left(c_k - p_j \cdot c_{k-1}\right)/(1 - p_i)$
end
// re-add item a_j with new probability p'_j
for $k = N, \ldots, 1$ **do**
  $c_k \leftarrow p'_j \cdot c_{k-1} + (1 - p'_j) \cdot c_k$
end
$c_0 \leftarrow (1 - p'_j) \cdot c_0$
return all c_k

and finally when $k = 0$ and $i = 1$ we obtain

$$p^{(1)}(0, 1) = p(laz(a_1) = 0, last(a_1) = 1) = p_1 \ .$$

Secondly, consider the instances where $last(\mathcal{A}_i) < i$, in which case a_i must be equal to 0. We obtain $p^{(i)}(k, j) = (1 - p_i)p^{(i-1)}(k, j)$ for all $k = 0, \ldots, i - 2$ and $j = 0, \ldots, i - 1$.

Using the equations above, we are able to compute $p(laz(\mathcal{A}) = k)$ using $O(N^2)$ memory slots. Specifically, for $N(N-1)/2 + 2$ out of all $N^2 - 1$ combinations of j and k, is $p(laz(\mathcal{A}) = k, last(\mathcal{A}) = j)$ potentially nonzero. Since in each step a quadratic number of probabilities is updated, $O(N^3)$ time is needed. However, we can reduce this to quadratic time and linear space. First of all, note that the right-hand side of Equation 5.5, being a sum of size linear in k, can be rewritten as a sum of constant size. Moreover, this sum only uses probabilities involving $\{a_1, \ldots, a_{i-1}\}$ having $a_i = 1$, which were computed in

the previous step. Hence for $k > 1$, the probability $p^{(i)}(k,i)$ equals

$$\begin{aligned} p^{(i)}(k,i) &= p_i \sum_{l=0}^{k} p^{(i-1)}(l, i+l-k-1) \\ &= p_i p^{(i-1)}(k, i-1) + p_i \sum_{l=0}^{k-1} p^{(i-1)}(l, i+l-k-1) \\ &= p_i p^{(i-1)}(k, i-1) + p_i \frac{1-p_{i-1}}{p_{i-1}} p^{(i-1)}(k-1, i-1) \,. \end{aligned}$$

For $k = 1$, it holds that

$$\begin{aligned} p^{(i)}(1,i) &= p_i p^{(i-1)}(1, i-1) + p_i p^{(i-1)}(0, i-2)) \\ &= p_i p^{(i-1)}(1, i-1) + p_i \frac{1-p_{i-1}}{p_{i-1}} p^{(i-1)}(0, i-1) - p_i p^{(i-1)}(0,0) \,. \end{aligned}$$

Finally, noting that

$$\begin{aligned} p(laz(\mathcal{A}) = k, last(\mathcal{A}) \leq j) = p(laz(\mathcal{A}) = k, last(\mathcal{A}) \leq j-1) \\ + p^{(j)}(k,j) \prod_{i=j+1}^{N} (1-p_i) \,, \end{aligned}$$

we can compute $p(laz(\mathcal{A}) = k)$ by gradually summing the second terms. The probability $p^{(j)}(k,j)$ can be computed in constant time using $p^{(j-1)}(k, j-1)$ and $p^{(j-1)}(k, j-1)$ only. Hence we can discard the terms $p^{(n)}(k,n)$, where $n < j-1$. Therefore, only $O(N)$ memory and $O(N^2)$ time is needed. The ComputeLazarusProb algorithm for the lazarus count statistic is given in Algorithm 5.4.

Joint Transaction Bounds

We need to compute $p(first(\mathcal{A}) = i, last(\mathcal{A}) = j)$ for an independence distribution p, with $i, j = 0, \ldots, N$. For the sake of brevity, let us denote $p(i,j) = p(first(\mathcal{A}) = i, last(\mathcal{A}) = j)$. Note that $p(i,j)$ is nonzero if $i = j = 0$ or $0 < i \leq j$. Hence there are $(N^2+N)/2+1$ probabilities to compute. We distinguish three cases

$$p(i,j) = \begin{cases} \prod_{k=1}^{N}(1-p_k) & \text{if } i = j = 0 \,, \\ p_i \prod_{k \neq i}(1-p_k) & \text{if } i = j \neq 0 \,, \\ \prod_{k=1}^{i-1}(1-p_k) p_i p_j \prod_{k=j+1}^{N}(1-p_k) & \text{if } 0 < i < j \,. \end{cases}$$

Algorithm 5.4: ComputeLazarusProb

input : an independence distribution p with parameters $p_1, \ldots, p_N$
output: the probabilities $c_k = p(laz(\mathcal{A}) = k)$ for $k = 0, \ldots, N-2$
// $c_k = p(laz(\mathcal{A}) = k, last(\mathcal{A}) < i)$ after the i-th round
// $r_k = p(Z(i,k,i))$ after the i-th round
// $s = p(Z(i-1,0,0))$ after i-th the round
// $u = \prod_{j=i}^{N} 1 - p_j$ after i-th the round

$r_0 \leftarrow p_1$
$s \leftarrow 1$
$u \leftarrow \prod_{j=1}^{N} 1 - p_j$
for each $i = 2, \ldots, N$ **do**
  $u \leftarrow u/(1 - p_{i-1})$
  for each $k = 0, \ldots, i-2$ **do**
    $c_k \leftarrow c_k + u r_k$
  end
  for each $k = i-2, \ldots, 2$ **do**
    $r_k \leftarrow p_i \left(r_k + \frac{1-p_{i-1}}{p_{i-1}} r_{k-1} \right)$
  end
  $s \leftarrow s(1 - p_{i-1})$
  $r_1 \leftarrow p_i \left(r_1 + \frac{1-p_{i-1}}{p_{i-1}} r_0 - s \right)$
  $r_0 \leftarrow p_i (r_0 + s)$
end
for each $k = 0, \ldots, N$ **do**
  $c_k \leftarrow c_k + r_k$
end
return all c_k

We can construct the $p(i,j)$ in quadratic time, by looping over i and j, and maintaining the products $\prod_{k=1}^{i}(1-p_k)$ and $\prod_{k=j}^{N}(1-p_k)$, which can be updated in constant time in each iteration. Using these products and the individual item probabilities, we can construct $p(i,j)$. The pseudo code of ComputeBoundsProb is given in Algorithm 5.5.

5.6 Estimating Itemset Frequencies

We noted before that computing $p^*(a_i = 1)$ is a special case of computing the frequency of an itemset $p^*(X = 1)$. In order to do that let us write

$$p^*(X=1) = \sum_{k=1}^{K} p^*(X=1, S(\mathcal{A})=k) = \sum_{k=1}^{K} \frac{v_k}{Z_r} q(X=1, S(\mathcal{A})=k) \; .$$

Let us denote $y = q(X=1) = \prod_{a_j \in X} q_j$. Note that since q is the independence model, we have $q(S(\mathcal{A})=k, X=1) = yq(S(\mathcal{A})=k \mid X=1)$. Now, to compute the quantity $q(S(\mathcal{A})=k \mid X=1)$ we call ComputeStatProb with parameters $p_j = q_j$ if $a_j \notin X$ and $p_j = 1$ if $a_j \in X$.

Example 5.5. *Assume that our model is the same toy model as given in Example 5.2. Let us compute the frequency estimate of $X = ab$. Since q is the uniform distribution, $q(ab=1) = 1/4$, and $q(ones(\mathcal{A})=2 \mid ab=1) = q(ones(\mathcal{A})=3 \mid ab=1) = 1/2$. Given that $v_2 = 2/3$ and $v_3 = 2$, the estimate is then*

$$p^*(ab=1) = \frac{1}{4}\left(\frac{2}{3}\frac{1}{2} + 2\frac{1}{2}\right) = \frac{1}{3} \; .$$

Note that this is exactly the frequency of the itemset ab in $\mathcal{D}$ given in Example 5.1.

5.7 Experiments

In this section we present the results of experiments on synthetic and real data. The source code of the C++ implementation of the algorithm is publicly available for download.[2]

[2] http://www.adrem.ua.ac.be/implementations

Algorithm 5.5: ComputeBoundsProb

input : an independence distribution p with parameters $p_1, \ldots, p_N$
output: the probabilities $c_{j,k} = p(\mathit{first}(\mathcal{A}) = j, \mathit{last}(\mathcal{A}) = k)$ for $j, k = 0, \ldots, N$

```
1  r ← 1
2  for j = 1, ..., N do
3  |  if j = 1 then
4  |  |  r ← p_1
5  |  else
6  |  |  r ← r · p_j · (1 − p_{j−1}) / p_{j−1}
7  |  end
8  |  for k = N, ..., j do
9  |  |  if j < k then
10 |  |  |  if k = N then
11 |  |  |  |  s ← p_k
12 |  |  |  else
13 |  |  |  |  s ← s · p_k · (1 − p_{k+1}) / p_{k+1}
14 |  |  |  end
15 |  |  else
16 |  |  |  if k = N then
17 |  |  |  |  s ← 1
18 |  |  |  else
19 |  |  |  |  s ← s · (1 − p_{k+1}) / p_{k+1}
20 |  |  |  end
21 |  |  end
22 |  |  c_{j,k} ← r · s
23 |  end
24 end
25 r ← r · (1 − p_N) / p_N
26 c_{0,0} ← r
27 return all c_{j,k}
```

Table 5.2: The datasets used in the experiments. Shown are the number of attributes $|\mathcal{A}|$ and the number of transactions $|\mathcal{D}|$.

Dataset	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{D}\rvert$
Independent	20	100 000
Clusters	20	100 000
Markov	20	100 000
BMS-Webview-1	150	52 840
Chess	75	3 196
DNA Amplification	391	4 590
Retail	221	81 998

Datasets

The basic characteristics of the datasets we used are given in Table 5.2.

We created three synthetic datasets. The first one has independent items with randomly chosen frequencies. The second dataset contains two clusters of equal size. In each cluster the items are independent with a frequency of 25% and 75% respectively. Hence, the row margin distribution has two peaks. In the third synthetic dataset, the items form a Markov chain. The first item has a frequency of 50%, and then each subsequent item is a noisy copy of the preceding one, that is, the item is inverted with a 25% probability.

The *DNA Amplification* dataset [Myllykangas et al., 2006] describes DNA copy number amplifications. The remaining real-world datasets we used are obtained from the FIMI Repository [Goethals and Zaki, 2003]. The *Chess* data contains chess board descriptions. The *BMS-Webview-1* dataset contains clickstream data from an e-commerce website [Kohavi et al., 2000], and the *Retail* dataset contains market basket data from an anonymous Belgian supermarket [Brijs et al., 1999]. For those two last datasets, the rare items with a frequency lower than 0.5% were removed. This resulted in some empty transactions which were subsequently also deleted.

Note that for the lazarus count and transaction bound statistics, an order is needed on the items. In this work we resort to using the order in which the items appear in the data. Despite the fact that this order is not necessarily optimal, we were able to improve over the independence model.

Table 5.3: Performance details of the algorithm. For each dataset, we show the time required to learn the various models, and the number of iterations until the algorithm converges.

	Margins		*Lazarus*		*Bounds*	
Dataset	iter	time	iter	time	iter	time
Independent	2	0.01 s	2	0.02 s	2	0.02 s
Clusters	2	0.01 s	9	0.07 s	10	0.07 s
Markov	2	0.01 s	8	0.05 s	12	0.07 s
BMS-Webview-1	3	5 s	14	45 s	93	267 s
Chess	3	0.6 s	400	153 s	28	8 s
DNA Amplification	8	313 s	96	90 m	119	66 m
Retail	4	26 s	11	110 s	19	171 s

Model Performance

Table 5.3 shows for each model the wall clock time and the number iterations the algorithm required to converge. For the majority of the datasets and models, we see that this is quite fast; often only a handful of iterations is required, and the runtime can usually be expressed in minutes or seconds.

In Table 5.4, we examine the log-likelihood of the learned models. We train the model on the whole data and then compute its likelihood, giving it a BIC penalty (see Section 2.3), equal to $\frac{k}{2} \log |\mathcal{D}|$ where k is the number of free parameters of each model. This penalty rewards models with few parameters, while penalizing complex ones.

Compared to the independence model, the likelihoods are all better on all datasets with the exception of the *Independent* data. This is expected since the *Independent* data is generated from an independence distribution, so using more advanced statistics will not improve the BIC score.

When looking at the other two synthetic datasets, we see that the margin model has the highest likelihood for the *Clusters* data, and the lazarus model for the *Markov* data. The *Clusters* data has two clusters, in both of which the items are independent. The distribution of the row margins has two peaks, one for each cluster. This information cannot be explained by the independence model alone, and adding this information improves the log-

Table 5.4: The BIC scores (negative log-likelihood plus BIC penalty) of the datasets for different models. The best (lowest) values are indicated in bold.

Dataset	*Independent*	*Margins*	*Lazarus*	*Bounds*
Independent	**1 658 486**	1 658 630	1 658 621	1 658 779
Clusters	2 000 159	**1 719 959**	1 889 308	1 946 942
Markov	2 000 159	1 938 960	**1 861 046**	1 890 648
BMS-Webview-1	836 624	778 361	783 733	**774 773**
Chess	142 054	132 921	**131 870**	137 213
DNA Amplification	185 498	173 305	**107 739**	109 572
Retail	1 796 126	**1 774 291**	1 783 054	1 775 588

likelihood dramatically. The attributes in the *Markov* dataset are ordered since they form a Markov chain. More specifically, since each attribute is a (noisy) copy of the previous one, we expect the transactions to consist of only a few blocks of consecutive ones, which implies that their lazarus count is quite low, and hence the lazarus model performs well.

Chess is originally a categorical dataset, which has been binarized to contain one item for each attribute-value pair. Hence, it is a rather dense dataset, with constant row margins, and lazarus counts and bounds centered around a peak. That is, *Chess* does not obey the independence model and we see that the likelihoods of all models are better than that of the independence model.

The likelihood of the lazarus model for the *DNA Amplification* dataset, which is very close to being fully banded [Garriga et al., 2008], is substantially lower than that of the independence model and the margin model, which indicates that using lazarus counts is a good idea in this case. The bounds model comes in second, also performing very well, which can again be explained by the bandedness of the data.

Finally, *BMS-Webview-1* and *Retail* are sparse datasets. The margin model performs well for both datasets. Therefore we can conclude that a lot of the structure of these datasets is captured in the row and column margins.

Frequency Estimation of Itemsets

Next, we perform experiments on estimating the supports of a collection of itemsets. The datasets are split in two parts, a training set and a test set.

We train the models on the training data, and use the test data to asses the frequency estimates of the top-10 000 closed frequent itemsets in the test data (or all closed itemsets if there are fewer).

Table 5.5 reports the average absolute and relative errors of the frequency estimates, for the independence, row margins, lazarus, and bounds models. For the *Independent* data, the independence model performs best, since the other models overfit the data. For all other datasets, except *Chess*, we see that using more information reduces both the average absolute and relative error of the frequency estimates. For instance, for *Clusters* the average absolute error is reduced from 9.39% to 0.2% using row and column margins, and likewise for *Markov* the average relative error is reduced from 47.8% to 21.11%. The *DNA Amplification*, *Retail* and *BMS-Webview-1* data are sparse. Therefore, the itemset frequencies are very low, as well as the absolute errors, even for the independence model. However, the relative errors are still quite high. In this case our models also outperform the independence model. For example, the relative error is reduced from 92.61% to 79.77% by using the margins model on the *BMS-Webview-1* dataset. For *DNA Amplification*, the average relative error drops 5% using lazarus counts.

The only exception is the *Chess* dataset where the average absolute and relative errors do not improve over the independence model. Note, however, that this dataset is originally categorical, and contains a lot of dependencies between the items. Interestingly enough, our models perform better than the independence model in terms of (penalized) likelihood. This suggests that in this case using additional information makes some itemsets that were not significant with respect the independence model, significant with respect a more suitable model.

The last experiment (given in Table 5.6) is the average improvement of log-likelihood of each itemset compared to the independence model. Let f be the empirical frequency of itemset X from the test data, and let p be the estimate given by one of the models, then the log-likelihood of X is computed as $|\mathcal{D}|(f \log p + (1-f)\log(1-p))$. We compute the difference between the log-likelihoods for the independence model and the other models, and take the average over the top 10 000 closed frequent itemsets. Again, for *Independent* and *Chess*, the independence model performs the best. For all the other datasets, we clearly see an improvement with respect to the independence model. For *Clusters* and *Markov*, the average log-likelihood increases greatly, and is highest for the margin model. For *BMS-Webview-1*, *DNA Amplification*,

Table 5.5: Average absolute and relative error of the top-10 000 closed frequent itemsets in the test data. Lower scores are better.

	Independent		*Margins*	
Dataset	absolute	relative	absolute	relative
Independent	0.11% ± 0.10%	1.54% ± 01.23%	0.11% ± 0.10%	1.52% ± 01.21%
Clusters	9.39% ± 0.73%	63.08% ± 11.81%	0.20% ± 0.11%	1.37% ± 00.86%
Markov	4.79% ± 2.29%	47.80% ± 20.86%	2.19% ± 1.64%	21.11% ± 13.41%
BMS-Webview-1	0.11% ± 0.07%	92.61% ± 17.27%	0.10% ± 0.06%	79.77% ± 23.94%
Chess	1.81% ± 1.35%	2.35% ± 01.80%	01.94% ± 1.43%	2.52% ± 01.91%
DNA Amplification	0.58% ± 1.08%	85.89% ± 30.91%	0.56% ± 1.04%	84.27% ± 31.76%
Retail	0.05% ± 0.11%	48.89% ± 27.95%	0.04% ± 0.09%	37.70% ± 28.64%
	Lazarus		*Bounds*	
Dataset	absolute	relative	absolute	relative
Independent	0.11% ± 0.10%	01.54% ± 1.23%	0.12% ± 0.10%	1.61% ± 01.30%
Clusters	7.00% ± 1.12%	47.04% ± 10.77%	8.21% ± 0.93%	55.33% ± 11.92%
Markov	2.27% ± 1.61%	22.41% ± 14.88%	3.36% ± 2.24%	33.54% ± 21.12%
BMS-Webview-1	0.11% ± 0.06%	88.35% ± 20.87%	0.11% ± 0.06%	90.60% ± 18.63%
Chess	2.06% ± 1.49%	2.68% ± 01.99%	1.81% ± 1.35%	2.35% ± 01.79%
DNA Amplification	0.45% ± 0.75%	80.24% ± 72.73%	0.54% ± 0.94%	82.63% ± 31.38%
Retail	0.04% ± 0.10%	42.90% ± 27.81%	0.04% ± 0.09%	43.71% ± 27.27%

Table 5.6: The average improvement of itemset log-likelihoods over the independence model for the top-10 000 closed frequent itemsets in the test data. Higher scores are better.

Dataset	*Margins*	*Lazarus*	*Bounds*
Independent	0.03 ± 0.12	−0.00 ± 0.09	−0.10 ± 0.36
Clusters	4 517.4 ±1 168.3	2 412.6 ±1 030.1	1 353.2 ±660.4
Markov	1 585.9 ±1 310.5	1 536.9 ±1 283.2	931.5 ±854.1
BMS-Webview-1	135.79 ± 90.32	64.72 ± 40.41	57.71 ±62.95
Chess	−0.40 ± 0.52	−0.81 ± 0.98	0.01 ± 0.17
DNA Amplification	119.9 ± 235.7	133.1 ± 274.5	106.2 ±224.8
Retail	9.69 ± 22.49	4.62 ± 11.05	5.37 ±24.94

and *Retail*, the increase in likelihood is somewhat lower. The reason for this is that both the estimates and observed frequencies are small and close to each other. For *DNA Amplification* the average increase is highest when using lazarus counts, while for *BMS-Webview-1* and *Retail* the margin model is best.

5.8 Related Work

A special case of the presented framework greatly resembles the work done by Gionis et al. [2007]. In that work the authors propose a procedure for assessing the results of a data mining algorithm by sampling datasets having the same margins for the rows and columns as the original data. While the goals are similar, there is a fundamental difference between the two frameworks. The key distinction is that we do not differentiate individual rows. Thus we do not know that, for example, the first row in the data has 5 ones but instead we know *how many* rows have 5 ones. The same key difference can be seen between our method and Rasch models where each individual row and column of the dataset is given its own parameter [Rasch, 1960].

Our approach and the approach given by Gionis et al. [2007] complement each other. When the results that we wish to assess do not depend on the order or identity of transactions, it is more appropriate to use our method. An example of such data mining algorithms is frequent itemset mining. On the other hand, if the data is to be treated as a *collection* of transactions, e.g.,

for segmentation, then we should use the approach by Gionis et al. [2007]. Also, our approach has a more theoretically sound ground since for sampling datasets the authors in [Gionis et al., 2007] rely on MCMC techniques with no theoretical guarantees that mixing has actually happened.

Comparing frequencies of itemsets to estimates has been studied in several works. The most common approach is to compare the itemset against the independence model [Brin et al., 1997, Aggarwal and Yu, 1998]. A more flexible approach has been suggested in [Jaroszewicz and Simovici, 2004, Jaroszewicz and Scheffer, 2005] where the itemset is compared against a Bayesian network. In addition, approaches where the maximum entropy models derived from some given known itemsets are suggested in [Meo, 2000, Tatti, 2008]. A common problem for these more general approaches is that the deriving of probabilities from these models is usually too complex. Hence, we need to resort to either estimating the expected value by sampling, or build a local model using only the attributes occurring in the query. In the latter case, it is shown by Tatti [2006b] that using only local information can distort the estimate and that the received frequencies are not consistent with each other. Our model does not suffer from these problems since it is a global model from which the frequency estimates can be drawn efficiently.

5.9 Conclusions

In this chapter we considered using count statistics such as row and column margins, lazarus counts, and transaction bounds as background knowledge to predict itemset frequencies. To this end we built a maximum entropy model from which we draw estimates for the frequency of itemsets and compare the observed value against the estimate. We introduced efficient polynomial techniques for solving and querying the model. Experiments showed that using these additional statistics improves the model in terms of likelihood and in terms of predicting itemset frequencies.

