TKK Dissertations in Information and Computer Science

Espoo 2010 TKK-ICS-D15

EXTENDING DATA MINING TECHNIQUES FOR FREQUENT PATTERN DISCOVERY

Trees, low-entropy sets, and crossmining

Hannes Heikinheimo

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Faculty of Information and Natural Sciences for public examination and debate in Auditorium E at the Aalto University School of Science and Technology (Espoo, Finland) on the 23rd of January, 2010, at 12 noon.

Aalto University School of Science and Technology
Faculty of Information and Natural Sciences
Department of Information and Computer Science

Aalto-yliopiston teknillinen korkeakoulu
Informaatio- ja luonnontieteiden tiedekunta
Tietojenkäsittelytieteen laitos

Distribution:
Aalto University School of Science and Technology
Faculty of Information and Natural Sciences
Department of Information and Computer Science
PO Box 15400
FI-00076 Aalto
FINLAND
URL: http://ics.tkk.fi
Tel. +358-9-470 01
Fax +358-9-470 23277
E-mail: series@ics.tkk.fi

ISBN 978-952-60-3003-6 (Print)
ISBN 978-952-60-3004-3 (Online)
ISSN 1797-5050 (Print)
ISSN 1797-5069 (Online)
URL: http://lib.tkk.fi/Diss/2010/isbn9789526030043

Monikko Oy/Multiprint
Espoo 2010

Heikinheimo, H. (2010): **Extending data mining techniques for frequent pattern discovery: trees, low-entropy sets, and crossmining**. Doctoral thesis, Aalto University School of Science and Technology, TKK Dissertations in Information and Computer Science, TKK-ICS-D15, Espoo, Finland.

Keywords: data analysis, frequent patterns, trees, entropy, minimum description length, pattern selection, clustering, mining mixed data types

Abstract

The idea of frequent pattern discovery is to find frequently occurring events in large databases. Such data mining techniques can be useful in various domains. For instance, in recommendation and e-commerce systems frequently occurring product purchase combinations are essential in user preference modeling. In the ecological domain, patterns of frequently occurring groups of species can be used to reveal insight into species interaction dynamics.

Over the past few years, most frequent pattern mining research has concentrated on efficiency (speed) of mining algorithms. However, it has been argued within the community that while efficiency of the mining task is no longer a bottleneck, there is still an urgent need for methods that derive compact, yet high quality results with good application properties. The aim of this thesis is to address this need.

The first part of the thesis discusses a new type of tree pattern class for expressing hierarchies of general and more specific attributes in unstructured binary data. The new pattern class is shown to have advantageous properties, and to discover relationships in data that cannot be expressed alone with the more traditional frequent itemset or association rule patterns.

The second and third parts of the thesis discuss the use of entropy as a score measure for frequent pattern mining. A new pattern class is defined, *low-entropy sets*, which allow to express more general types of occurrence structure than with frequent itemsets. The concept can also be easily applied to tree types of pattern. Furthermore, by applying minimum description length in pattern selection for low-entropy sets it is shown experimentally that in most cases the collections of selected patterns are much smaller than by using frequent itemsets.

The fourth part of the thesis examines the idea of *crossmining* itemsets, that is, relating itemsets to numerical variables in a database of mixed data types. The problem is formally defined and turns out to be NP-hard, although it is approximately solvable within a constant-factor of the optimum solution. Experiments show that the algorithm finds itemsets that convey structure in both the binary and the numerical part of the data.

Tiivistelmä

Usein toistuvien hahmojen etsintä on tiedonlouhinnan osa-alue, jolla on monenlaisia sovellusalueita. Esimerkkinä tästä ovat mm. sähköiset verkkokauppaja ostosuositusjärjestelmät, joissa usein ostettuja tuotekombinaatioita voidaan käyttää asiakkaiden mieltymysten mallintamiseen. Ekologiassa taas usein toistuvat lajien esiintymiskombinaatiot saattavat paljastaa tietoa lajien interaktion dynamiikasta.