Chapter 6

Succinctly Summarizing Data with Informative Itemsets

Knowledge discovery from data is an inherently iterative and incremental process. That is, what we know about the data greatly determines our expectations, and therefore, which results we would find interesting and/or surprising. Given new knowledge about the data, our expectations will change, and hence, in order to avoid reporting and analyzing redundant results, knowledge discovery algorithms need to follow a similar iterative updating procedure.

With this in mind, we introduce a well-founded approach for succinctly summarizing data with the most informative itemsets; using a probabilistic maximum entropy model, we iteratively find the itemset that provides us the most novel information—that is, for which the frequency in the data, given what we already know about it, surprises us the most—and in turn we update our model accordingly. As we use the Maximum Entropy principle to obtain unbiased probabilistic models, and only include those itemsets that

This chapter is based on work published as:
M. Mampaey, N. Tatti, and J. Vreeken. Tell me what I need to know: Succinctly summarizing data with itemsets. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, pages 573–581. ACM, 2011. (*Best Student Paper*)
M. Mampaey, J. Vreeken, and N. Tatti. Summarizing data succinctly with the most informative itemsets. Manuscript currently submitted to: *Transactions on Knowledge Discovery and Data Mining*, 2011.

are most informative with regard to the current model, the summaries we construct are guaranteed to be both descriptive and non-redundant.

The algorithm presented in this chapter to mine these summaries can either discover the top-k most informative itemsets, or it can employ either the Bayesian Information Criterion (BIC) or the Minimum Description Length (MDL) principle to automatically identify the set of itemsets that as a whole provides the best summary of the data. In other words, our method can automatically 'tell you what you need to know' about the data, without requiring the user to set any parameters. Further, it is a one-phase algorithm; rather than picking itemsets from a user-provided candidate set, itemset supports are mined on the fly. We also provide an efficient method to compute the maximum entropy distribution, using Quick Inclusion-Exclusion.

Experimental evaluation of our method using synthetic, benchmark, and real data, shows that the discovered summaries are succinct, and correctly identify the key patterns in the data. The models they form attain high likelihoods, and inspection shows that they summarize the data well with increasingly specific, yet non-redundant itemsets.

6.1 Introduction

Knowledge discovery from data is an inherently iterative process. That is, what we already know about the data greatly determines our expectations, and therefore, which results we would find interesting and/or surprising. Early on in the process of analyzing a database, for instance, we are happy to learn about the generalities underlying the data, while later on we will be more interested in the specifics that build upon these concepts. Essentially, this comes down to summarization: we want to know what is interesting in the data, and we want this to be reported succinctly and without redundancy.

As a simple example, consider supermarket basket analysis. Say we just learned that *pasta* and *tomatoes* are sold together very often, and that we already knew that many people buy *wine*. Then it would not be very interesting to be told that the combination of these three items is also sold frequently; although we might not have been able to predict the sales numbers exactly, our estimate would most likely have come very close, and hence we can say that this pattern is redundant.

At the same time, at this stage of the analysis we are probably also not interested in highly detailed patterns, e.g., an itemset representing the many ingredients of an elaborate Italian dinner. While the frequency of this itemset may be surprising, the pattern is also highly specific, and may well be better explained by some more general patterns. Still, this itemset might be regarded as highly interesting further on in the discovery process, after we have learned those more general patterns, and if this is the case, we would like it to be reported at that time. In a nutshell, that is the approach we adopt in this chapter: we incrementally adjust our model as we iteratively discover the most informative patterns, in order to obtain a non-redundant summary.

As natural as it may seem to update a knowledge model during the discovery process, and in particular to iteratively find those results that are most informative with regard to what we have learned so far, few pattern mining techniques actually follow such a dynamic approach. That is, while many techniques provide a series of patterns in order of interestingness, most score these patterns using a static model; during this process the model, and hence the itemset scores, are not updated with the knowledge gained from previously discovered patterns. For instance, Tan et al. [2002] study 21 of the most well-known interestingness measures, all of which are static, and most of which are based on the independence model. The static approach inherently

gives rise to the typical problem of traditional pattern mining: overwhelmingly large and highly redundant collections of patterns.

Our objective is to find a succinct summary of a binary dataset, that is, to obtain a small, yet high-quality set of itemsets that describes key characteristics of the data at hand, in order to gain useful insights. This is motivated by the fact that many existing algorithms often return too large collections of patterns with considerable redundancy, as stated above. The view that we take in this chapter on succinctness and redundancy is therefore fairly strict.

While we are not the first to propose a method that updates its scoring model dynamically—examples include the swap randomization-based approach by Hanhijärvi et al. [2009], and the compression-based approach by Vreeken et al. [2011]—there are several differences with the existing approaches. For instance, the former requires generating many complete randomized databases in order to estimate frequencies, whereas our model is probabilistic and allows for direct frequency calculation. The models for the latter are not probabilistic, and while non-redundant with respect to compression, due to its coding strategy can contain patterns that are variations of the same theme. We will discuss related work in closer detail in Section 6.2, but let us first discuss the basic properties of our approach.

To model the data, we use the powerful and versatile class of maximum entropy models. We construct a maximum entropy distribution that allows us to directly calculate the expected frequencies of itemsets. Then, at each iteration, we return the itemset that provides the most information, i.e., for which our frequency estimate was most off. We update our model with this new knowledge, and continue the process. The non-redundant model that contains the most important information is thus automatically identified. Therefore, we paraphrase our method as 'tell me what I need to know'.

While in general solving the maximum entropy model is infeasible, we show that in our setting it can be computed efficiently—depending on the amount of overlap between the selected patterns. Similarly, we give an efficient method for estimating frequencies from the model. Further, we provide a convex heuristic for effectively pruning the search space when mining the most informative itemsets. This heuristic allows us to mine collections of candidate itemsets on the fly, instead of picking them from a larger candidate collection that has to be materialized beforehand. Another important aspect of our approach is that it is parameter-free: no maximal error threshold or significance level needs to be provided, nor is the user required to provide

a minimum support threshold for the candidate collection—although, we do allow the user to specify a collection of itemsets from which to select the most informative summary.

We formalize the problem of identifying the most informative model both by the Bayesian Information Criterion (BIC), as well as by the Minimum Description Length (MDL) principle; both are well-known and well-understood model selection techniques that have natural interpretations. Alternatively, due to its iterative nature, our algorithm can also mine the top-k most interesting itemsets. Finally, our approach easily allows the user to infuse background knowledge into the model (in the form of itemset frequencies, column margins, and/or row margins), to the end that redundancy with regard to what the user already knows can be effectively avoided.

Experiments on real and synthetic data show that our approach results in succinct, non-redundant data summaries using itemsets, and provide intuitive descriptions of the data. Since they only contain a small number of key patterns from the data, they can easily be inspected manually by the user, and since redundancy is reduced to a minimum, the user knows that every pattern he or she looks at will be informative.

This chapter is organized as follows. First, Section 6.2 discusses related work. In Section 6.3 we give an introduction to Maximum Entropy models and how we can use these to measure the interestingness of a set of itemsets efficiently. We give a formal problem statement in Section 6.4. Next, we present our algorithm in Section 6.5. In Section 6.6 we report on the experimental evaluation of our method. We round up with a discussion in Section 6.7 and conclude in Section 6.8.

6.2 Related Work

Selecting or ranking interesting patterns is a well-studied topic in data mining. Existing techniques can roughly be split into two groups.

Static Approaches

The first group consists of techniques that measure how *surprising* the support of an itemset is compared against some null hypothesis: the more the observed frequency deviates from the expected value, the more interesting it is. The simplest null hypothesis is the independence model [Brin et al.,

1997, Aggarwal and Yu, 1998]. More flexible models have been suggested, for example, Bayesian Networks [Jaroszewicz and Simovici, 2004]. The major caveat of these approaches is that the null hypothesis is static and hence we keep rediscovering the same information. As a result, this will lead to pattern collections with high levels of redundancy.

Swap randomization was proposed by Gionis et al. [2007] and Hanhijärvi et al. [2009] as a way to assess the significance of data mining results by means of randomization. To this end, Gionis et al. [2007] gave an algorithm by which randomized data samples can be drawn by simply repeatedly swapping values locally, such that the background knowledge is maintained—that is, essentially a Markov chain is defined. Then, by repeatedly sampling such random datasets, one can assess the statistical significance of a result by calculating empirical p-values. While the original proposal only considered row and column margin as background knowledge, Hanhijärvi et al. [2009] extended the approach such that cluster structures and itemset frequencies can be maintained.

While a very elegant approach, swap randomization does suffer from some drawbacks. First of all, there are no theoretical results on the mixing time of the Markov chain, and hence one has to rely on heuristics (e.g., swap as many times as there are ones in the data). Second, since typically many swaps are required to obtain a randomized sample of the data, and finding suitable swaps is non-trivial, the number of randomized datasets we can realistically obtain is limited, and hence so is the p-value resolution by which we measure the significance of results. As our approach is to model the data probabilistically by the Maximum Entropy principle, we do not suffer from convergence issues, and moreover, as our model is analytical in nature, we can calculate exact probabilities and p-values.

Dynamic Approaches

The alternative approach to measuring informativeness statically, is to rank and select itemsets using a dynamic hypothesis. That is, when new knowledge arrives, e.g., in the form of a most interesting pattern, the model is updated such that we take this newly discovered information into account, and hence we avoid reporting redundant results. The method presented in this chapter falls in this category.

Besides extending the possibilities for incorporating background knowledge into a static model, the aforementioned approach by Hanhijärvi et al. [2009] further argues that by iteratively updating the randomization model, redundancy is naturally eliminated.

The MINI algorithm by Gallo et al. [2007] also uses row and column margins to rank itemsets. It first orders all potentially interesting itemsets by computing their p-value according to these margins. Then, as subsequent itemsets are added, the p-values are recomputed, and the itemsets are reordered according to their new p-values. This method, however, does not allow querying, and requires a candidate collection to be mined beforehand.

KRIMP, by Siebes et al. [2006], Vreeken et al. [2011], employs the MDL principle to select those itemsets that together compress the data best. As such, patterns that essentially describe the same part of the data are rejected. The models it finds are not probabilistic, and cannot straightforwardly be used to calculate probabilities (although Vreeken et al. [2007] showed data strongly resembling the original can be sampled from the resulting code tables). Further, while non-redundant from a compression point of view, many of the patterns it selects are variations on the same theme. The reason for this lies in the encoding scheme KRIMP utilizes: it is cheaper to encode highly specific itemsets with *one* relatively long code, than to encode it with *multiple* slightly shorter codes. Other differences to our method are that KRIMP considers its candidates in a static order, and that it is not trivial to make it consider background knowledge. A recent extension of KRIMP by Siebes and Kersten [2011] is the GROEI algorithm, which identifies the optimal set of k itemsets by beam search, instead of identifying the optimal set overall.

Modeling by Maximum Entropy

The use of maximum entropy models in pattern mining has been proposed by several authors, e.g., [Wang and Parthasarathy, 2006, Tatti and Heikinheimo, 2008, Tatti, 2008, Kontonasios and De Bie, 2010, De Bie, 2011b]. Discovering itemset collections with good BIC scores was suggested by Tatti and Heikinheimo [2008]. Alternatively, Tatti [2010] samples collections and bases the significance of an itemset on its occurrence in the discovered collections. However, in order to guarantee that the score can be computed, the authors restrict themselves to a particular type of collections: downward closed and decomposable collections of itemsets.

The method of Tatti [2008] uses local models. That is, to compute the support of an itemset X, the method only uses sub-itemsets of X, and outputs a p-value. Unlike our approach, it requires a threshold to determine whether X is important. Relatedly, Webb [2010] defines itemsets as *self-sufficient*, if their support differs significantly from what can be inferred from their sub- and supersets; therefore such a model is also local.

Wang and Parthasarathy [2006] incrementally build a maximum entropy model by adding itemsets that deviate more than a given error threshold. The approach ranks and adds itemsets in level-wise batches, i.e., first itemsets of size 1, then of size 2, and so on. This may still, however, lead to redundancy within a batch of itemsets.

De Bie [2011b] proposed an alternative to swap-randomization for obtaining randomized datasets, by modeling the whole data by maximum entropy, using row and column sums as background information; besides faster, and more well-founded, unlike swap-randomization this approach does not suffer from convergence issues. Furthermore, by its analytical nature, exact p-values can be calculated. Kontonasios and De Bie [2010] used the model to analytically define an interestingness measure, Information Ratio, for noisy tiles, by considering both the expected density of a tile, and the complexity of transferring the true tile to the user.

Although both approaches model data by the Maximum Entropy principle, there exist important differences between the two approaches. The most elementary one is that while we regard the data as a *bag of samples* from a distribution, De Bie considers it to be a *monolithic* entity. That is, De Bie models the entire binary matrix, while we construct a probabilistic model for individual rows. Roughly speaking, De Bie considers the location of a row in the matrix to be important, whereas we do not. Both these approaches have different advantages. While our approach intuitively makes more sense when modeling, say, a supermarket basket dataset, which consists of individual independent samples, the monolithic approach is more suited to model data where the rows have meaning, say, for a dataset where we have mammal presences per location. Moreover, while for our models it is straightforward to include itemset frequencies (which does not include specifying transaction identifiers), such as "*tomatoes* and *pasta* are sold in 80% of the transactions", this is currently not possible for the whole-dataset model. Additionally, while the De Bie framework in general allows the background knowledge of a user to be false, in this work we only consider background knowledge that is con-

sistent with the data. As opposed to Kontonasios and De Bie [2010], we do not just rank patterns according to interestingness, but formalize model selection techniques (specifically BIC and MDL) such that we can identify the optimal model, and hence avoid discovering overly complex models.

As overall comments to the methods described above, we note that in contrast to our approach, most of the above methods require the user to set one or several parameters, such as maximum error thresholds or significance levels. Many also cannot easily be used to estimate itemset frequencies. Further, all of them are two-phase algorithms, i.e., they require that the user provides a collection of candidate (frequent) itemsets to the algorithm, which must be completely mined and stored first, before running the actual algorithm.

6.3 Identifying the Best Summary

Our goal is to discover the collection of itemsets and frequencies $\mathcal{C}$ that is the most informative about the data $\mathcal{D}$, while also being succinct and containing as little redundancy as possible. By informative we mean that we should be able to reliably describe or predict the data using these itemsets and their frequencies. By non-redundancy we mean that every element of $\mathcal{C}$ provides significant information for describing the data that cannot be inferred from the rest of $\mathcal{C}$. This is equivalent to requiring that the frequency of an itemset $X \in \mathcal{C}$ should be surprising with respect to $\mathcal{C} \setminus X$. In other words, we do not want $\mathcal{C}$ to be unnecessarily complex as a collection, or capture spurious information. We want it to contain only those itemsets that we really need.

Informally, assume that we have a quality score $s(\mathcal{C}, \mathcal{D})$ which measures the quality of an itemset collection $\mathcal{C}$ with respect to $\mathcal{D}$. Then our aim is to find that $\mathcal{C}$ with the best score $s(\mathcal{C}, \mathcal{D})$. Analogously, if we want to restrict our attention to only k itemsets, we look for the collection $\mathcal{C}$ of size at most k, with the best score $s(\mathcal{C}, \mathcal{D})$.

Next, we will detail how we define our models, how we define this score, provide theoretical evidence why it is a good choice, and discuss how to compute it efficiently.

Example 6.1. *As a running example, assume that we have a binary dataset $\mathcal{D}$ with eight items, a to h. Furthermore, consider the set of itemsets $\mathcal{C} = \{abc, cd, def\}$ with frequencies* 0.5, 0.4 *and* 0.8, *respectively. Assume for the moment that based on $\mathcal{C}$, our method predicts that the frequency of the itemset agh is* 0.19. *Now, if we observe*

in the data that $fr(agh) = 0.18$, then we can safely say that agh is redundant with regard to what we already know, as it does not contribute a lot of novel information, and the slight deviation from the expected value may even be coincidental. On the other hand, if $fr(agh) = 0.7$, then the frequency of agh is surprising with respect to $\mathcal{C}$, and hence adding it to $\mathcal{C}$ would strongly increase the amount of information it gives us about the data; in other words $\mathcal{C} \cup \{agh\}$ provides a substantially improved description of $\mathcal{D}$.

Maximum Entropy Model

In our approach we make use of maximum entropy models. This is a class of probabilistic models that are identified by the Maximum Entropy principle [Csiszár, 1975, Jaynes, 1982]. This principle states that the best probabilistic model is the model that makes optimal use of the provided information, and that is fully unbiased (i.e., fully random, or, maximally entropic) otherwise. This property makes these models very suited for identifying informative patterns: by using maximum entropy models to measure the quality of a set of patterns, we know that our measurement only relies on the information we provide it, and that it will not be thrown off due to some spurious structure in the data. These models have a number of theoretically appealing properties, which we will discuss after a formal introduction.

Assume that we are given a collection of k itemsets and their corresponding frequencies

$$\langle \mathcal{C}, \Phi \rangle = \langle \{X_1, \ldots, X_k\}, \{f_1, \ldots, f_k\} \rangle ,$$

where $X_i \subseteq \mathcal{A}$ and $f_i \in [0,1]$, for $i = 1, \ldots, k$. Note that we do not require that the frequencies f_i of the itemsets are equal to the frequencies $fr(X_i)$ in the data. If this does hold, we will call $\langle \mathcal{C}, \Phi \rangle$ *consistent with* $\mathcal{D}$. For notational convenience, we will often omit writing the frequencies Φ, and simply use $\mathcal{C} = \{X_1, \ldots, X_k\}$, especially when it is clear from the context what the corresponding frequencies are. Now, we consider those distributions over $\mathcal{T}$ that satisfy the constraints imposed by $\langle \mathcal{C}, \Phi \rangle$. That is, we consider the following set of distributions

$$\mathcal{P}_{\langle \mathcal{C}, \Phi \rangle} = \{ p \mid p(X_i = 1) = f_i, \text{ for } i = 1, \ldots, k \} . \tag{6.1}$$

In the case that $\mathcal{P}_{\langle \mathcal{C}, \Phi \rangle}$ is empty, we call $\langle \mathcal{C}, \Phi \rangle$ *inconsistent*. Among these distributions we are interested in only one, namely the unique distribution

that maximizes the entropy

$$p^*_{\langle \mathcal{C},\Phi\rangle} = \arg\max_{p\in\mathcal{P}_{\langle \mathcal{C},\Phi\rangle}} H(p) \ .$$

Again, for notational convenience we will often simply write $p^*_{\mathcal{C}}$, or even omit $\langle \mathcal{C},\Phi\rangle$ altogether, whenever it is clear from the context.

The following famous theorem states that the maximum entropy model has an exponential form. This form will help us to discover the model and will be useful to compute the quality score of a model.

Theorem 6.1 (Theorem 3.1 in [Csiszár, 1975])**.** *Given a collection of itemsets and frequencies $\langle \mathcal{C},\Phi\rangle = \langle \{X_1,\ldots,X_k\}, \{f_1,\ldots,f_k\}\rangle$, let $\mathcal{P}_{\langle \mathcal{C},\Phi\rangle}$ be the set of distributions as defined in Eq. 6.1. If there is a distribution in $\mathcal{P}_{\langle \mathcal{C},\Phi\rangle}$ that has only nonzero entries, then the maximum entropy distribution $p^*_{\langle \mathcal{C},\Phi\rangle}$ can be written as*

$$p^*_{\langle \mathcal{C},\Phi\rangle}(\mathcal{A}=t) = u_0 \prod_{X\in\mathcal{C}} u_X^{S_X(t)} \ , \tag{6.2}$$

where all $u_X \in \mathbb{R}$, and u_0 is a normalization factor such that $p^*_{\langle \mathcal{C},\Phi\rangle}$ is a proper distribution.

Note that the normalization factor u_0 can be thought of as corresponding to the constraint that the empty itemset $\emptyset$ should have a frequency $f_\emptyset = 1$. Theorem 6.1 has the technical requirement that $\mathcal{P}$ needs to contain a distribution with non-zero entries. The easiest way to achieve this is to apply a Bayesian shift by redefining the frequencies $f'_i = (1-\epsilon)f_i + \epsilon 2^{-|X_i|}$ for some small constant $\epsilon > 0$.

Identifying the Best Model

Here we describe how we can quantify the goodness of a pattern collection.

A natural first choice is to directly measure the goodness of fit, using the log-likelihood of the maximum entropy model, that is,

$$\log p^*_{\langle \mathcal{C},\Phi\rangle}(\mathcal{D}) = \log \prod_{t\in\mathcal{D}} p^*_{\langle \mathcal{C},\Phi\rangle}(\mathcal{A}=t) = \sum_{t\in\mathcal{D}} \log p^*_{\langle \mathcal{C},\Phi\rangle}(\mathcal{A}=t) \ .$$

Note that if $\langle \mathcal{C},\Phi\rangle$ is inconsistent, p^* does not exist. In this case we define the likelihood to be zero, and hence the log-likelihood to be $-\infty$.

The following corollary shows that for exponential models we can easily calculate the log-likelihood.

Corollary 6.2 (of Theorem 6.1)**.** *The log-likelihood of the maximum entropy distribution $p^*_{\langle \mathcal{C},\Phi\rangle}$ for a set of itemsets and frequencies $\langle \mathcal{C},\Phi\rangle$ is equal to*

$$\begin{aligned}\log p^*_{\langle \mathcal{C},\Phi\rangle}(\mathcal{D}) &= |\mathcal{D}|\left(\log u_0 + \sum_{(X_i,f_i)\in\langle \mathcal{C},\Phi\rangle} f_i \log u_{X_i}\right)\\ &= -|\mathcal{D}|H\left(p^*_{\langle \mathcal{C},\Phi\rangle}\right)\,.\end{aligned}$$

Thus, to calculate the log-likelihood of a collection $\langle \mathcal{C},\Phi\rangle$, it suffices to compute the parameters u_X and u_0 of the corresponding distribution $p^*_{\langle \mathcal{C},\Phi\rangle}$.

The following theorem states that if we are searching for high likelihood collections, we can restrict ourselves to collections consistent with the data.

Theorem 6.3. *For a fixed collection of itemsets $\mathcal{C} = \{X_1,\ldots,X_k\}$, the likelihood $p^*_{\langle \mathcal{C},\Phi\rangle}(\mathcal{D})$ is maximized if and only if $\langle \mathcal{C},\Phi\rangle$ is consistent with $\mathcal{D}$, i.e., $f_i = fr(X_i)$.*

For our goal, maximum entropy models are theoretically superior over any other model. Let us discuss why. Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two datasets such that $fr(X \mid \mathcal{D}_1) = fr(X \mid \mathcal{D}_2)$ for any $X \in \mathcal{C}$. Let p^*_1 and p^*_2 be the corresponding maximum entropy models, then, by definition, $p^*_1 = p^*_2$. In other words, the model depends only on the supports of the chosen itemsets. This is a natural requirement, since we wish to measure the quality of the statistics in $\mathcal{C}$ and nothing else. Similarly, Corollary 6.2 implies that $p^*_1(\mathcal{D}_2) = p^*_1(\mathcal{D}_1)$. This is also a natural property because otherwise, the score would be depending on some statistic not included in $\mathcal{C}$. Informally said, the scores are equal if we cannot distinguish between $\mathcal{D}_1$ and $\mathcal{D}_2$ using the information $\mathcal{C}$ provides us. The next theorem states that among all such models, the maximum entropy model has the best likelihood, in other words, the maximum entropy model uses the available information as efficiently as possible.

Theorem 6.4. *Assume a collection of itemsets $\mathcal{C} = \{X_1,\ldots,X_k\}$ and let $p^*_{\mathcal{C}}$ be the maximum entropy model, computed from a given dataset $\mathcal{D}$. Assume also an alternative model $r(\mathcal{A} = t \mid f_1,\ldots,f_k)$, where $f_i = fr(X_i \mid \mathcal{D})$, that is, a statistical model parametrized by the frequencies of $\mathcal{C}$. Assume that for any two datasets $\mathcal{D}_1$ and $\mathcal{D}_2$, where $fr(X \mid \mathcal{D}_1) = fr(X \mid \mathcal{D}_2)$ for any $X \in \mathcal{C}$, it holds that*

$$1/|\mathcal{D}_1| \log r(\mathcal{D}_1 \mid f_1,\ldots,f_k) = 1/|\mathcal{D}_2| \log r(\mathcal{D}_2 \mid f_1,\ldots,f_k)\,.$$

Then $p^*_{\mathcal{C}}(\mathcal{D}) \geq r(\mathcal{D})$ for any dataset $\mathcal{D}$.

Proof. There exists a sequence of finite datasets D_j such that $fr(X_i \mid D_j) = f_i$ and $q_{D_j} \to p^*_{\mathcal{C}}$. To see this, first note that the f_i are rational numbers so that the set of distributions $\mathcal{P}_{\mathcal{C}}$ is a polytope with faces defined by rational equations. In other words, there is a sequence of distributions p_j with only rational entries reaching p^*. Since a distribution with rational entries can be represented by a finite dataset, we can have a sequence D_j such that $q_{D_j} = p_j$.

Now as j goes to infinity, we have

$$\begin{aligned}\frac{1}{|D|}\log r(D) &= \frac{1}{|D_j|}\log r(D_j) \\ &= \sum_{t\in\mathcal{T}} q_{D_j}(\mathcal{A}=t)\log r(\mathcal{A}=t) \to \sum_{t\in\mathcal{T}} p^*_{\mathcal{C}}(\mathcal{A}=t)\log r(\mathcal{A}=t)\ .\end{aligned}$$

Since the left side does not depend on j we actually have a constant sequence. Hence,

$$\begin{aligned}0 \le KL(p^*_{\mathcal{C}} \,\|\, r) &= H(p^*_{\mathcal{C}}) - \sum_{t\in\mathcal{T}} p^*_{\mathcal{C}}(\mathcal{A}=t)\log r(\mathcal{A}=t) \\ &= 1/|D|(\log p^*_{\mathcal{C}}(D) - \log r(D))\ ,\end{aligned}$$

which proves the theorem. □

Using only log-likelihood to evaluate a model, however, suffers from overfitting: larger collections of itemsets will always provide more information, hence allow for better estimates, and therefore have a better log-likelihood. Consequently, we need to prevent our method from overfitting. In order to do so, we will explore the Bayesian Information Criterion (BIC), and the Minimum Description Length (MDL) principle—both of which are well-known and well-founded model selection techniques (see Section 2.3). We start by discussing BIC, which is the least strict, and least involved of the two.

The Bayesian Information Criterion (BIC) measures the quality of a model by taking both its log-likelihood, and the number of parameters of said model into account. It favors models that fit the data well using few parameters; in our case, the number of parameters of the model p^* corresponds exactly to the number of itemsets k. It has a strong theoretical support in Bayesian model selection [Schwarz, 1978]. The BIC score of a collection $\mathcal{C}$ is defined as

$$\text{BIC}(\langle \mathcal{C}, \Phi\rangle, \mathcal{D}) = -\log p^*_{\langle \mathcal{C},\Phi\rangle}(\mathcal{D}) + k/2 \log |\mathcal{D}|\ .$$

The better a model fits the data, the higher its likelihood. On the other hand, the more parameters the model has—i.e., the more complex it is—the higher its penalty. Therefore, it is possible that a model that fits the data slightly worse, but contains few parameters, is favored by BIC over a model that fits the data better, but is also more complex. From Corollary 6.2 we see that the first term of the BIC score is equal to $|\mathcal{D}|H(p^*_{\langle \mathcal{C},\Phi\rangle})$. Hence, the likelihood term grows faster than the penalty term with respect to the size of $\mathcal{D}$. As such, the more data (or evidence) we have available, the more complicated the model is allowed to be in order to fit the data well.

Corollary 6.5 (of Theorem 6.3)**.** *For a fixed itemset collection $\mathcal{C} = \{X_1, \ldots, X_k\}$, the* BIC *score* $\text{BIC}(\langle \mathcal{C}, \Phi\rangle, \mathcal{D})$ *is maximized if and only if $\langle \mathcal{C}, \Phi\rangle$ is consistent with $\mathcal{D}$, that is, $f_i = fr(X_i)$ for all $i = 1, \ldots, k$.*

Proof. Follows directly from Theorem 6.3 and the fact that the BIC penalty term, $k/2 \log |\mathcal{D}|$, does not depend on the frequencies f_i. □

While the BIC score helps to avoid overfitting, it is somewhat simplistic. That is, it only incorporates the *number* of itemsets to penalize a summary, and not their complexity. As stated in the introduction, if possible, we would typically rather be given some number of general patterns than the same number of highly involved patterns. The MDL principle provides us with the means to define a score that also takes into account the complexity of the itemsets in $\mathcal{C}$. Recall that according to the MDL principle, given a dataset $\mathcal{D}$ and a set of models $\mathcal{M}$ for $\mathcal{D}$, the best model $M \in \mathcal{M}$ is the one that minimizes

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

in which

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

To use MDL, we have to define what our set of models $\mathcal{M}$ is, how a model M describes a database, and how all of this is encoded in bits. Intuitively, we want to favor itemset collections that are small, i.e., collections which can describe the data well, using few itemsets. At the same time, we also prefer collections with small itemsets over collections with large ones.

Definition 6.1. *Given an itemset collection* $\langle \mathcal{C}, \Phi \rangle = \langle \{X_1, \ldots, X_k\}, \{f_1, \ldots, f_k\} \rangle$, *let* $x = \sum_{i=1}^{k} |X_i|$. *We define the* MDL *score of* $\langle \mathcal{C}, \Phi \rangle$ *with respect to a dataset* $\mathcal{D}$ *as*

$$\text{MDL}(\langle \mathcal{C}, \Phi \rangle, \mathcal{D}) = L(\mathcal{D} \mid \langle \mathcal{C}, \Phi \rangle) + L(\langle \mathcal{C}, \Phi \rangle) \,,$$

where

$$L(\mathcal{D} \mid \langle \mathcal{C}, \Phi \rangle) = -\log p^*_{\langle \mathcal{C}, \Phi \rangle}(\mathcal{D}) \quad \textit{and} \quad L(\langle \mathcal{C}, \Phi \rangle) = l_1 k + l_2 x + 1 \,,$$

with

$$l_1 = \log |\mathcal{D}| + N \log(1 + N^{-1}) + 1 \approx \log |\mathcal{D}| + \log e + 1$$

and

$$l_2 = \log N \,.$$

Whenever $\mathcal{D}$ *is clear form the context, we simply write* $\text{MDL}(\langle \mathcal{C}, \Phi \rangle)$.

The first term is simply the negative log-likelihood of the model, which corresponds to the description length of the data given the maximum entropy model induced by $\mathcal{C}$. The second part is a penalty term, which corresponds to the description length of the model. It is a linear function of $k = |\mathcal{C}|$ and $x = \sum_i |X_i|$, of which the coefficients depend on N and $|\mathcal{D}|$. How it is derived is explained further below. The smaller this score, the better the model. Given two collections with an equal amount of itemsets, the one containing fewer items is penalized less; conversely, if they have the same total number of items, the one that contains those items in fewer itemsets is favored. Consequently, the best model is identified as the model that provides a good balance between high likelihood and low complexity. Moreover, we automatically avoid redundancy, since models with redundant itemsets are penalized for being too complex, without sufficiently improving the likelihood.

With this quality score we evaluate collections of itemsets, rather than the (maximum entropy) distributions we construct from them. The reason for this is that we want to summarize the data with a succinct set of itemsets, not model it with a distribution. A single distribution, after all, may be described by many different collections of itemsets, simple or complex. Further, we assume that the set of models $\mathcal{M}$ consists of collections of itemsets which are represented as vectors, rather than as sets. This choice keeps the quality score function computationally simple and intuitive, and is not disadvantageous: if $\mathcal{C}$ contains duplicates, they simply increase the penalty term.

Additionally, we impose no restrictions on the consistency of $\langle \mathcal{C}, \Phi \rangle$, i.e., there are collections for which $\mathcal{P}_{\langle \mathcal{C}, \Phi \rangle}$ is empty, and hence the maximum entropy distribution does not exist. As mentioned above, in this case we define the likelihood to be zero, and hence the description length is infinite.

We now describe the derivation of the penalty term, which equals

$$k \log |\mathcal{D}| + x \log N + (k+1) + kN \log(1 + N^{-1}) \; .$$

To describe an itemset we encode a support using $\log |D|$ bits and the actual items in the itemsets using $\log N$ bits each. This gives us the first two terms. We use the third term to express whether there are more itemsets, one bit after each itemset, and one extra bit to accommodate the case $\mathcal{C} = \emptyset$. The term $kN \log(1 + N^{-1})$ is a normalization factor, to ensure that the encoding is *optimal* for the prior distribution over all pattern collections. That is, the encoding corresponds to a distribution

$$\mathit{prior}(\langle \mathcal{C}, \Phi \rangle) = 2^{-l_1 k - l_2 x - 1} \; ,$$

which assigns high probability to simple summaries, and low probability to complex ones. The following equation shows that the above encoding is optimal with respect to this prior.

$$\begin{aligned}
\sum_{\langle \mathcal{C}, \Phi \rangle} \mathit{prior}(\langle \mathcal{C}, \Phi \rangle) &= \sum_{k=0}^{\infty} \sum_{x=0}^{kN} \binom{kN}{x} |\mathcal{D}|^k 2^{-l_1 k - l_2 x - 1} \\
&= \sum_{k=0}^{\infty} 2^{-k-1} 2^{-kN \log(1+N^{-1})} 2^{-k \log |D|} |\mathcal{D}|^k \sum_{x=0}^{kN} \binom{kN}{x} N^{-x} \\
&= \sum_{k=0}^{\infty} 2^{-k-1} 2^{-kN \log(1+N^{-1})} (1 + N^{-1})^{kN} \\
&= \sum_{k=0}^{\infty} 2^{-k-1} = 1
\end{aligned}$$

The corollary below shows that for identifying the MDL optimal model, it suffices to only consider summaries that are consistent with the data.

Corollary 6.6 (of Theorem 6.3)**.** *For a fixed itemset collection $\mathcal{C} = \{X_1, \ldots, X_k\}$, the* MDL *score* $\textsc{mdl}(\langle \mathcal{C}, \Phi \rangle, \mathcal{D})$ *is maximized if and only if $\langle \mathcal{C}, \Phi \rangle$ is consistent with $\mathcal{D}$, that is, $f_i = fr(X_i)$ for all $i = 1, \ldots, k$.*

Proof. Follows directly from Theorem 6.3 and the fact that for a fixed $\mathcal{C}$, the frequencies f_i are encoded with a constant length $\log|\mathcal{D}|$, and hence the penalty term is always the same. □

Therefore, in the remainder of this chapter we will assume that $\langle \mathcal{C}, \Phi \rangle$ is always consistent with $\mathcal{D}$, and hence we will omit Φ from notation.

Reducing Redundancy

Here we show that our score favors itemset collections with low redundancy, and make a theoretical link with some popular lossless redundancy reduction techniques from the pattern mining literature. Informally, we define redundancy as anything that does not deviate (much) from our expectation, or in other words is unsurprising given the information that we already have. The results below hold for BIC as well as for MDL, and hence we write s to denote either one of these scores.

A baseline technique for ranking itemsets is to compare the observed frequency against the expected value of some null hypothesis. The next theorem shows that if the observed frequency of an itemset X agrees with the expected value $p^*(X = 1)$, then X is redundant.

Theorem 6.7. *Let $\mathcal{C}$ be a collection of itemsets and let p^* be the corresponding maximum entropy model. Let $X \notin \mathcal{C}$ be an itemset such that $fr(X) = p^*(X = 1)$. Then $s(\mathcal{C} \cup \{X\}, \mathcal{D}) > s(\mathcal{C}, \mathcal{D})$.*

Proof. We will prove the theorem by showing that the likelihood terms for both collections are equal. Define the collection $\mathcal{C}_1 = \mathcal{C} \cup \{X\}$ and let $\mathcal{P}_1$ be the corresponding set of distributions. Let p_1^* be the distribution maximizing the entropy in $\mathcal{P}_1$. Note that since $\mathcal{C} \subset \mathcal{C}_1$, we have $\mathcal{P}_1 \subseteq \mathcal{P}$ and hence $H(p_1^*) \leq H(p^*)$. On the other hand, the assumption in the theorem implies that $p^* \in \mathcal{P}_1$ and so $H(p^*) \leq H(p_1^*)$. Thus, $H(p^*) = H(p_1^*)$ and since the distribution maximizing the entropy is unique, we have $p^* = p_1^*$. This shows that the likelihood terms in $s(\mathcal{C}, \mathcal{D})$ and $s(\mathcal{C}_1, \mathcal{D})$ are equal. The penalty term of the latter is larger, which concludes the proof. □

Theorem 6.7 states that adding an itemset X to $\mathcal{C}$ improves the score only if its observed frequency deviates from the expected value. The amount of deviation required to lower the score, is determined by the penalty term.

This gives us a convenient advantage over methods that are based solely on deviation, since they require a user-specified threshold.

Two useful corollaries follow from Theorem 6.7, which connect our approach to well-known techniques for removing redundancy from pattern set collections—so-called *condensed representations*. (See also Section 2.1.) The first corollary relates our approach to *closed* itemsets [Pasquier et al., 1999], and *generator* itemsets (also known as *free* itemsets [Boulicaut et al., 2003]). An itemset is closed if all of its supersets have a strictly lower support. An itemset is a generator if all of its subsets have a strictly higher support.

Corollary 6.8 (of Theorem 6.7)**.** *Let $\mathcal{C}$ be a collection of itemsets. Assume that $X, Y \in \mathcal{C}$ such that $X \subset Y$ and $fr(X) = fr(Y) \neq 0$. Assume that $Z \notin \mathcal{C}$ such that $X \subset Z \subset Y$. Then $s(\mathcal{C} \cup \{Z\}, \mathcal{D}) > s(\mathcal{C}, \mathcal{D})$.*

Proof. Let $p \in \mathcal{P}$, as defined in Eq. 6.1. We have that $p(X = 1) = fr(X) = fr(Y) = p(Y = 1)$. Hence we must have $p(Z = 1) = fr(Z)$. Since $p^* \in \mathcal{P}$, it must hold that $p^*(Z = 1) = fr(Z)$. The result follows from Theorem 6.7. □

Corollary 6.8 implies that if the closure and a generator of an itemset Z are already in the collection, then adding Z will worsen the score. The second corollary provides a similar relation with *non-derivable* itemsets [Calders and Goethals, 2007]. An itemset is called derivable if its support can be inferred exactly given the supports of all of its proper subsets.

Corollary 6.9 (of Theorem 6.7)**.** *Let $\mathcal{C}$ be a collection of itemsets. Assume that $X \notin \mathcal{C}$ is a derivable itemset and all sub-itemsets of X are included in $\mathcal{C}$. Then $s(\mathcal{C} \cup \{X\}, \mathcal{D}) > s(\mathcal{C}, \mathcal{D})$.*

Proof. The proof of this corollary is similar to the proof of Corollary 6.8. □

An advantage of our method is that it can avoid redundancy in a very general way. The closed and non-derivable itemsets are two types of lossless representations, whereas our method additionally can give us lossy redundancy removal. For example, in Corollary 6.8, $fr(X)$ does not have to equal $fr(Y)$ exactly in order to reject Z from $\mathcal{C}$. This allows us to prune redundancy in a more aggressive way.

Algorithm 6.1: IterativeScaling

input : an itemset collection $\mathcal{C} = \{X_1, \ldots, X_k\}$; corresponding frequencies $fr(X_1), \ldots, fr(X_k)$
output: the parameters u_X and u_0 of the maximum entropy distribution $p^*_{\mathcal{C}}$ satisfying $p^*_{\mathcal{C}}(X_i) = fr(X_i)$ for all i

```
1 initialize p
2 while p has not converged do
3     for each X in C do
4         compute p(X = 1)
5         u_X ← u_X · fr(X)/p(X=1) · (1−p(X=1))/(1−fr(X))
6         u_0 ← u_0 · (1−fr(X))/(1−p(X=1))
7     end
8 end
9 return p
```

Efficiently Computing the Maximum Entropy Model

Computing the maximum entropy model comes down to finding the u_X and u_0 parameters from Theorem 6.1. To achieve this, we use the well-known Iterative Scaling procedure by Darroch and Ratcliff [1972], which is given here as Algorithm 6.1. Simply put, it iteratively updates the parameters of an exponential distribution, until it converges to the maximum entropy distribution p^* which satisfies a given set of constraints—itemset frequencies in our case. The distribution is initialized with the uniform distribution, which is done by setting the u_X parameters to 1, and $u_0 = 2^{-N}$ to properly normalize the distribution. Then, for each itemset $X \in \mathcal{C}$, we adjust the corresponding parameter u_X to enforce $p(X = 1) = fr(X)$ (line 5,6). This process is repeated in a round robin fashion until p converges, and it can be shown (see Darroch and Ratcliff [1972]) that p always converges to the maximum entropy distribution p^*. Typically the number of iterations required for convergence is low (usually <10 in our experiments).

Example 6.2. *In our running example, with $\mathcal{C} = \{abc, cd, def\}$, the maximum entropy model has three parameters u_1, u_2, u_3, and a normalization factor u_0. Initially we set $u_1 = u_2 = u_3 = 1$ and $u_0 = 2^{-N} = 2^{-8}$. Then we iteratively loop over the itemsets and scale the parameters. For instance, for the first itemset abc with*

frequency 0.5, *we first compute its current estimate to be* $2^{-3} = 0.125$. *Thus, we update the first parameter* $u_1 = 1 \cdot (0.5/2^{-3}) \cdot ((1-2^{-3})/0.5) = 7$. *The normalization factor becomes* $u_0 = 2^{-8} \cdot 0.5/(1-2^{-3}) \approx 2.2 \cdot 10^{-3}$. *Next, we do the same for itemset cd, and so on. After a few iterations, the model parameters converge to* $u_1 = 28.5$, $u_2 = 0.12$, $u_3 = 85.4$ *and* $u_0 = 3 \cdot 10^{-4}$.

Straightforward as this procedure may seem, the greatest computational bottleneck is the inference of the probability of an itemset, on line 4 of the algorithm,

$$p(X = 1) = \sum_{\substack{t \in \mathcal{T} \\ S_X(t)=1}} p(\mathcal{A} = t) \; . \tag{6.3}$$

Since this sum ranges over all possible transactions supporting X, it is infeasible to calculate by brute force, even for a moderate number of items N. In fact, it has been shown that querying the maximum entropy model is **PP**-hard in general [Tatti, 2006a].

Therefore, in order to be able to query the model efficiently, we introduce a partitioning scheme, which makes use of the observation that many transactions have the same probability in the maximum entropy distribution. Remark that an itemset collection $\mathcal{C}$ partitions $\mathcal{T}$ into blocks of transactions which support the same set of itemsets. That is, two transactions t_1 and t_2 belong to the same block T if and only if $S_X(t_1) = S_X(t_2)$ for all X in $\mathcal{C}$. Therefore, we know that $p(\mathcal{A} = t_1) = p(\mathcal{A} = t_2)$ if p is of the form in Eq. 6.2. This property allows us to define $S_X(T) = S_X(t)$ for any $t \in T$ and $X \in \mathcal{C}$. We denote the partition of $\mathcal{T}$ induced by $\mathcal{C}$ as $\mathcal{T}_\mathcal{C}$. We can now compute the probability of an itemset as

$$p(X = 1) = \sum_{\substack{T \in \mathcal{T}_\mathcal{C} \\ S_X(T)=1}} p(\mathcal{A} \in T) \; .$$

The sum in Eq. 6.3 has been reduced to a sum over blocks of transactions, and the inference problem has been moved from the transaction space $\mathcal{T}$ to the block space $\mathcal{T}_\mathcal{C}$. In our setting we will see that $|\mathcal{T}_\mathcal{C}| \ll |\mathcal{T}|$, which makes inference a lot more feasible. In the worst case, this partition may contain $2^{|\mathcal{C}|}$ blocks, however, through the interplay of the itemsets, it can be as low as $|\mathcal{C}| + 1$. As explained further on, we can exploit or even choose to limit the structure of $\mathcal{C}$, such that practical computation is guaranteed.