Menneinä vuosina usein toistuvien hahmojen etsintä on tieteenä lähinnä keskittynyt tehokkaiden (nopeiden) louhinta-algoritmien suunnitteluun. Vaikka louhinnan tehokkuus ei enää muodostakaan menetelmien pullonkaulaa, tiedonlouhinnan yhteisön sisällä on esitetty, että monilla sovellusalueilla olisi edelleen tarvetta uusille menetelmille, jotka kykenisivät paremmin tuottamaan pieniä mutta ominaisuuksiltaan korkealaatuisia hahmojoukkoja. Tämän väitöskirjatyön tarkoituksena on suunnata huomio tähän tarpeeseen.

Työn ensimmäisessä osassa esitellään uudenlainen puutyyppinen hahmoluokka, jolla on mahdollista esittää attribuuttien hierarkioita binääriaineistossa niiden yleisyyden perusteella. Työssä osoitetaan, että tällä hahmoluokalla on hyödyllisiä ominaisuuksia. Sillä pystytään ilmaisemaan attribuuttien interaktiota tavalla, joka ei ole mahdollista kattavien joukkojen tai assosiaatiosääntöjen avulla.

Työn toisessa ja kolmannessa osassa keskustellaan entropian käytöstä mallinarvostusfunktiona usein toistuvien hahmojen etsinnässä. Tähän liittyen työ määrittelee uuden hahmoluokan, matalan entropian joukot, joilla on mahdollista ilmaista yleisempiä interaktiota kuin tavallisilla kattavilla joukoilla. Entropiaa on myös helppo soveltaa puutyyppisissä hahmoissa. Lisäksi työssä näytetään kokeellisesti, että MDL-periaatteen mukaisen hahmonvalinnan soveltaminen matalan entropian joukkoihin johtaa usein paljon pienempiin hahmojen loppujoukkoihin kuin mitä kattavilla joukoilla.

Työn neljännessä osassa tutkitaan kattavien joukkojen louhintaa hyödyntäen aineiston numeerisia attribuutteja. Ongelmalle muotoillaan formaali määritelmä ja todetaan, että sen optimaalinen ratkaiseminen on kompleksisuudeltaan NP-kova ongelma. Sille on kuitenkin olemassa vakiokertoiminen approksimaatioalgoritmi. Kokeet oikealla aineistolla osoittavat, että menetelmä löytää rakennetta sekä aineiston binäärisestä että numeerisesta osiosta.

Contents

Acknowledgements

During the last four years I have been a researcher at the Helsinki Institute for Information Technology (HIIT) as well as the Department of Information and Computer Science at Helsinki University of Technology (TKK). I have also been a doctoral student both at TKK and at the Graduate School in Computational Biology, Bioinformatics, and Biometry (Combi). I thank these organizations and their staff for providing the necessary facilities and resources to allow me to concentrate on what I considered to be essential.

This work would not have been possible without the support of my advisor, professor Heikki Mannila. I am very grateful for the kind and insightful way that he has mentored me during this time, as well as for the valuable time that he has given me in the form of discussions and advice. I regard it as a great privilege to have been a part of his research group.

I am also very grateful to professor Mikael Fortelius. His enthusiasm and intelligence have been truly inspiring. Working with him has been a great pleasure.

Furthermore, I would like to thank all my other co-authors and colleagues: Antti Ukkonen, Sami Hanhijärvi, Jouni Seppänen, Kai Puolamäki, Gemma Garriga, Jilles Vreeken, Arno Siebes, Jussi Eronen, Taneli Mielikäinen, and Nikolaj Tatti, as well as the whole data mining group at TKK, including Jaakko Hollmén, Ella Bingham, Panos Papapetrou, Mikko Korpela, Janne Toivola, Jefrey Lijffijt, Markus Ojala, Niko Vuokko and Ville Pettersson. I am truly fortunate to have worked among such gifted and good-hearted people.

A special word of thanks goes to Antti Ukkonen for his support and for our discussions together on topics ranging from data mining to algorithmic trading and to movie scripts and beyond. The Finnish-Dutch collaboration with Jilles Vreeken and professor Arno Siebes is something that I want to mention separately. It has been a truly enjoyable experience.

I also thank the pre-examiners of this thesis: Professor Jian Pei of the Simon Fraser University and Professor Carson Kai-Sang Leung of the University of Manitoba.

Special thanks go to Juha Kettunen for helping me with the cover and other illustrations of the thesis.