Algorithm 6.2: ComputeBlockSizes

```
input : an itemset collection C = {X_1, ..., X_k}
output: the size e(T) for each transaction block T in T_C
for T in T_C do
    I ← ⋃{X | X ∈ sets(T; C)}
    c(T) ← 2^{N−|I|}
end
sort the blocks in T_C
for T_i in T_C do
    e(T_i) ← c(T_i)
    for T_j in T_C, with j < i do
        if T_i ⊂ T_j then
            e(T_i) ← e(T_i) − e(T_j)
        end
    end
end
return T_C
```

All we must do now is obtain the block probabilities $p(\mathcal{A} \in T)$. Since all transactions t in a block T have the same probability

$$p(\mathcal{A} = t) = u_0 \prod_{X \in \mathcal{C}} u_X^{S_X(t)} ,$$

it suffices to compute the number of transactions in T to get $p(\mathcal{A} \in T)$. So, let us define $e(T)$ to be the number of transactions in T, then

$$p(\mathcal{A} \in T) = \sum_{t \in T} p(\mathcal{A} = t) = e(T)\, u_0 \prod_{X \in \mathcal{C}} u_X^{S_X(T)} .$$

Algorithm 6.2 describes ComputeBlockSizes, a basic method to compute the block sizes $e(T)$. To this end, we introduce a partial order on $\mathcal{T}_\mathcal{C}$. Let

$$sets(T; \mathcal{C}) = \{X \in \mathcal{C} \mid S_X(T) = 1\}$$

be the itemsets of $\mathcal{C}$ that occur in the transactions of T. Note that every block corresponds to a unique subset of $\mathcal{C}$; conversely a subset of $\mathcal{C}$ either

corresponds to an empty block of transactions, or to a unique nonempty transaction block. We can now define the partial order on $\mathcal{T}_\mathcal{C}$ as follows,

$$T_1 \subseteq T_2 \quad \text{if and only if} \quad \mathit{sets}(T_1;\mathcal{C}) \subseteq \mathit{sets}(T_2;\mathcal{C}) \ .$$

In order to compute the size $e(T)$ of a block, we start from its *cumulative* size,

$$c(T) = \sum_{T' \supseteq T} e(T') \ ,$$

which is the number of transactions that contain *at least* all the itemsets in $\mathit{sets}(T;\mathcal{C})$. For a given block T, let $I = \bigcup\{X \mid X \in \mathit{sets}(T;\mathcal{C})\}$. That is, I are the items that occur in all transactions of T. Then it holds that $c(T) = 2^{N-|I|}$, where N is the total number of items. To obtain the block sizes $e(T)$ from the cumulative sizes $c(T)$, we use the Inclusion-Exclusion principle. To that end, the blocks are topologically sorted such that if $T_2 \subset T_1$, then T_1 occurs before T_2. The algorithm then reversely iterates over the blocks in a double loop, subtracting block sizes, using the identity

$$e(T) = c(T) - \sum_{T' \supsetneq T} e(T') \ .$$

Example 6.3. *Assume again that we have a dataset with eight items (a to h), and an itemset collection containing three itemsets* $\mathcal{C} = \{abc, cd, def\}$ *with frequencies* 0.5, 0.4 *and* 0.8, *respectively.*

Table 6.1 shows the sizes of the transaction blocks. Note that while there are 256 transactions in $\mathcal{T}$, *there are only 7 blocks in* $\mathcal{T}_\mathcal{C}$, *whose sizes and probabilities are to be computed; the eighth combination 'abc and def but not cd' is clearly impossible.*

Let us compute the sizes of the first three blocks. For the first block, $I = abcdef$ *and therefore* $c(T) = 4$, *for the second block* $I = abcd$, *and for the third block* $I = abc$. *Since the first block is maximal with respect to the order* $\subseteq$, *its cumulative size is simply its size, so* $e(T) = 4$. *For the second block, we subtract the first block, and obtain* $e(T) = 16 - 4 = 12$. *From the third block we subtract the first two blocks, and we have* $e(T) = 32 - 12 - 4 = 16$. *Now, to compute, say,* $p(abc = 1)$, *we simply need the sizes of the blocks containing abc, and the model parameters,*

$$p(abc = 1) = 4(u_0u_1u_2u_3) + 12(u_0u_1u_2) + 16(u_0u_1) \ .$$

Table 6.1: Transaction blocks for the running example above, with $X_1 = abc$, $X_2 = cd$, and $X_3 = def$.

X_1	X_2	X_3	$c(T)$	$e(T)$	$p(\mathcal{A} = t)$
1	1	1	4	4	$u_0u_1u_2u_3$
1	1	0	16	12	$u_0u_1u_2$
1	0	0	32	16	u_0u_1
0	1	1	16	12	$u_0u_2u_3$
0	1	0	64	36	u_0u_2
0	0	1	32	16	u_0u_3
0	0	0	256	160	u_0

Since the algorithm performs a double loop over all transaction blocks, the complexity of ComputeBlockSizes equals $O(|\mathcal{T}_\mathcal{C}|^2)$. Note that topologically sorting the blocks (line 5) takes $O(|\mathcal{T}_\mathcal{C}| \log |\mathcal{T}_\mathcal{C}|) \leq O(|\mathcal{T}_\mathcal{C}|k)$, however, we can also simply ensure that the blocks are topologically sorted by construction.

Next, we show that we can substantially improve upon the ComputeBlockSizes algorithm, by using a generalized version of the Quick Inclusion-Exclusion (qie) algorithm, introduced by Calders and Goethals [2005]. The new algorithm presented here, called QieBlockSizes, has a lower complexity than ComputeBlockSizes. The idea behind Quick Inclusion-Exclusion is to reuse intermediate results to reduce the number of subtractions. The standard qie algorithm computes the supports of all generalized itemsets based on some given itemset of size k (recall that a generalized itemset is an itemset containing both positive and negative items, e.g., $a\bar{b}$ means a and not b), using the supports of all of its (positive) subsets. For instance, from the supports of ab, a, b, and the empty set, we can infer the support of $\overline{ab}$. qie therefore works on an array of size 2^k, which allows an implementation of the algorithm to employ very efficient array indexing using integers, and makes it very easy to locate subsets using bit operations on the indices—where a positive item is represented by a 1 and a negative item by a 0.

In our setting, we want to find the sizes of transaction blocks which correspond to subsets of $\mathcal{C}$, starting from the cumulative sizes of said blocks. We can represent each block T by a binary vector defined by the indicator functions S_X. However, an important difference with the qie algorithm is that not every possible binary vector necessarily corresponds to a (nonempty) block

of transactions, i.e., it is possible that $|\mathcal{T}_\mathcal{C}| < 2^k$. Clearly, if $|\mathcal{T}_\mathcal{C}| \ll 2^k$, it would be inefficient to use an array of size 2^k. Therefore, we must take these gaps into account. Before we can discuss the algorithm itself, we first need to introduce the following definitions.

Definition 6.2. *Given a collection of itemsets $\mathcal{C} = \{X_1, \ldots, X_k\}$ and an integer $j \in \{0, \ldots, k\}$, the j-*prefix *of $\mathcal{C}$ is defined as*

$$\mathcal{C}_j = \{X_1, \ldots, X_j\} .$$

For a subset $\mathcal{G}$ of $\mathcal{C}$, we define the closure $: 2^\mathcal{C} \to 2^\mathcal{C}$ as

$$\mathit{closure}(\mathcal{G}) = \{X_i \in \mathcal{C} \mid X_i \subseteq \bigcup_{X \in \mathcal{G}} X\} .$$

*The j-*closure *of $\mathcal{G}$ is defined as*

$$\begin{aligned} \mathit{closure}(\mathcal{G}, j) &= \mathcal{G} \cup \{X_i \in \mathcal{C} \mid X_i \notin \mathcal{C}_j \text{ and } X_i \subseteq \bigcup_{X \in \mathcal{G}} X\} \\ &= \mathcal{G} \cup \big(\mathit{closure}(\mathcal{G}) \setminus \mathcal{C}_j\big) \ . \end{aligned}$$

The following lemma states that there is a one-to-one mapping between the closed subsets $\mathcal{G}$ of $\mathcal{C}$, and the transaction blocks of $\mathcal{T}_\mathcal{C}$.

Lemma 6.10. *Let $\mathcal{G}$ be an itemset collection. Then $\mathcal{G} = \mathit{closure}(\mathcal{G})$ if and only if there exists a block T in $\mathcal{T}_\mathcal{C}$ such that $\mathcal{G} = \mathit{sets}(T; \mathcal{C})$.*

Proof. Assume that $\mathcal{G} = \mathit{closure}(\mathcal{G})$. Let $U = \bigcup_{X \in \mathcal{G}} X$ and let $t \in \mathcal{T}$ be such that $t_i = 1$ if $a_i \in U$, and $t_i = 0$ otherwise. Let $T \in \mathcal{T}_\mathcal{C}$ be the block containing t. If $X \in \mathcal{G}$, then $S_X(t) = 1$. On the other hand, if $S_X(t) = 1$, then $X \subseteq U$ and consequently $X \in \mathit{closure}(\mathcal{G}) = \mathcal{G}$. Hence, $\mathcal{G} = \mathit{sets}(T; \mathcal{C})$.

Assume now that there is a T such that $\mathcal{G} = \mathit{sets}(T; \mathcal{C})$, let $t \in T$ and $U = \bigcup_{X \in \mathcal{G}} X$. It follows that $S_U(t) = 1$. Let $X \in \mathit{closure}(\mathcal{G})$, then $X \subseteq U$ and $S_X(t) = 1$. Hence, $X \in \mathit{sets}(T; \mathcal{C}) = \mathcal{G}$. Since $\mathcal{G} \subseteq \mathit{closure}(\mathcal{G})$, the lemma follows. □

Using the above lemma, we can introduce the following function, which maps subsets of $\mathcal{C}$ to their corresponding blocks.

Algorithm 6.3: QieBlockSizes

input : an itemset collection $\mathcal{C} = \{X_1, \ldots, X_k\}$
output: the size $e(T)$ for each transaction block T in $\mathcal{T}_\mathcal{C}$

```
1  for T in T_C do
2  |  I ← ⋃{X | X ∈ sets(T; C)}
3  |  c(T) ← 2^{N−|I|}
4  end
5  for i = 1, ..., k do
6  |  for each T in T_C do
7  |  |  G ← sets(T; C)
8  |  |  if X_i ∉ G then
9  |  |  |  G' ← closure(G ∪ {X_i}, i − 1)
10 |  |  |  T' ← block(G')
11 |  |  |  if T' ≠ ∅ then
12 |  |  |  |  e(T) ← e(T) − e(T')
13 |  |  |  end
14 |  |  end
15 |  end
16 end
```

Definition 6.3. *For a subset $\mathcal{G}$ of a collection of itemsets $\mathcal{C}$, we define*

$$block(\mathcal{G}) = \begin{cases} T \in \mathcal{T}_\mathcal{C} \text{ s.t. } sets(T;\mathcal{C}) = \mathcal{G} & \text{if } closure(\mathcal{G}) = \mathcal{G}\ , \\ \emptyset & \text{otherwise}\ . \end{cases}$$

That is, if $\mathcal{G}$ is closed, the block function simply maps it to the corresponding block in $\mathcal{T}_\mathcal{C}$. If $\mathcal{G}$ is not closed, it is mapped to the empty transaction block. Note that $block(sets(T;\mathcal{C})) = T$ for all $T \in \mathcal{T}_\mathcal{C}$.

QieBlockSizes is given as Algorithm 6.3. As before, we first compute the cumulative probability of every block T in $\mathcal{T}_\mathcal{C}$ (line 3). Then, for each itemset X_i (line 5), the algorithm subtracts from each block T for which $X_i \notin \mathcal{G} = sets(T;\mathcal{C})$, the current size of the block T' corresponding to $\mathcal{G}' = closure(\mathcal{G} \cup \{X_i\}, i-1)$ if T' exists in $\mathcal{T}_\mathcal{C}$, i.e., the size of $T' = block(\mathcal{G}')$.

Below we prove that QieBlockSizes correctly computes the sizes of all blocks of transactions in $\mathcal{T}_\mathcal{C}$. First, we state two lemmas, which are then used in the proof of the subsequent theorem.

Lemma 6.11. *Let $\mathcal{G}$ and $\mathcal{H}$ be itemset collections such that $\mathcal{G} \subseteq \mathcal{H} \subseteq closure(\mathcal{G})$, then $closure(\mathcal{H}) = closure(\mathcal{G})$.*

Proof. Write $\mathcal{F} = closure(\mathcal{G})$ and let $U = \bigcup_{X \in \mathcal{G}} X$, $V = \bigcup_{X \in \mathcal{H}} X$, and $W = \bigcup_{X \in \mathcal{F}} X$. By definition we have $U \subseteq V$ which implies that $closure(\mathcal{G}) \subseteq closure(\mathcal{H})$. Also since $V \subseteq W$, we have $closure(\mathcal{H}) \subseteq closure(\mathcal{F}) = closure(\mathcal{G})$, where the second equality follows from the idempotency of closure. □

Lemma 6.12. *Let $\mathcal{G}$ be an itemset collection and let $Y \notin \mathcal{G}$ be an itemset. Assume that there is a transaction $t \in \mathcal{T}$ such that $S_X(t) = 1$ for every $X \in \mathcal{G}$ and $S_Y(t) = 0$. Then $Y \notin closure(\mathcal{G})$.*

Proof. $S_Y(t) = 0$ implies that there is $a_i \in Y$ such that $t_i = 0$. Note that $a_i \notin X$ for any $X \in \mathcal{G}$, otherwise $S_X(t) = 0$. This implies that $Y \nsubseteq \bigcup_{X \in \mathcal{G}} X$ which proves the lemma. □

With the lemmas above, we can now prove the main theorem.

Theorem 6.13. *Given a collection of itemsets $\mathcal{C} = \{X_1, \ldots, X_k\}$, let $\mathcal{T}_\mathcal{C}$ be the corresponding partition with respect to $\mathcal{C}$. The algorithm* QieBlockSizes *correctly computes the block sizes $e(T)$ for all $T \in \mathcal{T}_\mathcal{C}$.*

Proof. Let us denote $e^0(T) = c(T)$ for the initialized value, and let $e^i(T)$ be the value of $e(T)$ after the execution of the i-th iteration of QieBlockSizes for $i = 1, \ldots, k$.

Let us write

$$S^i(\mathcal{G}) = \{t \in \mathcal{T} \mid S_X(t) = 1 \text{ for } X \in \mathcal{G} \text{ and } S_X(t) = 0 \text{ for } X \in \mathcal{C}_i \setminus \mathcal{G}\} \ .$$

The following properties hold for S^i:

1. $S^i(\mathcal{G}) \subseteq S^{i-1}(\mathcal{G})$ for any $\mathcal{G}$ and $i > 0$.
2. If $X_i \notin \mathcal{G}$, then $S^{i-1}(\mathcal{G}) = S^i(\mathcal{G}) \cup S^{i-1}(\mathcal{G} \cup \{X_i\})$, and $S^{i-1}(\mathcal{G} \cup \{X_i\}) \cap S^i(\mathcal{G}) = \emptyset$.
3. $S^i(\mathcal{G}) = S^i(closure(\mathcal{G}, i))$ for any $\mathcal{G}$.
4. If $X_i \in \mathcal{G}$, then $S^i(\mathcal{G}) = S^{i-1}(\mathcal{G})$.

Note that $\left|S^k(sets(T;\mathcal{C}))\right| = e(T)$, hence to prove the theorem we will show by induction that $e^i(T) = \left|S^i(sets(T;\mathcal{C}))\right|$ for $i = 0,\ldots,k$ and for $T \in \mathcal{T}_\mathcal{C}$.

For $i = 0$ the statement clearly holds. Let $i > 0$ and make an induction assumption that $e^{i-1}(T) = \left|S^{i-1}(sets(T;\mathcal{C}))\right|$. If $X_i \in \mathcal{G}$, then by definition $e^i(T) = e^{i-1}(T)$. Property 4 now implies that $S^i(\mathcal{G}) = S^{i-1}(\mathcal{G})$, proving the induction step.

Otherwise, assume that $X_i \notin \mathcal{G}$, let us define $\mathcal{F} = \mathcal{G} \cup \{X_i\}$ and let $\mathcal{G}' = closure(\mathcal{F}, i-1)$ and $\mathcal{H} = closure(\mathcal{G}')$. Since it holds that $\mathcal{F} \subseteq \mathcal{G}' \subseteq closure(\mathcal{F})$, Lemma 6.11 implies that $\mathcal{H} = closure(\mathcal{F})$.

Assume that $T' = block(\mathcal{G}') = \emptyset$. Then we have $e^i(T) = e^{i-1}(T)$, hence we need to show that $S^i(\mathcal{G}) = S^{i-1}(\mathcal{G})$. Assume otherwise. We will now show that $\mathcal{G}' = \mathcal{H}$ and apply Lemma 6.10 to conclude that $block(\mathcal{G}') \neq \emptyset$, which contradicts our assumption.

First note that if there would exist an $X \in \mathcal{H} \setminus \mathcal{G}'$, then $X \in \mathcal{C}_{i-1}$, since

$$\begin{aligned}
\mathcal{H} \setminus \mathcal{G}' &= \mathcal{H} \setminus (\mathcal{F} \cup (closure(\mathcal{F}) \setminus \mathcal{C}_{i-1})) \\
&= \mathcal{H} \setminus (\mathcal{F} \cup (\mathcal{H} \setminus \mathcal{C}_{i-1})) \\
&\subseteq \mathcal{H} \setminus (\mathcal{H} \setminus \mathcal{C}_{i-1}) \\
&\subseteq \mathcal{C}_{i-1} .
\end{aligned}$$

Since $S^i(\mathcal{G}) \subseteq S^{i-1}(\mathcal{G})$, we can choose by our assumption $t \in S^{i-1}(\mathcal{G}) \setminus S^i(\mathcal{G})$. Since $t \in S^{i-1}(\mathcal{G})$, we have $S_X(t) = 1$ for every $X \in \mathcal{G}$. Also $S_{X_i}(t) = 1$, otherwise $t \in S^i(\mathcal{G})$. Lemma 6.12 now implies that for every $X \in \mathcal{C}_{i-1} \setminus \mathcal{F}$, it holds that $X \notin closure(\mathcal{F}) = \mathcal{H}$. This proves that $\mathcal{H} = \mathcal{G}'$, and Lemma 6.10 now provides the needed contradiction. Consequently, $S^{i-1}(\mathcal{G}) = S^i(\mathcal{G})$.

Assume now that $T' = block(\mathcal{G}') \neq \emptyset$, i.e., there exists a $T' \in \mathcal{T}_\mathcal{C}$ such that $\mathcal{G}' = sets(T', \mathcal{C})$. The induction assumption now guarantees that $e^{i-1}(\mathcal{G}') = \left|S^{i-1}(\mathcal{G}')\right|$.

We have

$$\begin{aligned}
\left|S^i(\mathcal{G})\right| &= \left|S^{i-1}(\mathcal{G})\right| - \left|S^{i-1}(\mathcal{F})\right| && \text{Property 2} \\
&= \left|S^{i-1}(\mathcal{G})\right| - \left|S^{i-1}(\mathcal{G}')\right| && \text{Property 3} \\
&= e^{i-1}(\mathcal{G}) - e^{i-1}(\mathcal{G}') , && \text{induction assumption}
\end{aligned}$$

which proves the theorem. □

Example 6.4. *Let us apply* QieBlockSizes *to our running example. Recall that* $\mathcal{C} = \{abc, cd, def\}$ *(see Table 6.1). For brevity, we restrict ourselves to the first three blocks. In step 1, the first three blocks remain unaffected, since they all contain* X_1*. In step 2, only the third block does not contain* X_2*; we subtract the second block from it. In step 3, we subtract the first block from the second block. From the third block we do not have to subtract anything, since the* 3*-closure of the corresponding block does not appear in* $\mathcal{T}_{\mathcal{C}}$*. We have thus calculated the sizes of the first three blocks using two subtractions, rather than three, as was previously required in Example 6.3.*

Finally, we can significantly optimize the algorithm as follows. Assume that we can divide $\mathcal{C}$ into two disjoint groups $\mathcal{C}_1$ and $\mathcal{C}_2$, such that if $X_1 \in \mathcal{C}_1$ and $X_2 \in \mathcal{C}_2$, then $X_1 \cap X_2 = \emptyset$. Let $B = \bigcup \mathcal{C}_1$ be the set of items occurring in $\mathcal{C}_1$. Theorem 6.1 implies that $p^*(\mathcal{A}) = p^*(B)p^*(\mathcal{A} \setminus B)$. In other words, the maximum entropy distribution can be factorized into two *independent* distributions, namely $p^*(B)$ and $p^*(\mathcal{A} \setminus B)$, more importantly, the factor $p^*(B)$ depends only on $\mathcal{C}_1$. Consequently, if we wish to compute the probability $p^*(X = 1)$ such that $X \in B$, we can ignore all variables outside B and all itemsets outside $\mathcal{C}_1$. The number of computations to be performed by QieBlockSizes can now be greatly reduced, since in the case of disjointness $|\mathcal{T}_{\mathcal{C}}| = |\mathcal{T}_{\mathcal{C}_1}| \times |\mathcal{T}_{\mathcal{C}_2}|$, and we can simply compute the block sizes for $\mathcal{T}_{\mathcal{C}_1}$ and $\mathcal{T}_{\mathcal{C}_2}$ separately. Naturally, this decomposition can also be applied when there are more than two disjoint groups of itemsets.

Moreover, in order to *guarantee* that we can apply the above separation, we could reduce the solution space slightly by imposing a limit on the number of items (or itemsets) per group, such that the number of blocks for each group remains small. Alternatively, we could first partition the items of the dataset into smaller, approximately independent groups, and subsequently apply the algorithm for each group separately. To do this, the attribute clustering method of Chapter 4, which identifies the optimal partitioning using MDL, would be a logical choice.

Querying the Model

We have seen how we can efficiently query the probability of an itemset $X \in \mathcal{C}$ when given an exponential distribution $p_{\mathcal{C}}$ based on $\mathcal{C}$. In order to compute the probability of an arbitrary itemset Y that is not a member of $\mathcal{C}$, we do the following. We first set $\mathcal{C}' = \mathcal{C} \cup \{Y\}$ and compute the block

probabilities $e(T')$ for all T' in $\mathcal{T}_{\mathcal{C}'}$ by calling QieBlockSizes. Then, we can simply use the parameters of $p_{\mathcal{C}}$ to compute $p_{\mathcal{C}}(Y=1)$ as follows,

$$p_{\mathcal{C}}(Y=1) = \sum_{\substack{T \in \mathcal{T}_{\mathcal{C}'} \\ S_Y(T)=1}} e(T) \prod_{X \in \mathcal{C}} u_X^{S_X(T)} \ .$$

Thus, to obtain the probability of an itemset, it suffices to compute the block probabilities in $\mathcal{T}_{\mathcal{C}'}$, for which we know that $|\mathcal{T}_{\mathcal{C}'}| \leq 2|\mathcal{T}_{\mathcal{C}}|$.

Computational Complexity

Let us analyze the complexity of the IterativeScaling algorithm. To this end, we define $nb(\mathcal{C}) = |\mathcal{T}_{\mathcal{C}}|$ as the number of blocks in $\mathcal{T}_{\mathcal{C}}$. The computational complexity of QieBlockSizes is

$$O\left(k \cdot nb(\mathcal{C}) \log nb(\mathcal{C})\right) \ ,$$

for a given collection $\mathcal{C}$, with $|\mathcal{C}| = k$. The logarithmic factor comes from looking for the block T' on line 11. Note that $nb(\mathcal{C}) \leq 2^k$, and hence $\log nb(\mathcal{C}) \leq k$. Assume now that we can split $\mathcal{C}$ into L disjoint parts $\mathcal{C} = \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_L$, such that if $X \in \mathcal{C}_i$ and $Y \in \mathcal{C}_j$ then $X \cap Y = \emptyset$. As mentioned in Section 6.3, we can now simply compute L independent distributions at a lower total cost. Denoting $B_i = \bigcup_{X \in \mathcal{C}_i} X$, it holds that $nb(\mathcal{C}_i) \leq \min\left(2^{|\mathcal{C}_i|}, 2^{|B_i|}\right)$. If $\mathcal{C}_i$ cannot be split any further, this usually means that either $|\mathcal{C}_i|$ is small, or the itemsets in $\mathcal{C}_i$ overlap a lot and $nb(\mathcal{C}_i) \ll 2^{|\mathcal{C}_i|}$. The total execution time of IterativeScaling is

$$O\left(K \sum_{i=1}^{L} |C_i| nb(\mathcal{C}_i) \log nb(\mathcal{C}_i)\right) \ ,$$

where K is the number of iterations, which is usually low. The complexity of estimating the frequency of an itemset requires running QieBlockSizes once and, hence is equal to

$$O\left(\sum_{i=1}^{L} |C_i| nb(\mathcal{C}_i) \log nb(\mathcal{C}_i)\right) \ .$$

Including Background Knowledge into the Model

Typically, when analyzing data we have some basic background knowledge about it. For instance, we may already know the individual frequencies of the items, i.e., the *column margins*. These margins supply some basic information about the data, for instance whether *tomatoes* are sold often or not in a supermarket database. These individual frequencies are intuitive and easy to calculate, yet already provide information on whether some combinations of items are more or less likely to occur frequently. For this reason, many existing techniques use the independence model as a basis to discover interesting patterns, e.g., [Brin et al., 1997, Aggarwal and Yu, 1998]. Another form of background information that is often used are *row margins*, that is, the probabilities that a transaction contains a certain number of items, e.g., [Gionis et al., 2007, Hanhijärvi et al., 2009, Kontonasios and De Bie, 2010, Tatti and Mampaey, 2010]—see also Chapter 5. If we know that most transactions are rather small, large itemsets are more likely to have a low frequency.

Clearly, when we analyze data we want to incorporate this background knowledge, since otherwise we would simply rediscover it. If we do include it in our analysis, we discover itemsets that are interesting *with respect to* what we already know. Therefore, we extend the BIC and MDL quality scores of Definition 6.1 to incorporate background knowledge, say, $\mathcal{B}$. Although in this section we focus on row and column margins as forms of background knowledge, many other patterns or count statistics that can be expressed as linear constraints on transactions could be used, for instance, a set of association rules and their confidence. Therefore, we intentionally omit the explicit specification of $\mathcal{B}$.

Definition 6.4. *Given a dataset $\mathcal{D}$ and some background knowledge $\mathcal{B}$, we define the* BIC *score of a collection of itemsets $\mathcal{C} = \{X_1, \ldots, X_k\}$ with respect to $\mathcal{B}$ as*

$$\textsc{bic}(\mathcal{C}, \mathcal{D}; \mathcal{B}) = -\log p^*_{\mathcal{B},\mathcal{C}}(\mathcal{D}) + k/2 \log |D| \ ,$$

similarly, we define the MDL *score of $\mathcal{C}$ with respect to $\mathcal{B}$ as*

$$\textsc{mdl}(\mathcal{C}, \mathcal{D}; \mathcal{B}) = -\log p^*_{\mathcal{B},\mathcal{C}}(\mathcal{D}) + l_1 k + l_2 x + 1 \ ,$$

where $p^*_{\mathcal{B},\mathcal{C}}$ is the maximum entropy distribution satisfying the background knowledge $\mathcal{B}$ and $p^*_{\mathcal{B},\mathcal{C}}(X = 1) = fr(X)$ for all $X \in \mathcal{C}$, and l_1 and l_2 are the same as in Definition 6.1.

Note that while the background knowledge is included in the log-likelihood term of $s(\mathcal{C}, \mathcal{D}; \mathcal{B})$ (where s denotes either BIC or MDL), it is not included in the penalty term. We choose not to do so because we will assume that our background knowledge is consistent with the data and invariable. We could alternatively define

$$s(\mathcal{C}, \mathcal{D}, \mathcal{B}) = s(\mathcal{C}, \mathcal{D}; \mathcal{B}) + L(\mathcal{B}) \ ,$$

where $L(\mathcal{B})$ is some term which penalizes $\mathcal{B}$, however, since this term would be equal for all $\mathcal{C}$, it might as well be omitted.

In this section, we show how to include row and column margins as background knowledge into our algorithm without, however, blowing up its computational complexity. If we were, for instance, to naively add all singleton itemsets $\mathcal{I}$ and their frequencies to an itemset collection $\mathcal{C}$, the number of transaction blocks in the corresponding partition would become $|\mathcal{T}_{\mathcal{C}\cup\mathcal{I}}| = |\mathcal{T}| = 2^N$, by which we would be back at square one. Therefore, we consider row and column margins separately from $\mathcal{C}$ in our computations.

First, let us consider using only column margins, viz., item frequencies. With these, we will build an independence model, while with $\mathcal{C}$ we again partition the transactions $\mathcal{T}$ as above. Then we simply combine the two to obtain the maximum entropy distribution. As before, the maximum entropy model has an exponential form:

$$p^*_{\mathcal{C}'}(\mathcal{A} = t) = u_0 \prod_{X \in \mathcal{C}} u_X^{S_X(t)} \prod_{i \in \mathcal{I}} v_i^{S_i(t)} \ .$$

The second part of the product defines an independence distribution

$$v(\mathcal{A} = t) = v_0 \prod_i v_i^{S_i(t)} \ ,$$

where v_0 is a normalization factor. It is not difficult to see that $v(a_i = 1) = v_i/(1 + v_i)$, for all for $a_i \in \mathcal{A}$. It should be noted that while $p^*(a_i = 1) = fr(a_i)$, in general it does not necessarily hold that $v(a_i = 1) = fr(a_i)$. Now we can write

$$p^*_{\mathcal{C}'}(\mathcal{A} \in T) = v(\mathcal{A} \in T) \frac{u_0}{v_0} \prod_{X \in \mathcal{C}} u_X^{S_X(T)} \ .$$

Thus, we simply need to compute $v(\mathcal{A} \in T)$ for each block T, which is computed very similarly to $e(T)$, using QIEBLOCKSIZES. Note that $e(\mathcal{A} = t)$

is in fact nothing more than a uniform distribution over $\mathcal{T}$, multiplied by 2^N. To compute $v(\mathcal{A} \in T)$, we simply initiate the algorithm with the cumulative sizes of the blocks with respect to v, which are equal to

$$c(\mathcal{A} \in T) = v_0 \prod_{i \in I} v_i \ ,$$

where $I = \bigcup sets(T; \mathcal{C})$. Hence, we can include the item frequencies at a negligible additional cost. To update the v_i parameters, we must also query the probability of a single item. We can achieve this by simply adding the corresponding singleton to $\mathcal{C}$, the same way as we would query any itemset.

Next, we also include row margins in the background information. Let us define the indicator functions $S^j(t) : \mathcal{T} \to \{0, 1\}$ for $j \in \{0, \ldots, N\}$ such that $S^j(t) = 1$ if and only if the number of ones in t, denoted as $|t|$, is equal to j. Further, for any distribution p on $\mathcal{A}$, let us write $p(|\mathcal{A}| = j)$ to indicate the probability that a transaction contains j items. Again, the maximum entropy distribution has an exponential form,

$$p^*_{\mathcal{B},\mathcal{C}}(\mathcal{A} = t) = u_0 \prod_{X \in \mathcal{C}} u_X^{S_X(t)} \prod_{i \in \mathcal{I}} v_i^{S_i(t)} \prod_{j=0}^{N} w_j^{S^j(t)} \ .$$

The row and column margins define a distribution

$$w(\mathcal{A} = t) = v_0 \prod_{i \in \mathcal{I}} v_i^{S_i(t)} \prod_{j=0}^{N} w_j^{S^j(t)} \ ,$$

where v_0 is a normalization factor. Now, the probabilities $p^*(\mathcal{A} \in T)$ are computed using the probabilities

$$\begin{aligned} p^*(\mathcal{A} \in T, |\mathcal{A}| = j) &= w(\mathcal{A} \in T, |\mathcal{A}| = j) \frac{u_0}{v_0} \prod_{X \in \mathcal{C}} u_X^{S_X(T)} \\ &= w_j v(\mathcal{A} \in T, |\mathcal{A}| = j) \frac{u_0}{v_0} \prod_{X \in \mathcal{C}} u_X^{S_X(T)} \end{aligned}$$

for $j = 0, \ldots, N$, and we marginalize over j to obtain

$$\begin{aligned} p^*(\mathcal{A} \in T) &= \sum_{j=0}^{N} p^*(\mathcal{A} \in T, |\mathcal{A}| = j) \\ &= \frac{u_0}{v_0} \prod_{X \in \mathcal{C}} u_X^{S_X(T)} \sum_{j=0}^{N} w_j v(\mathcal{A} \in T, |\mathcal{A}| = j) \ . \end{aligned}$$

Just as before, we compute the probabilities $v(\mathcal{A} \in T, |\mathcal{A}| = j)$ using the QieBlockSizes algorithm. Let $I = \bigcup\{X \mid X \in \mathit{sets}(\mathcal{C}; T)\}$, then the corresponding cumulative probability becomes

$$c(\mathcal{A} \in T, |\mathcal{A}| = j) = v_0 \prod_{i \in I} v_i \cdot v(|\mathcal{A}| = j \mid I = 1) \; .$$

Computing the probabilities $v(|\mathcal{A}| = j)$, and similarly $v(|\mathcal{A}| = j \mid I = 1)$, can be done from scratch in $O(N^2)$ time and $O(N)$ space, using the following recurrence relation,

$$v(|\mathcal{A}_i| = j) = v(a_i) \cdot v(|\mathcal{A}_{i-1}| = j - 1) + (1 - v(a_i)) \cdot v(|\mathcal{A}_{i-1}| = j) \; ,$$

where $\mathcal{A}_i = \{a_1, \ldots, a_i\}$. Starting from $\mathcal{A}_0 = \emptyset$, the ComputeSizeProbabilities algorithm adds each item a_i until we have computed all probabilities $v(|\mathcal{A}_N| = j)$ where $\mathcal{A}_N = \mathcal{A}$; see Algorithm 6.4. The time complexity can be reduced to $O(N)$, by applying the updates that IterativeScaling performs on $v(a_i)$, to the probabilities $v(|\mathcal{A}| = j)$ as well, this is done by UpdateSizeProbabilities in Algorithm 6.5. The algorithm first removes the item, and then re-adds it with the updates probability. (Note that Algorithms 6.4 and 6.5 are repeated here from Chapter 5 for convenience, further details can be found there.) Computing item frequencies is done by adding singletons to $\mathcal{C}$. To compute the row margin probabilities $p^*(|\mathcal{A}| = j)$, we simply marginalize over $\mathcal{T}_\mathcal{C}$,

$$p^*(|\mathcal{A}| = j) = \sum_{T \in \mathcal{T}_\mathcal{C}} p^*(\mathcal{A} \in T, |\mathcal{A}| = j) \; .$$

Hence, including the row margins increases the time and space complexity of model computation and inference by a factor of N.

6.4 Problem Statements

In this section we identify four different problems that we intend to solve using the theory introduced above. We assume some given set $\mathcal{B}$ that represents our background knowledge, e.g., the individual item frequencies, some arbitrary collection of itemsets, or simply the empty set. We start simple, with a size constraint k and a collection $\mathcal{F}$ of potentially interesting itemsets to choose from, for instance, frequent itemsets, closed itemsets, itemsets of a certain size, etc.

Algorithm 6.4: ComputeSizeProbabilities

input : an independence distribution v over $\mathcal{A}$, with item probabilities $v(a_i)$ for $i = 1, \ldots, N$
output: the probabilities $g_j = v(|\mathcal{A}| = j)$ for $j = 0, \ldots, N$

$g_0 \leftarrow 1$
for $j = 1, \ldots, N$ **do**
  $g_j \leftarrow 0$
end
for $i = 1, \ldots, N$ **do**
  for $j = i, \ldots, 1$ **do**
    $g_j \leftarrow v(a_i) \cdot g_{j-1} + (1 - v(a_i)) \cdot g_j$
  end
  $g_0 \leftarrow (1 - v(a_i)) \cdot g_0$
end
return g

Algorithm 6.5: UpdateSizeProbabilities

input : the probabilities $g_j = v(|\mathcal{A}| = j)$ for an independence distribution v; an updated probability x for item a_i
output: the updated probabilities $g_j = v(|\mathcal{A}| = j)$ for $j = 0, \ldots, N$

$g_0 \leftarrow g_0 / (1 - v(a_i))$
for $j = 1, \ldots, N$ **do**
  $g_j \leftarrow \left(g_j - v(a_i) g_{j-1}\right) / (1 - v(a_i))$
end
update v such that $v(a_i) = x$
for $j = N, \ldots, 1$ **do**
  $g_j \leftarrow v(a_i) \cdot g_{j-1} + (1 - v(a_i)) \cdot g_j$
end
$g_0 \leftarrow (1 - v(a_i)) \cdot g_0$
return g

Problem 6.1 (Most Informative k-Subset)**.** *Given a dataset $\mathcal{D}$, a set $\mathcal{B}$ that represents our background knowledge, an integer k, and a collection of potentially interesting itemsets $\mathcal{F}$, find the subset $\mathcal{C} \subseteq \mathcal{F}$ with $|\mathcal{C}| \leq k$ such that $s(\mathcal{C}, \mathcal{D}; \mathcal{B})$ is minimized.*

Note that if we choose $k = 1$, this problem reduces to 'Find the Most Interesting Itemset in $\mathcal{F}$', which means simply scoring $\mathcal{C} = \{X\}$ with respect to $\mathcal{B}$ for each set $X \in \mathcal{F}$, and selecting the best one. Further, these scores provide a ranking of the itemsets $X \in \mathcal{F}$ with regard to our background knowledge $\mathcal{B}$.

Now, if we do not want to select a k ourselves, we can rely on either BIC or MDL to identify the best-fitting, least-redundant model.

Problem 6.2 (Most Informative Subset)**.** *Given a dataset $\mathcal{D}$, a set $\mathcal{B}$ that represents our background knowledge, and a collection of potentially interesting itemsets $\mathcal{F}$, find the subset $\mathcal{C} \subseteq \mathcal{F}$ such that $s(\mathcal{C}, \mathcal{D}; \mathcal{B})$ is minimized.*

When we do not want to constrain the itemset collection $\mathcal{F}$, we can simply use all itemsets. Problem 6.1 then generalizes to the following.

Problem 6.3 (k Most Informative Itemsets)**.** *Given a dataset $\mathcal{D}$, an integer k, and a set $\mathcal{B}$ that represents our background knowledge, find the collection of itemsets $\mathcal{C}$, with $|\mathcal{C}| \leq k$, such that $s(\mathcal{C}, \mathcal{D}; \mathcal{B})$ is minimized.*

Similarly, and most generally, we can simply consider finding the best collection of itemsets altogether.

Problem 6.4 (Most Informative Itemsets)**.** *Given a dataset $\mathcal{D}$ and a set $\mathcal{B}$ that represents our background knowledge, find the collection of itemsets $\mathcal{C}$ such that $s(\mathcal{C}, \mathcal{D}; \mathcal{B})$ is minimized.*

Note that these problem statements do not require $\mathcal{F}$ to be explicitly available beforehand (let alone the complete set of itemsets), i.e., it does not have to be mined or materialized in advance (the details of this are described at the end of Section 6.5).

In the next section, we discuss how we can mine those collections of itemsets to solve the above problems.

6.5 Mining Informative Succinct Summaries

In Section 6.3 we described how to compute the maximum entropy model and its BIC or MDL quality score *given* a set of itemsets. Finding the *optimal* collection as stated in Section 6.4, however, is clearly infeasible. The size of the search space is

$$\sum_{j=0}^{k} \binom{|\mathcal{F}|}{j} \leq 2^{|\mathcal{F}|} .$$

If we do not restrict the candidate itemsets, then the total number of all (non-singleton) itemsets is $|\mathcal{F}| = 2^N - N - 1$. Moreover, our quality scores are not monotonically increasing or decreasing, which prevents us from straightforwardly exploring the search space.

Therefore, we resort to using a heuristic, greedy approach. Starting with a set of background knowledge—for instance the column margins—we incrementally construct our summary by iteratively adding the itemset that reduces the score function the most. The algorithm stops either when k interesting itemsets are found, or when the score no longer decreases. The pseudo-code of the MTV algorithm, which mines Maximally informaTiVe itemset summaries, is given in Algorithm 6.6.

Due to its incremental nature, we note that we can apply an optimization to the algorithm. When we call IterativeScaling on line 5, rather than computing p^* from scratch, we can initialize the algorithm with the parameters of the previous p^* (line 1 of Algorithm 6.1), instead of with the uniform distribution. In doing so, the IterativeScaling procedure converges faster. Further, we can also reuse part of the computations from QieBlockSizes.

A Heuristic for Scoring Itemsets

Finding the most informative itemset to add to the current collection is practically infeasible, since it involves solving the maximum entropy model for each and every candidate. This remains infeasible even if we restrict the search space (for example, using only frequent itemsets). Therefore, instead of selecting the candidate that optimizes the BIC or MDL score directly, we select the candidate that maximizes a heuristic which expresses the divergence between its frequency and its estimate. To derive and motivate this heuristic we first present the following theorem.

Algorithm 6.6: MTV

input : a binary dataset $\mathcal{D}$; background knowledge $\mathcal{B}$; an integer $k \leq \infty$
output: an itemset collection $\mathcal{C}$ minimizing $s(\mathcal{C}, \mathcal{D}; \mathcal{B})$

$\mathcal{I} \leftarrow$ items in $\mathcal{D}$
while $s(\mathcal{C}, \mathcal{D}; \mathcal{B})$ decreases **and** $|\mathcal{C}| < k$ **do**
  $X \leftarrow$ FindMostInformativeItemset$(\emptyset, \mathcal{I}, \emptyset)$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{X\}$
  $p^*_{\mathcal{B},\mathcal{C}} \leftarrow$ IterativeScaling$(\mathcal{C})$
  compute $s(\mathcal{C}, \mathcal{D}; \mathcal{B})$
end
return $\mathcal{C}$

Theorem 6.14. *Given an itemset collection $\mathcal{C}$, a dataset $\mathcal{D}$, and a collection of candidate itemsets $\mathcal{F}$. Let s denote either* BIC *or* MDL*. It holds that*

$$\begin{aligned}\arg\min_{X \in \mathcal{F}} s(\mathcal{C} \cup \{X\}) &= \arg\max_{X \in \mathcal{F}} KL\Big(p^*_{\mathcal{C} \cup \{X\}} \,\|\, p^*_{\mathcal{C}}\Big) - r(X) \\ &= \arg\min_{X \in \mathcal{F}} KL\Big(q_{\mathcal{D}} \,\|\, p^*_{\mathcal{C} \cup \{X\}}\Big) + r(X)\end{aligned}$$

where

$$r(X) = \begin{cases} 0 & \text{if } s = \text{BIC} \\ |X| \log N / |D| & \text{if } s = \text{MDL} \end{cases}$$

Proof. Let us write $\mathcal{C}' = \mathcal{C} \cup \{X\}$. Corollary 6.2 states that $-\log p^*_{\mathcal{C}'}(\mathcal{D}) = |\mathcal{D}| H\big(p^*_{\mathcal{C}'}\big)$. In addition, we can show with a straightforward calculation that

$$KL(p^*_{\mathcal{C}'} \,\|\, p^*_{\mathcal{C}}) = H(p^*_{\mathcal{C}}) - H(p^*_{\mathcal{C}'}) \ .$$

For BIC the difference in the penalty terms of $s(\mathcal{C})$ and $s(\mathcal{C}')$ is equal to $1/2 \log |D|$, which is identical for all itemsets X, and hence may be eliminated from the arg max. For MDL, the difference in penalty terms can similarly be reduced to $|X| \log N$. The second equality follows similarly. □

Thus, we search for the itemset X for which the new distribution diverges maximally from the previous one, or equivalently, brings us as close to the empirical distribution as possible—taking into account the penalty term for

MDL. Note that for BIC, since $r(X) = 0$, the algorithm simply tries to maximize the likelihood of the model, and the penalty term functions as a stopping criterion; the algorithm terminates when the increase in likelihood (i.e., decrease of the negative log-likelihood) is not sufficient to counter the increase of the penalty. When we use MDL, on the other hand, $r(X)$ represents part of the penalty term, and hence this guides the algorithm in its search.