I am grateful for the student grants that I have received from the Nokia Foundation and the Foundation for Technology Promotion (TES).

Finally, I want to express my profound gratitude to my dearest companion throughout this journey, Katri, for her support and love. I am also grateful to my mother, my father, my sister and my grandparents for all that they have given me and done for me.

Hannes Heikinheimo, Otaniemi, December 2009

Chapter 1

Introduction

1.1 Motivation

Large amounts of data are being gathered in various situations in which we interact with society. For instance, on a daily trip to the grocery store the set of products we buy is stored in a database owned by the supermarket chain. When we use a credit card, the transaction is recorded in the database of the credit card company. Furthermore, during the trip, information about our location, phone calls, messages and mobile Internet traffic is stored in the databases of a cell phone service provider. When we use public transportation, the trip is recorded in the database of the bus or train company when our (electronic) fare ticket is stamped. The list goes on and on.

This thesis is about methods and algorithms that can be used to create value from this stream of data. The idea is to seek for something called *knowledge*, which means regularities, rules and structure hidden in the data. This activity is a subfield of computer science called *knowledge discovery* or sometimes *data mining*. This knowledge will help in making decisions and conclusions that lead to value creation for both the user and the owner of the data. For instance, the purchase information collected by a supermarket chain may help the supermarket to adjust product offering and availability to better suit the needs of its customers. A credit card company familiar with the purchase history of its customers can detect when a credit card has been stolen and used to buy goods or services that the customer would be unlikely to buy himself. Using location tracking technologies, a cell phone service provider can offer map-based services such as navigation or search of nearby restaurants. Moreover, bus and train companies can use recorded passenger data to help plan bus services to run more often where needed.

This thesis focuses on a specific topic of data mining called *frequent pattern mining*, first studied in 1993 by Agrawal *et al.* [2]. The application of the paper by Agrawal *et al.* was related to finding popular product combinations from a supermarket transaction database. An example of such a product combination could be, say,

$$\{\mathsf{milk},\ \mathsf{cheese},\ \mathsf{bread}\}.$$

The idea is that a database user defines a frequency condition, such as all product combinations that occur in more than 25 percent of customer transactions, based on which an algorithm then enumerates all cases in concordance with this condition. From such product combinations it is then further possible to compute rules such as

$$\{\mathsf{milk},\ \mathsf{bread}\} \rightarrow \{\mathsf{cheese}\},$$

that have sufficient accuracy, again defined by the database user. The example rule means that if a person buys milk and cheese, he or she is most likely to buy bread as well. Such rules are called *association rules*, while combinations of items having a frequency over a certain threshold are called *frequent itemsets.* Both association rules and frequent itemsets are examples of *patterns.*

Since Agrawal *et al.*, the frequent pattern mining problem has been studied extensively with alternative problem formulations, as well as new variants of existing algorithms together with new applicational settings such as telecommunications, bioinformatics, web mining, text mining, and many more. In retrospect, however, the efficiency (run time) of enumerating the complete set of frequent patterns has attracted most research in the subject.

In this thesis we turn our attention from run time efficiency to address issues related to the interpretation and practicality of frequent pattern mining results. Although run time efficiency is also an important aspect of practicality, in real life application settings, coming up with a few high quality patterns is likely to be more valuable than simply enumerating a massive number of patterns in a very short time. Indeed, in a recent survey on frequent pattern mining by Han *et al.* [31] it was argued that while efficiency of the mining task is no longer a bottleneck, there is still an urgent need for methods that derive compact, yet high quality results with good application properties.

This work concentrates on developing expressive new frequent pattern mining methods for gaining more compact and interpretable data mining results. More precisely, the expressiveness of a data mining method is its ability to express relationships between attributes in the data, such as hierarchical relationships or relationships based on types of dependency structure other than simple co-occurrence. Compactness refers to the method's ability to convey the most relevant and important interactions of attributes in the data in a concise and non-redundant manner. Expressiveness and compactness are intertwined: for good compact data mining results we need expressive mining methods.