The heuristic we employ uses an approximation of the *KL* divergence, and is in fact a simpler *KL* divergence itself. In the expression

$$KL(p^*_{\mathcal{C}'} \parallel p^*_{\mathcal{C}}) = \sum_{t \in \mathcal{T}} p^*_{\mathcal{C}'}(\mathcal{A} = t) \log \frac{p^*_{\mathcal{C}'}(\mathcal{A} = t)}{p^*_{\mathcal{C}}(\mathcal{A} = t)} \tag{6.4}$$

we merge the terms containing X in one term, and the terms not containing X into another term. To differentiate between these two divergences, let us write the function $kl : [0,1] \times [0,1] \to \mathbb{R}^+$ as follows,

$$kl(x, y) = x \log \frac{x}{y} + (1 - x) \log \frac{1 - x}{1 - y} \,,$$

then we approximate Eq. 6.4 by $kl(fr(X)\,, p^*_{\mathcal{C}}(X = 1))$. We will write the latter simply as $kl(X)$ when fr and p^* are clear from the context. To compute this heuristic, we only need the frequency of X, and its estimate according to the current p^* distribution. This gives us a measure of the divergence between $fr(X)$ and $p^*_{\mathcal{C}}(X = 1)$, i.e., its surprisingness given the current model.

The following theorem shows the relation between *KL* and *kl*.

Theorem 6.15. *For an itemset collection $\mathcal{C}$ and an itemset X, it holds that*

$$0 \leq kl(X) \leq KL\Big(p^*_{\mathcal{C} \cup \{X\}} \parallel p^*_{\mathcal{C}}\Big) \,.$$

Moreover, $kl(X) = 0$ if and only if $KL\Big(p^*_{\mathcal{C} \cup \{X\}} \parallel p^*_{\mathcal{C}}\Big) = 0$, i.e., when $fr(X) = p^*_{\mathcal{C}}(X = 1)$.

Proof. Both inequalities follow directly from the log-sum inequality, which states that for any nonnegative numbers a_i, b_i, with $a = \sum_i a_i$ and $b = \sum_i b_i$, it holds that

$$\sum_i a_i \log \frac{a_i}{b_i} \geq a \log \frac{a}{b} \,.$$

For the equality to zero, we have $kl(fr(X), p^*_{\mathcal{C}}(X = 1)) = 0$ if and only if $fr(X) = p^*_{\mathcal{C}}(X = 1)$. In this case it holds that $p^*_{\mathcal{C}'} = p^*_{\mathcal{C}}$ which is true if and only if $KL(p^*_{\mathcal{C}'} \,\|\, p^*_{\mathcal{C}}) = 0$. □

Using Theorem 6.14, the heuristic we employ is defined as

$$h(X) = kl(fr(X), p^*_{\mathcal{C}}(X = 1)) - r(X)$$

and we will make use of the following assumption:

$$\arg\min_{X \in \mathcal{F}} s(\mathcal{C} \cup \{X\}) = \arg\max_{X \in \mathcal{F}} h(X) .$$

Note that h has an elegant interpretation: it is equal to the Kullback-Leibler divergence after exactly one step in the IterativeScaling algorithm (when initializing p^* with the parameters from the previous model, as discussed above). Since the KL divergence increases monotonically during the Iterative Scaling procedure, if the total divergence is large, then we expect to already see this in the first step of the procedure.

Searching for the Most Informative Itemset

To find the itemset maximizing $h(X)$, we take a depth-first branch-and-bound approach. We exploit the fact that kl is a convex function, and employ the bound introduced by Nijssen et al. [2009] to prune large parts of the search space as follows. Say that for a candidate itemset X in the search space, its maximal possible extension in the branch below it is $X \cup Y$ (denoted XY), then for any itemset W such that $X \subseteq W \subseteq XY$, it holds that

$$h(W) = kl(W) - r(W) \leq \max\Big\{kl\big(fr(X), p^*(XY)\big), kl\big(fr(XY), p^*(X)\big)\Big\} - r(X) .$$

If this upper bound is lower than the best value of the heuristic seen so far, we know that no (local) extension W of X can ever become the best itemset with respect to the heuristic, and therefore we can safely prune the branch of the search space below X. The FindMostInformativeItemset algorithm is given in Algorithm 6.7.

An advantage of this approach is that we do not need to collect the frequencies of all candidate itemsets beforehand. Instead, we just compute them on the fly as we need them (line 1). For instance, if we wish to pick itemsets

Algorithm 6.7: FindMostInformativeItemset

input : an itemset X; remaining items Y; the currently best set Z
output: the itemset between X and XY maximizing h, or Z

compute $fr(X)$ and $p^*(X)$
if $h(X) = kl(fr(X), p^*(X)) - r(X) > h(Z)$ **then**
  $Z \leftarrow X$
end
compute $fr(XY)$ and $p^*(XY)$
$b \leftarrow \max\{kl(fr(X), p^*(XY)), kl(fr(XY), p^*(X))\} - r(X)$
if $b > h(Z)$ **then**
  for $y \in Y$ **do**
    $Y \leftarrow Y \setminus \{y\}$
    $Z \leftarrow$ FindMostInformativeItemset($X \cup \{y\}$, Y, Z)
  end
end
return Z

from a collection $\mathcal{F}$ of frequent itemsets for some minimum support threshold, we can integrate the support counting with the depth-first traversal of the algorithm, rather than first mining and storing $\mathcal{F}$ in its entirety. Since for real datasets and non-trivial support thresholds billions of frequent itemsets are easily discovered, this indubitably makes our approach more practical.

6.6 Experiments

In this section we experimentally evaluate our method and empirically validate the quality of the returned summaries.

Setup

The MTV algorithm was implemented in C++, and the source code is provided for research purposes.[1] All experiments were executed on a 2.67GHz (six-core) Intel Xeon machine with 12GB of memory, running Linux. All reported timings are of the single-threaded implementation of our algorithm.

[1] http://www.adrem.ua.ac.be/implementations

We evaluate our method on three synthetic datasets, as well as on eleven real datasets. Their basic characteristics are given in Table 6.2.

The *Independent* data has independent items with random frequencies between 0.2 and 0.8. In the *Markov* dataset each item is a noisy copy of the previous one, with a random copy probability between 0.5 and 0.8. The *Mosaic* dataset is generated by randomly planting five itemsets of size 5 with random frequencies between 0.2 and 0.5 in a database with 1% noise.

The *Abstracts* dataset contains the abstracts of all accepted papers at the ICDM conference up to 2007, where all words have been stemmed and stop words have been removed [Kontonasios and De Bie, 2010].

The *Accidents* [Geurts et al., 2003], *Kosarak*, *Mushroom*, and *Retail* [Brijs et al., 1999] datasets were downloaded from the FIMI dataset repository [Goethals and Zaki, 2003].

The datasets *Chess (kr–k)* and *Plants* were obtained from the UCI ML Repository [Frank and Asuncion, 2010], the former was converted into binary form by creating an item for each attribute-value pair. The latter contains a list of plants, and the U.S. and Canadian states where they occur.

The *DNA Amplification* data contains information on DNA copy number amplifications [Myllykangas et al., 2006]. Such copies activate oncogenes and are hallmarks of nearly all advanced tumors. Amplified genes represent attractive targets for therapy, diagnostics and prognostics. In this dataset items are genes, and transactions correspond to patients.

The *Lotto* dataset was obtained from the website of the Belgian National Lottery, and contains the results of all lottery draws between May 1983 and May 2011.[2] Each draw consist of seven numbers (six plus one bonus ball) out of a total of forty-two.

The *Mammals* data consists of presence records of European mammals within geographical areas of 50×50 km^2 [Mitchell-Jones et al., 1999].[3]

The *MCADD* dataset was obtained from the Antwerp University Hospital. Medium-Chain Acyl-coenzyme A Dehydrogenase Deficiency (MCADD) [Baumgartner et al., 2005, Van den Bulcke et al., 2011] is a deficiency newborn babies are screened for during a Guthrie test on a heel prick blood sample. The instances are represented by a set of 21 features: twelve different acylcarnitine concentrations measured by tandem mass spectrometry

[2] http://www.nationaleloterij.be

[3] The full version of the *Mammals* dataset is available for research purposes from the Societas Europaea Mammalogica at http://www.european-mammals.org.

Table 6.2: The synthetic and real datasets used in the experiments. Shown for each dataset are the number of items $|\mathcal{A}|$, the number of transactions $|\mathcal{D}|$, the minimum support threshold for the set of candidate frequent itemsets $\mathcal{F}$ and its size.

	Data Properties		*Candidate Collection*	
Dataset	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{D}\rvert$	*minsup*	$\lvert\mathcal{F}\rvert$
Independent	50	100 000	5 000	1 055 921
Markov	50	100 000	5 000	377 011
Mosaic	50	100 000	5 000	101 463
Abstracts	3 933	859	10	75 061
Accidents	468	340 183	50 000	2 881 487
Chess (kr–k)	58	28 056	5	114 148
DNA Amplification	391	4 590	5	$4.57{\cdot}10^{12}$
Kosarak	41 270	990 002	1 000	711 424
Lotto	42	2 386	1	139 127
Mammals	121	2 183	200	93 808 244
MCADD	198	31 924	50	1 317 234
Mushroom	119	8 124	100	66 076 586
Plants	70	34 781	2 000	913 440
Retail	16 470	88 162	10	189 400

(TMS), together with four of their calculated ratios and five other biochemical parameters, each of which was discretized using k-means clustering with a maximum of ten clusters per feature.

Our method is parameter-free. That is, given enough time, it can select the best set of itemsets from the complete space of possible itemsets. However, although quite efficient, in practice it may not always be desirable or feasible to consider *all* itemsets, for instance, for dense or large datasets, for which we might want to explicitly exclude low-frequency itemsets from our search. General speaking, choosing a larger candidate space, yields a larger search space, and hence potentially better models. In our experiments we therefore consider collections of frequent itemsets $\mathcal{F}$ mined at support thresholds as low as feasible. The actual thresholds and corresponding size of $\mathcal{F}$ are depicted in Table 6.2. Note that the minimum support threshold is used to

limit the size of $\mathcal{F}$, and is strictly speaking not a parameter of the algorithm itself. To ensure efficient computation, we impose a maximum of 10 items per group, as described at the end of Section 6.3. Further, we terminate the algorithm if the runtime exceeds two hours. For the sparse datasets with many transactions (the synthetic ones, *Accidents*, *Kosarak*, and *Retail*), we mine and store the supports of the itemsets, rather than computing them on the fly. Caching the supports is faster in these cases, since for large datasets support counting is relatively expensive. The runtimes reported below include this additional step.

In all experiments, we set the background information $\mathcal{B}$ to contain the column margins, that is, we start from the independence model. For *Chess (kr–k)* and *Mushroom*, we also perform experiments which include the column margins, since their original form is categorical, and hence we know that each transaction has a fixed size. For the *Lotto* data we additionally use *only* the row margins, which implicitly assumes all numbers to be equiprobable, and uses the fact that each draw consists of seven numbers.

A First Look at the Results

In Tables 6.3 and 6.4 we give the scores and sizes of the discovered summaries, the time required to compute them, and for comparison we include the score of the background model, using BIC and MDL respectively as the quality score. For most of the datasets we consider, the algorithm identifies the optimum quite rapidly, i.e., within minutes. For these datasets, the number of discovered itemsets, k, is indicated in boldface. For three of the datasets, i.e., the dense *MCADD* dataset, and the large *Kosarak* and *Retail* datasets, the algorithm did not find the optimum within two hours.

For both BIC and MDL, we note that the number of itemsets in the summaries is very small, and hence manual inspection of the results by an expert is highly feasible. We see that for most datasets the scores (and thus relatedly the negative log-likelihood) decreases a lot, implying that the summaries we find are of high quality. Moreover, for highly structured datasets such as *DNA Amplification*, *Kosarak*, and *Plants*, this improvement is very large, and only a handful of itemsets is required to describe their main structure.

Comparing the two tables, we see that the number of itemsets selected by BIC compared to MDL tends to be the same or a bit higher, indicating that BIC is more permissive in adding itemsets—an expected result. If we (crudely)

Table 6.3: The BIC scores of the discovered models, with respect to the background information $\mathcal{B}$. Shown are the number of itemsets k, the wall clock time, the score of $\mathcal{C}$, and the score of the empty collection. (Lower scores are better.) Values for k identified as optimal by BIC are given in boldface.

Dataset	k	*time*	$\text{BIC}(\mathcal{C}, \mathcal{D}; \mathcal{B})$	$\text{BIC}(\emptyset, \mathcal{D}; \mathcal{B})$
Independent	**7**	2m 14s	4 494 219	4 494 242
Markov	**62**	15m 44s	4 518 101	4 999 963
Mosaic	**16**	6m 06s	807 603	2 167 861
Abstracts	**220**	27m 21s	233 483	237 771
Accidents	**74**	18m 35s	24 592 869	25 979 244
Chess (kr–k)	**67**	83m 32s	766 235	786 651
DNA Amplification	**204**	4m 31s	79 164	183 121
Kosarak	261	120m 36s	66 385 663	70 250 533
Lotto	**29**	0m 31s	64 970	65 099
Mammals	**76**	25m 23s	99 733	119 461
MCADD	80	121m 02s	2 709 240	2 840 837
Mushroom	**80**	28m 11s	358 369	441 130
Plants	**94**	66m 56s	732 145	1 271 950
Retail	62	121m 46s	8 352 161	8 437 118

interpret the raw BIC and MDL scores as negative log-likelihoods, we see that BIC achieves lower, and hence better, scores. This follows naturally from the larger collections of itemsets that BIC selects; the larger the collection, the more information it provides, and hence the higher the likelihood.

Summary Evaluation

Here we inspect the discovered data summaries in closer detail.

For the *Independent* dataset we see that using MDL the returned summary is empty, i.e., no itemset can improve on the description length of the background model, which is the independence model. In general, MDL does not aim to find any underlying 'true' model, but simply the model that it considers best, given the available data. In this case, however, we see that the discovered model correctly corresponds to the process by which we generated the data. Using BIC, on the other hand, the algorithm discovers 7 itemsets.

Table 6.4: The MDL scores of the discovered models, with respect to the background information $\mathcal{B}$. Shown are the number of itemsets k, the wall clock time, the description length of $\mathcal{C}$, and the description length of the empty collection. (Lower scores are better). Values for k identified as optimal by MDL are given in boldface.

Dataset	k	*time*	MDL$(\mathcal{C}, \mathcal{D}; \mathcal{B})$	MDL$(\emptyset, \mathcal{D}; \mathcal{B})$
Independent	**0**	1m 57s	4 494 243	4 494 243
Markov	**62**	16m 16s	4 519 723	4 999 964
Mosaic	**15**	7m 23s	808 831	2 167 861
Abstracts	**29**	1m 26s	236 522	237 772
Accidents	**75**	19m 35s	24 488 943	25 979 245
Chess (kr–k)	**54**	76m 20s	767 756	786 652
DNA Amplification	**120**	1m 06s	96 967	183 122
Kosarak	262	120m 53s	66 394 995	70 250 534
Lotto	**0**	0m 01s	65 100	65 100
Mammals	**53**	2m 17s	102 127	119 461
MCADD	82	123m 34s	2 700 924	2 840 838
Mushroom	**74**	22m 00s	361 891	441 131
Plants	**91**	63m 09s	734 558	1 271 951
Retail	57	122m 44s	8 357 129	8 437 119

These itemsets have observed frequencies in the data that are slightly higher or lower than their predicted frequencies under the independence assumption. Among the 7 itemsets, the highest absolute difference between those two frequencies is lower than 0.3%. While these itemsets are each significant by themselves, after correcting their p-values to account for Type I error (see further), we find that none of them are statistically significant.

For the *Markov* data, we see that all itemsets in the discovered summary are very small (about half of them of size 2, the rest mostly of size 3 or 4), and they all consist of consecutive items. Since the items in this dataset form a Markov chain, this is very much in accordance with the underlying generating model. In this case, the summaries for BIC and MDL are identical. Knowing that the data is generated by a Markov chain, we can compute the MDL score of the 'best' model. Given that $\mathcal{B}$ contains the singleton frequencies, the model that fully captures the Markov chain contains all 49 itemsets

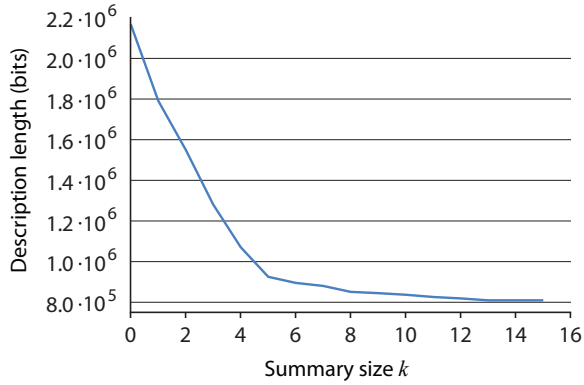


Figure 6.1: Description length of the *Mosaic* dataset as a function of the summary size k. The first five discovered itemsets correspond to the process which generated the dataset. The minimum MDL score is attained at $k = 15$.

of size two containing consecutive items. The description length for this model is 4 511 624, which is close to what the MTV algorithm discovers.

The third synthetic dataset, *Mosaic*, contains 5 itemsets embedded in a 1%-noisy database. The five first itemsets returned, both by BIC and MDL, are exactly those itemsets. These sets are responsible for the better part of the decrease of the score, as can be seen in Figure 6.1, which depicts the description length as a function of the summary size. After these first five highly informative itemsets, a further ten additional itemsets are discovered that help explain the overlap between the itemsets—which cannot be inferred otherwise, because we construct a probabilistic model using itemset frequencies—and while these further itemsets do help in decreasing the score, their effect is much less strong than for the first sets. After discovering fifteen itemsets, the next best itemset does not decrease the log likelihood sufficiently to warrant the MDL model complexity penalty, and the algorithm terminates.

For the *Lotto* dataset, we see that using MDL, our algorithm fails to discover any informative itemsets.[4] Using BIC, we find a summary of 29 itemsets, which are all combinations of 3 numbers, having been drawn between one and three times in the past twenty-odd years. While for each itemset individ-

[4]This is a desired result, assuming of course that the lottery is fair.

ually this is significantly lower than the expected absolute support (which is about 11), when we adjust for Type I error, they are no longer statistically significant. This gives evidence that in our setup, and with these data sizes, BIC may not be strict enough. We additionally ran experiments using only row margins as background knowledge, i.e., using the fact that every transaction contains exactly seven items, the numbers of each lottery draw. Then, according to the maximum entropy background distribution, this implies that every number has exactly the same probability of being picked in a particular draw—namely $1/6$. If our algorithm should find that a ball is drawn significantly more or less often, then it would be included in the model. Further, if there were any meaningful correlation between some of the numbers (positive or negative), these numbers would be included in a reported itemset. Using MDL, our algorithm again finds no itemsets of interest. Using BIC, we find 7 itemsets (after two hours), which are all combinations of five or six numbers, with an absolute support between 2 and 4, which is higher than expected. After applying the Bonferroni adjustment, none of these itemsets turn out to be significant.

In the case of the *DNA Amplification* data, our algorithm finds that the data can be described using 120 itemsets. As this dataset is banded, it contains a lot of structure [Garriga et al., 2008]. Our method correctly discovers these bands, i.e., blocks of consecutive items corresponding to related genes, which lie on the same chromosomes. The first few dozen sets are large, and describe the general structure of the data. Then, as we continue, we start to encounter smaller itemsets, which describe more detailed nuances in the correlations between the genes. Figure 6.2 depicts a detail of the *DNA* dataset, together with a few of the itemsets from the discovered summary.

For the *Chess (kr–k)* and *Mushroom* datasets, we also ran experiments including the row margins, since we know that these datasets originated from categorical data, and hence their margins are fixed. As noted in Section 6.3, this increases the runtime of the algorithm by a factor N. For both BIC and MDL, the algorithm therefore took longer to execute, and was terminated after two hours for both datasets. The summary size in all cases was equal to six. Comparing the discovered summaries revealed that some (but not all) of the itemsets in them also occurred in the summaries that did not use the row margins as background knowledge, which leads us to conclude that the use of row margins as background information for these datasets, does not have a substantial effect on the discovered summaries, at least not for the top-six.

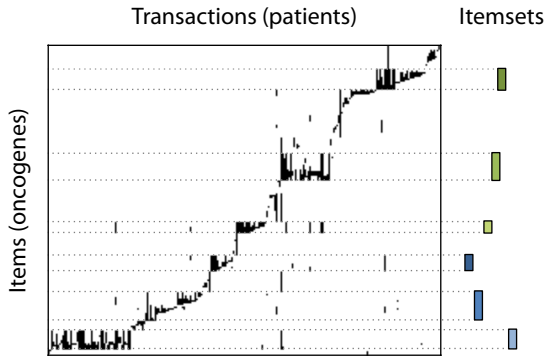


Figure 6.2: Detail of the *DNA Amplification* dataset (left), along with six of the discovered itemsets (right).

The items in the *Mammals* data are European mammals, and the transactions correspond to their occurrences in geographical areas of 50×50 km^2. The discovered itemsets represent sets of mammals that are known to coexist in certain regions. For instance, one such set contains the Eurasian elk (moose), the mountain hare, the Eurasian lynx, and the brown bear, which are all animals living in colder, northern territories. Going back to the transactions of the data, we can also look at the areas where these itemsets occur. Figure 6.3 depicts the geographical areas for a few of the discovered sets. Some clear regions can be recognized, e.g., Scandinavia (for the aforementioned itemset), Central Europe, the Iberian peninsula, or Eastern Europe.