1.2 Outline of the thesis

The outline of the rest of the thesis is as follows:

- Chapter 2 presents some basic definitions and notation. A short introduction to frequent itemset mining is given, and the basic k-means clustering algorithm is briefly discussed. An overview of the data sets used in the experiments sections of the thesis is also given.
- In Chapter 3, a new type of tree pattern class for unstructured binary data is defined. There is a theoretical discussion of some of its properties, such as behavior in random data, and monotonicity, enabling a simple level-wise algorithm to be devised. In the experimental section examples of patterns discovered using the method are given. Experiments with random data are also provided.
- In Chapter 4, a new score, entropy, is proposed for frequent pattern mining. It is shown that entropy can be easily applied to both set and tree types of pattern, and that as a monotonic concept it allows the use of the level-wise approach. In the experimental section the behavior of the approach is tested empirically with real life data sets, as well as swap randomized data.
- Chapter 5 presents the idea of using low entropy-based patterns and minimum description length (MDL) for compact data description. It is shown that each row in the data can be encoded using the maximum likelihood principle, which is in relation to minimizing data encoding length. In the experiments section, we study the collections of low-entropy patterns resulting from the approach, and compare them with results using frequent itemsets.
- Chapter 6 introduces the idea of relating itemsets to numerical variables in a database of mixed data types. This can be considered either as a pattern selection approach or as a constraint clustering problem. Theoretical discussion includes proof of NP-completeness and a simple greedy constant-factor approximation algorithm. In the experiments section we test the algorithm's ability to convey structure, in both the numerical and the binary part of the data.

1.3 Author's contribution and relation to other publications

This thesis is based on papers [27, 39, 40, 41]. More precisely, Chapter 3 is based on the work presented in [40], while Chapters 4 and 5 are based on the work in [39] and [41]. Chapter 6 discusses ideas published in [27]. Notation and terminology have been made consistent within this thesis. Missing details, such as proofs of the most important theorems and propositions have been supplied, while a few minor topics discussed in the original papers have been omitted.

The specific contributions of the author are the following:

- The theoretical ideas of Chapter 3 were jointly developed with the co-authors of [40], however with major contributions to the design of the main concept by the author, as with the theoretical property discussed in Section 3.3.3. Implementation of the concept and the experiments, using real data, was performed by the author.
- The main ideas presented in Chapter 4 were jointly developed with the co-authors of [39]. All experiments and some of the implementions were carried out by the author.
- The most important theoretical contributions by the author in Chapter 5 include the ideas presented in Section 5.3.2, as well as Algorithm 4, Proposition 5.1 and candidate ordering in Section 5.3.3. Other main ideas were developed together with the co-authors of [41]. The author also performed a significant part of the implementation and experimentation tasks presented in the chapter.
- The results of Chapter 6 were due to a joint effort with the co-authors of [27]. The most significant contributions by the author are related to the design of the main concept and the proof of Theorem 6.1. The implementation task, as well as most experiments, were also performed by the author.

Chapter 2

Preliminaries

2.1 Basic definitions and notation

A dataset is a set $\mathcal{D}$ of observations made over a set of attributes $\mathcal{A}$. More specifically, each observation in $\mathcal{D}$ is a vector of (measured) values observed simultaneously for the set of attributes $\mathcal{A}$. As a whole, $\mathcal{D}$ can be viewed as a matrix of n rows of observation vectors and m columns with attributes as headers. An observation can also be referred to as a *data point*.

We denote a single attribute by a capital letter from the beginning of the alphabet, that is, by $A, B, C \ldots$, and so on. An observation is denoted by the row vector t. Given an attribute A and a row t, we denote the value of attribute A on row t by $t(A)$. Depending on the type of attribute A, the value of $t(A)$ may be either binary, that is $t(A) \in \{0, 1\}$, or numerical, that is, $t(A) \in \mathbb{R}$.

Binary valued attributes are often called *items*. Items are attributes that can be either present or absent at the moment of observation. The entire set of items is denoted by $\mathcal{I}$. For datasets consisting of only binary attributes we have $\mathcal{I} = \mathcal{A}$, and by convention, a row t may be expressed as a subset of the universe of all attributes, i.e., $t \subseteq \mathcal{I}$, containing those binary attributes with value 1 in row t. Observations of binary attributes are often called *transactions*, hence the symbol t. Subsets of $\mathcal{I}$ are called *itemsets*, and denoted with a capital letter from the end of the alphabet, that is, by $\ldots, X, Y, Z$. We omit the braces around singleton sets, e.g. we write A instead of $\{ A \}$. We call an itemset X of size k a *k-itemset*.