The *Plants* dataset is similar to the *Mammals* data, except than now the plants form transactions, and the items represent the U.S. and Canadian states where they occur. The discovered itemsets, then, are states that exhibit similar vegetation. Naturally, these are bordering states. For instance, some of the first discovered itemsets are {CT, IL, IN, MA, MD, NJ, NY, OH, PA, VA} (North-Eastern states), {AL, AR, GA, KY, LA, MO, MS, NC, SC, TN} (South-Eastern states), and {CO, ID, MT, OR, UT, WA, WY} (North-Western states).

Finally, for the *MCADD* data, we find about 80 itemsets after running the algorithm for two hours. The attributes in this dataset are measured fatty acid concentrations, ratios between these, and some further biochemical parameters, which have all been discretized. The items therefore are attribute-value

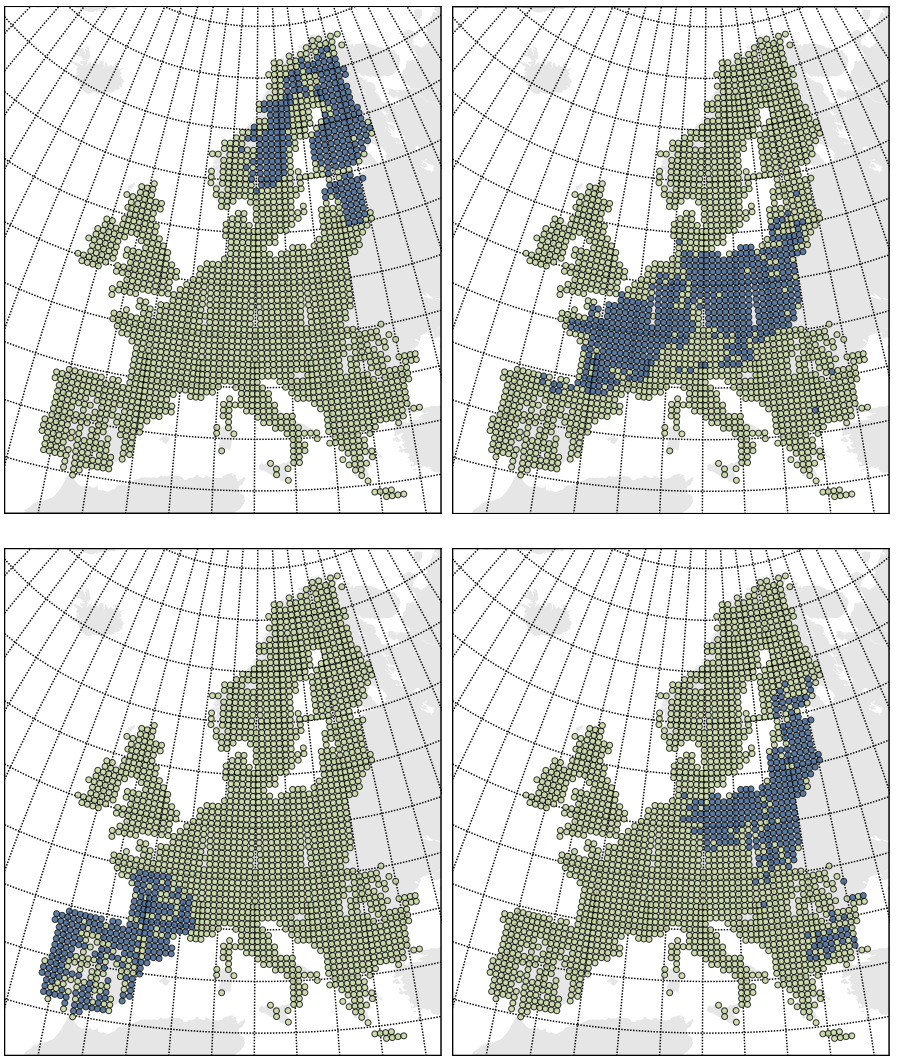


Figure 6.3: For the *Mammals* datasets, for four of the itemsets discovered by our method, we depict the transactions (i.e., locations) that support that itemset. A transaction supports the itemset if all of the mammals identified by the set have been recorded in the data to occur in that area.

pairs. In the discovered summary, we see that the selected itemsets mostly consist of items corresponding to a few known key attributes. Furthermore, we can identify strongly correlated attributes by regarding those combinations of attributes within the itemsets selected in the summary. As an example, one of the first itemsets in the summary corresponds to particular values for attributes $\{MCADD, C8, \frac{C8}{C2}, \frac{C8}{C10}, \frac{C8}{C12}\}$, respectively the class label, an acid, and some of its calculated ratios. This acid and its ratios, and the identified values, are commonly used diagnostic criteria for screening MCADD, and were also discovered in previous in-depth studies [Baumgartner et al., 2005, Van den Bulcke et al., 2011].

Comparison with Other Methods

In Table 6.5, we give the top-10 itemsets in the *Abstracts* dataset, as discovered by our algorithm (using MDL), the method by Kontonasios and De Bie [2010], the compression-based Krimp algorithm [Vreeken et al., 2011], and Tiling [Geerts et al., 2004]. We see that our algorithm discovers recognizable data mining topics such as *support vector machines*, *naive bayes*, and *frequent itemset mining*. Further, there is little overlap between the itemsets, and there is no variations-on-the-same-theme type of redundancy present.

The results of the Information-Theoretic Noisy Tiles algorithm by Kontonasios and De Bie [2010], based on the Information Ratio of tiles, are different from ours, but seem to be more or less similar in quality for this dataset.

The Krimp algorithm does not provide its resulting itemsets in an order, so in order to compare between the different methods, following its MDL approach, we selected the top-10 itemsets from the code table that have the highest usage, and hence, the shortest associated code. From a compression point of view, the items in these sets co-occur often, and thus result in small codes for the itemsets. Arguably, this does not necessarily make them the most interesting, however, and we observe that some rather general terms such as *state [of the] art* or *consider problem* are ranked highly.

Finally, for Tiling we provide the top-10 tiles of at least two items. Without this size restriction, only singletons are returned, which, although objectively covering the largest area individually, are not very informative. Still, the largest discovered tiles are of size two, and contain quite some redundancy, for instance, the top-10 contains only 13 (out of 20) distinct items.

Table 6.5: The top-10 itemsets of the *Abstracts* dataset as discovered by the MTV algorithm (top left), the method of Kontonasios and De Bie [2010] (top right), KRIMP [Vreeken et al., 2011] (bottom left), and Tiling [Geerts et al., 2004] (bottom right).

MTV
support vector machin svm
associ rule mine
nearest neighbor
frequent itemset mine
naiv bay
linear discrimin analysi lda
cluster high dimension
state art
frequent pattern mine algorithm
synthet real

Information-Theoretic Noisy Tiles
support vector machin
effici discov frequent pattern mine algorithm
associ rule mine algorithm database
train learn classifi perform set
frequent itemset
mine high dimensional cluster
synthetic real
time seri
decis tree classifi
problem propos approach experiment result

KRIMP
algorithm experiment result set
demonstr space
larg databas
consid problem
knowledg discoveri
experiment demonstr
rule mine associ databas
algorithm base approach cluster
state art
global local

Tiling
algorithm mine
algorithm base
result set
approach problem
propos method
experiment result
algorithm perform
model base
set method
algorithm gener

Significant Itemsets

Next, we investigate the significance of the itemsets that are included in the summaries we discover, as well as the significance of the itemsets in $\mathcal{F}$ that were not included. To properly compute the p-values of the itemsets, we employ holdout evaluation, as described by Webb [2007]. That is, we equally split each dataset into an exploratory (training) set $\mathcal{D}_e$ and a holdout (test) set $\mathcal{D}_h$, apply our algorithm to the exploratory data, and evaluate the itemsets on the holdout set. Since $\mathcal{D}_e$ contains less data than $\mathcal{D}$, the discovered models tend to contain fewer itemsets (i.e., as there is less data to fit the model on, BIC and MDL both allow for less complex models).

The significance of an itemset X is evaluated as follows. We compute its estimated probability $p^*(X = 1)$ (using either $\mathcal{B}$ or $\mathcal{B}$ and $\mathcal{C}$, consistent with D_e). This is the null hypothesis. Then, we calculate the two-tailed p-value given the observed frequency in the holdout data, $fr(X)$. Let us denote $d = |\mathcal{D}_h| = |\mathcal{D}|/2$, $f = d \cdot fr(X)$, and $p = p^*(X = 1)$. The p-value of X expresses the probability of observing an empirical frequency $q_{\mathcal{D}_h}(X)$ at least as extreme (i.e., improbable) as the observed frequency $fr(X)$, with respect to the model, according to a binomial distribution parametrized by d and p

$$B(d; p)(f) = \binom{d}{f} p^f (1-p)^{(d-f)} .$$

Assuming $fr(X) \geq p^*(X = 1)$, we calculate the two-tailed p-value of X as

$$\begin{aligned} \text{p-value} &= \text{Prob}(q_{\mathcal{D}_h}(X) \geq f \mid p) + \text{Prob}(q_{\mathcal{D}_h}(X) \leq f' \mid p) \\ &= \sum_{i=f}^{d} \binom{d}{i} p^i (1-p)^{d-i} + \sum_{i=0}^{f'} \binom{d}{i} p^i (1-p)^{d-i} \end{aligned}$$

where

$$f' = \max\{f' \in [0, dp) \mid B(d; p)(f') \leq B(d; p)(f)\} .$$

The p-value for the case $fr(X) < p^*(X = 1)$ is defined similarly.

It is expected that the itemsets that our algorithm discovers are significant with respect to the background model. Simply put, if the frequency of an itemset is close to its expected value, it will have a high p-value, and thus it will not be significant. Moreover, it will also have a low heuristic value $h(X)$, and hence it will not greatly improve the model. The following demonstrates this connection between kl (and hence h), and the p-value.

Using Stirling's approximation, we can write the logarithm of the binomial probability $B(d; p)(f)$ as follows.

$$\begin{aligned}\log \binom{d}{f} p^f (1-p)^{d-f} &\approx d \log d - f \log f - (d-f) \log(d-f) \\ &\quad + f \log p + (d-f) \log(1-p) \\ &= -f \log \frac{f}{dp} - (d-f) \log \frac{d-f}{d(1-p)} \\ &= -d \cdot kl(\mathrm{X})\end{aligned}$$

Therefore, when we maximize the heuristic h, we also indirectly minimize the p-value. Note, however, that in our algorithm we do not employ a significance level; we simply let BIC or MDL decide when to stop adding itemsets. Further, we also take the complexity of the itemsets into account.

In Table 6.6 we show the number of significant selected and unselected itemsets. Since we are testing multiple hypotheses, we apply the well-known Bonferroni adjustment to avoid Type I error, i.e., to avoid falsely rejecting a true hypothesis [Shaffer, 1995]. That is, if we were to test, say, 100 true hypotheses at significance level 0.05, then by chance we expect to falsely reject five of them. Therefore, at significance level α, each p-value is compared against the adjusted threshold α / n where n is the number of tests performed.

The first column of Table 6.6 shows the number of itemsets in $\mathcal{C}$ that are significant with respect to the background knowledge $\mathcal{B}$. In general, we see that all itemsets are significant. However, for the *Accidents* dataset, e.g., we find two itemsets that are not significant with respect to $\mathcal{B}$. Nevertheless, they are not redundant in this case; upon inspection, it turns out that these itemsets (say, X_1 and X_2) are subsets of another itemset (say, Y) in $\mathcal{C}$ that was significant. After adding Y to the model, the estimates of X_1 and X_2 change, causing them to become significant with respect to the intermediate model. A similar observation is made for the *Chess (kr–k)* and *Mushroom* datasets. In the second column, we therefore also show the number of itemsets $X_{i+1} \in \mathcal{C}$ that are significant with respect to the previous model $\mathcal{C}_i$. In this case we see that indeed all itemsets are significant, from which we conclude that all itemsets really contribute to the summary, and are not redundant.

Next, we compute the p-values of the itemsets in the candidate set $\mathcal{F}$ that were not included in $\mathcal{C}$. Since for all datasets the candidate set $\mathcal{F}$ is very large, we uniformly sample 1 000 itemsets from $\mathcal{F} \setminus \mathcal{C}$, and compute their p-

values with respect to $p^*_{\mathcal{B},\mathcal{C}}$. We see that for seven of the datasets, there are few significant itemsets, which means that $\mathcal{C}$ captures the data quite well. For two datasets, a few hundred are significant, but still many are not. For the five remaining datasets, however, we see that almost all itemsets are significant. However, this does not necessarily mean that the discovered models are poor. That is, apart from considering the deviation of an itemset's frequency, we also consider its complexity and the complexity of a model as a whole. This means that even if the observed frequency of an itemset is surprising to some degree, it may be too complex to include it; upon inspection, we indeed find that among the sampled itemsets, there tend to be many large ones.

Table 6.6: The number of significant itemsets in the discovered summary $\mathcal{C}$, with respect to the background model $\mathcal{B}$ and each intermediate model $\mathcal{C}_i$, and the number of significant itemsets among 1 000 itemsets sampled from $\mathcal{F} \setminus \mathcal{C}$, denoted by $\mathcal{S}$. The significance level used is 0.05, and the p-values were corrected using the Bonferroni adjustment.

	$X \in \mathcal{C}$			$X \in \mathcal{S}$	
Dataset	*# signif. w.r.t.* $\mathcal{B}$	*# signif. w.r.t.* $\mathcal{C}_i$	$\lvert\mathcal{C}\rvert$	*# signif. w.r.t.* $\mathcal{C}$	$\lvert\mathcal{S}\rvert$
Independent	0	0	**0**	0	1 000
Markov	64	64	**64**	6	1 000
Mosaic	14	14	**14**	0	1 000
Abstracts	12	12	**12**	14	1 000
Accidents	69	71	**71**	883	1 000
Chess (kr–k)	42	42	**43**	37	1 000
DNA Amplification	87	87	**87**	990	1 000
Kosarak	268	268	268	993	1 000
Lotto	0	0	**0**	0	1 000
Mammals	39	39	**39**	986	1 000
MCADD	61	61	61	89	1 000
Mushroom	59	63	**63**	139	1 000
Plants	87	87	**87**	998	1 000
Retail	65	65	65	302	1 000

6.7 Discussion

The approach introduced in this chapter meets several intuitive expectations one might have about summarization, such as succinctness, providing a characteristic description of the data, and having little redundancy. The experiments show that quantitatively we can achieve good BIC or MDL scores with only a handful of itemsets, and that these results are highly qualitative and meaningful; moreover, we can discover them in a relatively short amount of time. In practice, we see that the results using MDL are slightly better than those for BIC; the former tends to be a bit more conservative, in the sense that it does not discover spurious itemsets. Furthermore, using our MDL score, we have more control over the complexity of the summary, since it takes into account not only the summary's size but also the sizes of the itemsets in it.

In this chapter we consider data mining as an iterative process. By starting off with what we already know—our background knowledge—we can identify those patterns that are the most surprising. Simply finding the itemset that is most surprising, is a problem that Hanhijärvi et al. [2009] describe as 'tell me something I don't know'. When we repeat this process, in the end we will have identified a group of itemsets that 'tell me all there is to know' about the data. Clearly, this group strongly overfits the data. This is where the MDL principle provides a solution, as it automatically identifies the most informative group. Hence, we can paraphrase our approach as 'tell me what I need to know'. As such, by our information theoretic approach it can be seen as an instantiation of the general iterative framework recently proposed by De Bie [2011a].

The view that we take here on succinctness and non-redundancy is fairly strict. Arguably, there are settings conceivable where limited redundancy (at the cost of brevity) can give some robustness to a technique, or provide alternative insights by restating facts differently. However, this is not the intention of our approach, and we furthermore argue that our method can perfectly be complemented by techniques such as redescription mining [Zaki and Ramakrishnan, 2005].

Data mining is not only an iterative process, but also an interactive one. The MTV algorithm simply returns a set of patterns to the user, with respect to his or her background knowledge. However, in practice we might want to dynamically guide the exploration and summarization process. That is, we may want to add or remove certain itemsets, next let the algorithm add some

more itemsets, etc. Our algorithm can easily be embedded into an interactive environment. Deleting an itemset from a summary is quite straightforward; we just collapse the transaction partition, and re-run the iterative scaling algorithm to discover the parameters of the simplified model.

There are still some further improvements possible for our method. For instance, one problem setting in which our method is applicable, is that of finding the best specialization of a given itemset X. That is, to identify the superset Y of X that provides the best score $s(\mathcal{C} \cup \{Y\})$. This setup allows experts to interactively discover interesting itemsets.

The experiments showed that we discover high-quality summaries, and that we can efficiently compute the maximum entropy model for a given collection of itemsets in practice—even though this is a **NP**-hard problem in general. In specific cases, however, there is room for further optimization. For computational reasons, we split up the distribution into smaller groups, by restricting the number of items per group. Exactly modeling a long Markov chain, for instance, is then not possible, as only parts up to n items will be modeled. While our model can easily describe a complete Markov chain, computing it may prove to be difficult in practice. It would be interesting to see then, how modeling techniques such as decomposition, e.g., using junction trees [Cowell et al., 1999], could be combined with our methods.

We see several further possibilities for improving upon our current, unoptimized, implementation. For example, massive parallelization can be applied to the search for the most informative itemset. This requires computing itemset probabilities with respect to a single model, for many candidates, which can easily be done in parallel. This can also benefit pruning in this phase, since we can use the maximum heuristic value over all processes. Another option is to simplify the computation of the row margin distribution, which adds a factor N to the algorithm's runtime. Rather than computing the probability that a transaction contains a certain number of items, we could group these probabilities into a coarser granularity, to reduce this factor.

6.8 Conclusions

We introduced a well-founded method for iteratively mining non-redundant collections of itemsets that form high-quality summaries of transaction data. By employing the Maximum Entropy principle, we obtain unbiased probabilistic models of the data, through which we can identify the most informa-

tive itemsets, and subsequently iteratively build succinct summaries of the data by updating our model accordingly. As such, unlike static interestingness models, our approach does not return patterns that are redundant with regard to what we have learned, or already knew as background knowledge, and hence the result is kept succinct and maximally informative.

To this end, we presented the MTV algorithm for mining informative summaries, which can either mine the top-k most informative itemsets, or by employing the Bayesian Information Criterion (BIC) or the Minimum Description Length (MDL) principle we can automatically identify that set of itemsets that together provide the best summarization of the data. Hence, informally said, our parameter-free method 'tells you what you need to know' about the data.

Although in the general case modeling by Maximum Entropy is **NP**-hard, we showed that in our case we can do so efficiently, using Quick Inclusion-Exclusion. Furthermore, MTV is a one-phase algorithm; rather than picking itemsets from a pre-materialized user-provided candidate set, it calculates on-the-fly the supports for only those itemsets that stand a chance of being included in the summary.

Experiments show that we discover succinct summaries, which correctly identify important patterns in the data. The resulting models attain high log-likelihoods using few itemsets, are easy to interpret, contain significant itemsets with low p-values, and for a wide range of datasets are shown to give highly intuitive results.

Chapter 7

Conclusions

This dissertation explored how we can summarize data using local patterns, to get an overview of and gain insight into a given dataset. While existing techniques can efficiently mine interesting patterns, they often return far too many of them for a user to handle. The reason for this lies in the fact that such techniques usually consider the interestingness of patterns on an individual basis. This results in redundancy, since they do not take into account how these patterns relate to each other as a whole.

We focussed on three important notions a data mining method should adhere to in order for it to be categorized as a proper summarization method: informativeness, conciseness, and interpretability. To this end, we relied heavily on information theory to characterize the interestingness of (sets of) patterns, as well as to incorporate the data miner's background knowledge. By employing patterns such as itemsets and attribute sets to describe the data, we ensured that the end result remains intuitive and interpretable. Finally, to guarantee the conciseness of a summary, we made use of information-theoretic techniques to reduce redundancy, as well as model selection techniques to keep the results succinct and manageable.

The main contributions of this thesis can be summarized as follows.

- In Chapter 3 we proposed an information-theoretic approach to mine dependence rules between attribute sets. Based on some theoretical properties, we were able to reduce redundancy in the set of results, and effectively reduce the size of the output by several orders of magnitude.

- In Chapter 4 we provided a simple technique to create a high-level overview of a dataset. This was achieved by clustering the attributes into correlated groups. For each group, a small distribution describes the corresponding part of the data. As such, an attribute clustering can be used as a simple yet effective approximation of the data.
- In Chapter 5 we studied the usage of intuitive statistics such as row and column margins, lazarus counts, and transaction bounds, to construct a background model for ranking itemsets. This background model is based on the Maximum Entropy principle, and we presented algorithms to construct and query it in polynomial time.
- In Chapter 6 we introduced a novel algorithm to mine a collection of itemsets as a global model for binary data. The proposed algorithm models the data using a Maximum Entropy distribution which is built based on the constraints imposed by a collection of itemsets. To discover the best collection of itemsets, we employed the Bayesian Information Criterion and the Minimum Description Length principle.

While the results obtained in this dissertation are encouraging, this is not the end of the road. The very nature of exploratory data mining offers numerous possible directions for future research. For one, we focussed on summarizing binary and categorical data. The extension of the proposed techniques to other data types such as ordinal or numeric data, or data of a different structure, such as graphs or time series, is evident. Also the possibility of using other patterns besides itemsets or attribute sets to construct data summaries, is worth investigating. Further, while we did not concentrate on it in detail in this dissertation, interactivity is an important aspect of exploratory data mining that should be taken into consideration, and which requires the development of dynamic algorithms that can anticipate and deal with a user's changing interests. Finally, the application of summaries beyond descriptive data mining, for instance, applying attribute clustering (Chapter 4) or the MTV algorithm (Chapter 6) for classification, may yield interesting new data mining techniques.

Nederlandse Samenvatting

Dankzij de technologische vooruitgang is het sinds enkele decennia mogelijk om goedkoop enorme hoeveelheden data van diverse soorten te vergaren en op te slaan; bijvoorbeeld, wetenschappelijke observaties, commerciële bestanden, medische data, overheidsdatabanken, multimedia, etc. Het verzamelen van zulke data is essentiëel om inzicht te krijgen in een probleem, om een bepaald fenomeen te bestuderen, of om weloverwogen beslissingen te kunnen nemen. Het is tevens een integraal onderdeel van de wetenschappelijke methode. Tegenwoordig hebben we echter zulke grote hoeveelheden data tot onze beschikking, dat het extraheren van informatie hieruit een niet-triviale taak geworden is. De analyse van zulke data met de hand is praktisch niet doenbaar. Deze situatie geeft aanleiding tot de noodzaak voor computationele technieken die nuttige, zinvolle informatie uit databases extraheren, om ze goed te benutten.

'Knowledge discovery from data' (KDD) heeft zich als gevolg hiervan in een snel tempo ontwikkeld tot een belangrijk onderzoeksgebied. Vooral het deelgebied data mining, een essentiëel onderdeel in het knowledge discovery proces, heeft veel momentum gekregen. Data mining, dat zich bevindt in de doorsnede van de statistiek, artificiële intelligentie, en database onderzoek, wordt door Hand et al. [2001] als volgt gedefinieerd.

> Data mining is de analyse van (vaak grote) observationele datasets om onverwachte verbanden te vinden en om de data samen te vatten op nieuwe manieren die zowel begrijpelijk als nuttig zijn voor de eigenaar van data.

Een aantal belangrijke (zij het vage) termen komen voor in deze definitie. De term *verbanden* is zeer breed, en kan verwijzen naar eender welke vorm

van regelmaat of structuur in de data, variërend van lokale patronen tot classificatiemodellen of clusteringen. Het sleutelwoord *onverwachte* is cruciaal, aangezien het doel van data mining het ontdekken van relaties is, die interessant, verrassend, of tot nog toe onbekend zijn. Hierbij wordt er inherent van uitgegaan dat de data miner een aantal verwachtingen heeft over de data, al dan niet in expliciete vorm. Elke afwijking van deze verwachting wordt interessant geacht. Als de data volledig aan de verwachtingen van de gebruiker zou voldoen, is deze immers vrij saai. We kunnen deze verwachting zien als de achtergrondkennis van de gebruiker.

Het *samenvatten* van data betekent de kenmerken ervan te presenteren aan de data miner, op een beknopte manier, zodat deze samenvatting een klein maar informatief stuk informatie vormt. Idealiter bevat een samenvatting enkel de gegevens die de gebruiker nodig heeft, niets meer en niets minder. De resultaten van bestaande data mining technieken zijn vaak echter zeer complex en gedetailleerd, of bevatten redundante informatie. Bij frequent itemset mining, bijvoorbeeld, kan de gebruiker overspoeld worden met talloze patronen. Ten slotte moet het resultaat van een data mining methode *begrijpelijk* zijn, wat betekent dat het intuïtief moet zijn voor een analist om te begrijpen wat het resultaat betekent, zodat hij of zij vervolgens in staat is om de opgedane kennis goed te kunnen benutten.

Data mining technieken kunnen grofweg worden onderverdeeld in twee categorieën: *lokale* en *globale*. Globale technieken streven ernaar een model van de data in haar geheel te construeren. Dergelijke modellen vatten de data meestal niet perfect, maar trachten in plaats daarvan de algemene tendensen ervan weer te geven, om ze te beschrijven of te begrijpen, of om voorspellingen te kunnen doen. Voorbeelden hiervan zijn clustering, classificatie en regressie, grafisch modellen, of het fitten van de parameters van een Gaussiaanse distributie. Lokale technieken beschrijven bepaalde aspecten van de data die op een of andere manier interessant kunnen zijn. Dergelijke lokale structuren (bijvoorbeeld frequent itemsets), beschrijven slechts een deel van de data, in plaats van ze in haar geheel te modelleren. Een voordeel is dat ze vaak gemakkelijk te interpreteren zijn, en niet moeilijk om de ontdekken. In de praktijk is het echter bijna onmogelijk om *alle* interessante patronen in een dataset te beschouwen, om de eenvoudige reden dat er wellicht veel te veel zijn om te verwerken. Het is algemeen bekend dat local pattern mining in de praktijk ofwel vergezeld dient te worden van een pruning, ranking, of filtering stap, die de resultaten drastisch inkort zonder aan informativiteit in

te boeten; of men moet direct sets van patronen als een geheel minen, door te beschouwen hoe ze zich tot elkaar verhouden. Het onderscheid tussen lokale en globale data mining technieken is echter niet altijd even scherp.

Het doel en onderwerp van dit proefschrift is te onderzoeken hoe lokale patronen gebruikt kunnen worden om data samen te vatten. Daartoe zetten we een aantal belangrijke concepten voorop waaraan zulke samenvattingen moeten voldoen.

- **Informativiteit** Het spreekt voor zich dat een samenvatting zinvolle en nuttige informatie aan een gebruiker moet communiceren; een samenvatting moet belangrijke structuren die in de data aanwezig zijn weergeven. Informatie theorie speelt daarom een belangrijke rol in dit proefschrift om interessantheid te meten. Verder richten we ons ook op het in beschouwing nemen van de achtergrondkennis van de data miner, want wat interessant is voor de ene persoon is dat niet noodzakelijk voor de andere.

- **Beknoptheid** Een samenvatting moet uiteraard beknopt zijn zodat een gebruiker ze kan overzien en gebruiken. Deze beknoptheid kan simpelweg verwijzen naar de grootte van de samenvatting, maar kan ook verwijzen naar de complexiteit ervan.

- **Interpreteerbaarheid** Een beknopte black box methode die nauwkeurig een gegeven dataset beschrijft, laat ons niet toe om de data ook beter te begrijpen. Daarom nemen we ook interpreteerbaarheid op als een vereiste. Lokale patronen zijn in het algemeen eenvoudig te begrijpen. In dit proefschrift worden twee soorten patronen beschouwd: itemsets en attribute sets, en dit voor binaire en categorische data.

Een samenvatting kan gezien worden als een model van de data, hoewel deze niet noodzakelijk query-baar is, of gepresenteerd is in een mathematische vorm. De samenvattingen in dit proefschrift moeten worden gecategoriseerd als *exploratory data mining*, onze bedoeling is immers onze kennis van een gegeven dataset waar we weinig over weten uit te breiden. Dit is een inherent onnauwkeurig probleem met geen duidelijk omschreven doel of één enkele correcte oplossing. Het doel van dit proefschrift is daarom niet het ontwikkelen van een allesomvattende samenvattingsmethode. In plaats daarvan beschouwen we verschillende methoden, omdat we in de praktijk

kunnen worden geconfronteerd met verschillende soorten data en toepassingen. Het gebruik van verschillende manieren om naar data te kijken, met verschillende patronen, criteria, etc, kan meer inzicht geven dan het gebruik van een enkele techniek.

Bijdragen van het Proefschrift

Dit proefschrift bestaat uit zeven hoofdstukken. Hoofdstukken 3 tot 6 bevatten de voornaamste bijdragen, en zijn gebaseerd op eerder gepubliceerd werk. De bijdragen kunnen als volgt samengevat worden.

Hoofdstuk 3 presenteert een methode voor het ontdekken van dependency rules tussen sets van attributen, met behulp van informatie-theoretische measures. We onderzoeken technieken om redundantie in de resultaten te verminderen, en presenteren een efficiënt algoritme om deze regels te ontdekken.

Hoofdstuk 4 geeft een techniek om een high-level overzicht van een dataset te creëren, door attributen in gecorreleerd groepen te clusteren. Voor elke attribute cluster geeft een kleine distributie een beschrijving van het desbetreffende deel van de data. Deze groeperingen kunnen vervolgens worden gebruikt als een benaderend model voor de data. Voor het vinden van goede attribute clusterings gebruiken we het Minimum Description Length principe.

Hoofdstuk 5 bestudeert het gebruik van intuïtieve statistieken om een achtergrondmodel te bouwen om itemsets te rangschikken. Deze statistieken zijn rij- en kolommarges, lazarus counts, en transactiegrenzen. Op basis van deze statistieken presenteren we een maximum entropy model, dat efficiënt geconstrueerd en bevraagd kan worden in polynomiale tijd, en waartegen itemsets gerangschikt worden.

Hoofdstuk 6 introduceert een algoritme dat een verzameling van itemsets zoekt als een globaal model voor data. Het voorgestelde algoritme modelleert de data met behulp van een maximum entropy distributie die geconstrueerd wordt op basis van een collectie itemsets. Technieken uit Hoofdstuk 4 om efficiënte berekening te garanderen, en uit Hoofdstuk 5 om achtergrondkennis te beschouwen, zijn opgenomen in het algoritme. Om de beste verzameling van itemsets die een bepaalde dataset samenvat te ontdekken, maken we gebruik van het Bayesian Information Criterion en het Minimum Description Length principe.

Bibliography

C. C. Aggarwal and P. S. Yu. A new framework for itemset generation. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 18–24. ACM Press, 1998.

R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), Santiago de Chile, Chile*, pages 487–499, 1994.

R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22(2):207–216, 1993.

W. H. Au, K. C. C. Chan, A. K. C. Wong, and Y. Wang. Attribute clustering for grouping, selection, and classification of gene expression data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(2):83–101, 2005.

J. L. Balcázar. Minimum-size bases of association rules. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Antwerp, Belgium*, volume 5211 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2008.

C. Baumgartner, C. Böhm, and D. Baumgartner. Modelling of classification rules on metabolic patterns including machine learning and expert knowledge. *Biomedical Informatics*, 38(2):89–98, 2005.

R. J. Bayardo. Efficiently mining long patterns from databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Seattle, WA*, pages 85–93. ACM New York, NY, USA, 1998.

R. J. Bayardo, B. Goethals, and M. J. Zaki. Fimi '04. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK*, volume 126, 2004.

J.-F. Boulicaut, A. Bykowski, and C. Rigotti. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery*, 7(1):5–22, 2003.

T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets. Using association rules for product assortment decisions: A case study. In *Proceedings of the 5th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, pages 254–260, 1999.

S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In J. Peckham, editor, *Proceedings of the ACM International Conference on Management of Data (SIGMOD), Tucson, AZ*, pages 265–276. ACM Press, 1997.

B. Bringmann and A. Zimmermann. The chosen few: On identifying valuable patterns. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM), Omaha, NE*, pages 63–72. IEEE, 2007.

B. Bringmann and A. Zimmermann. One in a million: picking the right patterns. *Knowledge and Information Systems (KAIS)*, 18(1):61–81, 2009.

T. Calders and B. Goethals. Quick inclusion-exclusion. In *Proceedings of the 4th International Workshop on Knowledge Discovery in Inductive Databases (KDID), Porto, Portugal*, volume 3933, pages 86–103. Springer, 2005.

T. Calders and B. Goethals. Non-derivable itemset mining. *Data Mining and Knowledge Discovery*, 14(1):171–206, 2007.

D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-associations. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Seattle, WA*, pages 79–88. ACM, 2004.

V. Chandola and V. Kumar. Summarization – compressing data into an informative representation. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM), Houston, TX*, pages 98–105. IEEE, 2005.

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.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience New York, 2006.

R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Statistics for Engineering and Information Science. Springer-Verlag, 1999.

I. Csiszár. I-divergence geometry of probability distributions and minimization problems. *The Annals of Probability*, 3(1):146–158, 1975.

M. M. Dalkilic and E. L. Robertson. Information dependencies. In *Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 245–253, 2000.

J. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):1470–1480, 1972.

G. Das, H. Mannila, and P. Ronkainen. Similarity of attributes by external probes. In *Proceedings of the 3rd ACM International Conference on Knowledge Discovery and Data Mining (KDD), Newport Beach, CA*, pages 23–29. ACM, 1997.

B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

T. De Bie. An information theoretic framework for data mining. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*. ACM, 2011a.

T. De Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Mining and Knowledge Discovery*, pages 1–40, 2011b.

I. S. Dhillon, S. Mallela, and R. Kumar. A divisive information theoretic feature clustering algorithm for text classification. *Journal of Machine Learning Research*, 3:1265–1287, 2003.

A. Frank and A. Asuncion. UCI machine learning repository. http://archive.ics.uci.edu/ml/, 2010.

A. Gallo, N. Cristianini, and T. De Bie. MINI: Mining informative non-redundant itemsets. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Warsaw, Poland*, pages 438–445. Springer, 2007.

G. C. Garriga, E. Junttila, and H. Mannila. Banded structure in binary matrices. In *Proceedings of the 14th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Las Vegas, NV*, pages 292–300. ACM New York, NY, USA, 2008.

F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. In *Proceedings of Discovery Science*, pages 278–289, 2004.

L. Geng and H. J. Hamilton. Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, 38(3):9, 2006.

K. Geurts, G. Wets, T. Brijs, and K. Vanhoof. Profiling high frequency accident locations using association rules. In *Proceedings of the 82nd Annual Transportation Research Board, Washington DC. (USA)*, page 18pp, 2003.

A. Gionis, H. Mannila, T. Mielikäinen, and P. Tsaparas. Assessing data mining results via swap randomization. *ACM Transactions on Knowledge Discovery from Data*, 1(3), 2007.

B. Goethals. Survey on frequent pattern mining. http://www.adrem.ua.ac.be/bibrem/pubs/fpm_survey.pdf, 2003.

B. Goethals and M. Zaki. Frequent itemset mining implementations repository. http://fimi.ua.ac.be/, 2003.

K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In *Proceedings of the 1st IEEE International Conference on Data Mining (ICDM), San Jose, CA*, pages 163–170, 2001.

P. D. Grünwald. Minimum description length tutorial. In P. D. Grünwald and I. J. Myung, editors, *Advances in Minimum Description Length*. MIT Press, 2005.

P. D. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *ACM SIGMOD Record*, 29(2):1–12, 2000.

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.

D. J. Hand, H. Mannila, and P. Smyth. *Principles of data mining*. MIT press, 2001.

S. Hanhijärvi, M. Ojala, N. Vuokko, K. Puolamäki, N. Tatti, and H. Mannila. Tell me something I don't know: randomization strategies for iterative data mining. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Paris, France*, pages 379–388, 2009.

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 13th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Jose, CA*, pages 350–359. ACM, 2007.

H. Heikinheimo, J. Vreeken, A. Siebes, and H. Mannila. Low-entropy set selection. In *Proceedings of the 9th SIAM International Conference on Data Mining (SDM), Sparks, NV*, pages 569–579. SIAM, 2009.

Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999a.

Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Tane homepage. http://www.cs.helsinki.fi/research/fdk/datamining/tane/, 1999b.

S. Jaroszewicz and T. Scheffer. Fast discovery of unexpected patterns in data, relative to a bayesian network. In *Proceedings of the 11th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Chicago, IL*, pages 118–127, New York, NY, USA, 2005. ACM.

S. Jaroszewicz and D. A. Simovici. Pruning redundant association rules using maximum entropy principle. In *Proceedings of the 6th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Taipei, Taiwan*, pages 135–147, 2002.

S. Jaroszewicz and D. A. Simovici. Interestingness of frequent itemsets using bayesian networks as background knowledge. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Seattle, WA*, pages 178–186, New York, NY, USA, 2004. ACM.

E. T. Jaynes. On the rationale of maximum-entropy methods. *Proceedings of the IEEE*, 70(9):939–952, 1982.

S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Statistical Physics*, 34(5):975–986, 1984.

J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1):129–149, 1995.

A. J. Knobbe and E. K. Y. Ho. Maximally informative *k*-itemsets and their efficient discovery. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Philadelphia, PA*, pages 237–244. ACM, 2006a.

A. J. Knobbe and E. K. Y. Ho. Pattern teams. In *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Berlin, Germany*, volume 4213, pages 577–584. Springer, 2006b.

R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDD-Cup 2000 organizers' report: Peeling the onion. *ACM SIGKDD Explorations*, 2(2): 86–98, 2000. http://www.ecn.purdue.edu/KDDCUP.

K.-N. Kontonasios and T. De Bie. An information-theoretic approach to finding noisy tiles in binary databases. In *Proceedings of the 10th SIAM International Conference on Data Mining (SDM), Columbus, OH*, pages 153–164. SIAM, 2010.

M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, 1993.

M. Mampaey. Mining non-redundant information-theoretic dependencies between itemsets. In *Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery (DaWaK), Bilbao, Spain*, volume 6263 of *LNCS*, pages 130–141. Springer, 2010.

M. Mampaey and J. Vreeken. Summarising data by clustering items. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Barcelona, Spain*, pages 321–336. Springer-Verlag, 2010.

M. Mampaey, N. Tatti, and J. Vreeken. Tell me what I need to know: Succinctly summarizing data with itemsets. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, pages 573–581. ACM, 2011.

H. Mannila and E. Terzi. Nestedness and segmented nestedness. In *Proceedings of the 13th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Jose, CA*, pages 480–489. ACM, 2007.

H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *KDD-94: AAAI workshop on Knowledge Discovery in Databases*, pages 181–192, 1994.

R. Meo. Theory of dependence values. *ACM Transactions on Database Systems (TODS)*, 25(3):380–406, 2000.

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.

S. Myllykangas, J. Himberg, T. Böhling, B. Nagy, J. Hollmén, and S. Knuutila. Dna copy number amplification profiling of human neoplasms. *Oncogene*, 25(55):7324–7332, 2006.

S. Nijssen, T. Guns, and L. De Raedt. Correlated itemset mining in ROC space: a constraint programming approach. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Paris, France*, pages 647–656. Springer, 2009.

N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of the 7th International Conference on Database Theory (ICDT), Jerusalem, Israel*, pages 398–416. Springer, 1999.

R. Pensa, C. Robardet, and J.-F. Boulicaut. A bi-clustering framework for categorical data. In *Proceedings of the 9th European Conference on Principles and*

Practice of Knowledge Discovery in Databases (PKDD), Porto, Portugal, pages 643–650. Springer-Verlag, 2005.

G. Rasch. *Probabilistic Models for Some Intelligence and Attainment Tests*. Danmarks paedagogiske Institut, 1960.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14(1):465–471, 1978.

J. Rissanen. *Information and complexity in statistical modeling*. Springer Verlag, 2007.

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6 (2):461–464, 1978.

J. P. Shaffer. Multiple hypothesis testing. *The Annual Review of Psychology*, 46 (1):561–584, 1995.

C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

A. Siebes and R. Kersten. A structure function for transaction data. In *Proceedings of the 11th SIAM International Conference on Data Mining (SDM), Mesa, AZ*. SIAM, 2011.

A. Siebes, J. Vreeken, and M. van Leeuwen. Item sets that compress. In *Proceedings of the 6th SIAM International Conference on Data Mining (SDM), Bethesda, MD*, pages 393–404. SIAM, 2006.

R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. *ACM SIGMOD Record*, 25(2):1–12, 1996.

P. N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Edmonton, Alberta*, pages 32–41. ACM New York, NY, USA, 2002.

P. N. Tan, V. Kumar, and J. Srivastava. Selecting the right objective measure for association analysis. *Information Systems*, 29(4):293–313, 2004.

N. Tatti. Computational complexity of queries based on itemsets. *Information Processing Letters*, 98(5):183–187, 2006a.

N. Tatti. Safe projections of binary data sets. *Acta Informatica*, 42(8–9):617–638, 2006b.

N. Tatti. Maximum entropy based significance of itemsets. *Knowledge and Information Systems (KAIS)*, 17(1):57–77, 2008.

N. Tatti. Probably the best itemsets. In *Proceedings of the 16th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, DC*, pages 293–302. ACM, 2010.

N. Tatti and H. Heikinheimo. Decomposable families of itemsets. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Antwerp, Belgium*, pages 472–487, 2008.

N. Tatti and M. Mampaey. Using background knowledge to rank itemsets. *Data Mining and Knowledge Discovery*, 21(2):293–309, 2010.

N. Tatti and J. Vreeken. Finding good itemsets by packing data. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM), Pisa, Italy*, 2008.

T. Uno, M. Kiyomi, and H. Arimura. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI), Brighton, UK*, 2004.

T. Van den Bulcke, P. Vanden Broucke, V. Van Hoof, K. Wouters, S. Vanden Broucke, G. Smits, E. Smits, S. Proesmans, T. Van Genechten, and F. Eyskens. Data mining methods for classification of Medium-Chain Acyl-CoA dehydrogenase deficiency (MCADD) using non-derivatized tandem MS neonatal screening data. *Biomedical Informatics*, 44(2):319–325, 2011.

N. K. Vereshchagin and P. M. B. Vitanyi. Kolmogorov's structure functions and model selection. *IEEE Transactions on Information Theory*, 50(12):3265–3290, 2004.

J. Vreeken, M. van Leeuwen, and A. Siebes. Preserving privacy through data generation. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM), Omaha, NE*, pages 685–690. IEEE, 2007.

J. Vreeken, M. van Leeuwen, and A. Siebes. Krimp: Mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

C. S. Wallace. *Statistical and inductive inference by minimum message length*. Springer-Verlag, 2005.

C. Wang and S. Parthasarathy. Summarizing itemset patterns using probabilistic models. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Philadelphia, PA*, pages 730–735, 2006.

J. Wang and G. Karypis. SUMMARY: Efficiently summarizing transactions for clustering. In *Proceedings of the 4th IEEE International Conference on Data Mining (ICDM), Brighton, UK*, pages 241–248. IEEE, 2004.

G. I. Webb. Discovering significant patterns. *Machine Learning*, 68(1):1–33, 2007.

G. I. Webb. Self-sufficient itemsets: An approach to screening potentially interesting associations between items. *ACM Transactions on Knowledge Discovery from Data*, 4(1), 2010.

T. Wu, Y. Chen, and J. Han. Association mining in large databases: A re-examination of its measures. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Warsaw, Poland*, pages 621–628, 2007.

X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing itemset patterns: A profile-based approach. In *Proceedings of the 11th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Chicago, IL*, pages 314–323. ACM, 2005.

M. J. Zaki. Generating non-redundant association rules. In *Proceedings of the 6th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Boston, MA*, pages 34–43, 2000.

M. J. Zaki and K. Gouda. Fast vertical mining using diffsets. In *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, DC*, pages 326–335. ACM, 2003.

M. J. Zaki and N. Ramakrishnan. Reasoning about sets using redescription mining. In *Proceedings of the 11th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Chicago, IL*, pages 364–373. ACM, 2005.

M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proceedings of the 13th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Jose, CA*, volume 20, 1997.

Index