2.2 Frequent itemset mining

Section 1.1 briefly discussed frequent itemsets and association rules. A more precise introduction to these basic pattern types is given in this section.

2.2.1 Frequent itemsets

Given a dataset $\mathcal{D}$ and an itemset X, the frequency $f(\mathcal{D}, X)$ of X in $\mathcal{D}$ is defined as

$$f(\mathcal{D}, X) = |\{t | X \subseteq t \text{ and } t \in \mathcal{D}\}|,$$

that is, the number of rows t where the items of X occur together. We say that an itemset X *covers* all such rows $t \in \mathcal{D}$ where $X \subseteq t$. We denote $f(X)$ when $\mathcal{D}$ is clear from the context. We call an itemset *frequent* when its frequency exceeds a certain threshold:

Definition 2.1 *Given some predefined threshold $\sigma \in [0, 1]$, a* frequent itemset *X is an itemset that has a frequency higher than σ in $\mathcal{D}$, that is, $f(\mathcal{D}, X) \geq n\sigma$, where n is the number of rows in $\mathcal{D}$.*

Given $\mathcal{D}$ and σ, we denote the collection of all frequent itemsets by $\mathcal{P}(\mathcal{D}, \sigma)$. The frequent itemset mining problem is then the following:

Problem 2.1 *Given $\mathcal{D}$ and σ, compute $\mathcal{P}(\mathcal{D}, \sigma)$.*

2.2.2 The level-wise algorithm

The trivial way of computing $\mathcal{P}(\mathcal{D}, \sigma)$ would be to exhaustively go through all possible itemsets, and then check from the data which sets satisfy the frequency condition. However, as the number of itemsets is exponential with respect to the entire set of attributes $\mathcal{I}$, this is not feasible.

In 1994, studies by Mannila *et al.* [63] and Agrawal *et al.* [4] led independently of each other to an alternative method that is based on a simple observation: for all pairs of itemsets Y and X, such that X is a subset of Y, the frequency of superset Y can only be equal to or smaller than the frequency of X, in other words $f(X) \geq f(Y)$, when $X \subseteq Y$. This is referred to as the *monotonicity* property of frequent itemsets. Hence, in order for the itemset Y to be frequent, all its subsets have to be frequent as well. Respectively, if we know that a subset X of Y is not frequent, we know without referring to the data that Y cannot be frequent.

The observation naturally gives rise to the following strategy. First look for the single attributes (frequent 1-itemsets) that fulfill the frequency condition. We call this level 1. Once the frequent 1-itemsets are known, we can proceed to level 2, that is, to look for all frequent 2-itemsets, and so on for levels 3, 4, etc., until the complete set of frequent itemsets has been found. At each level the monotonicity property can be used to efficiently prune the search. Namely, for any frequent k-itemset we know that all its $(k-1)$ subsets must also be frequent. That is, any frequent 2-itemset can only be derived from items that were confirmed frequent at level 1. Hence, before advancing to level k we can build a collection of *candidate* itemsets from the frequent (k-1)-itemsets, and check the frequency only for these candidates at level k, while pruning all the rest.

The algorithm is called the *level-wise algorithm* based on the breadth-first manner in which the collection of frequent itemsets is generated. Algorithm 1 gives a pseudo-code presentation of the level-wise algorithm.

The level-wise algorithm is efficient in situations where the data does not fit in main memory. At each level, only one database pass is needed: for each candidate a counter can be created and then incremented at each row containing the candidate. Efficient methods for situations where the data can be stored entirely in main memory include the depth-first search-based Eclat algorithm [95], which uses set intersection operations between attribute columns. Another often cited frequent itemset mining method is the FP-growth (frequent pattern growth) [32] algorithm. It avoids the process of candidate generation and testing by using an extended prefix-tree (FP-tree) structure to store the database in a compressed form, from which the collection of frequent itemsets can then be queried very efficiently.

2.2.3 Association rules

An association rule is an implication between two itemsets, that is

$$X \rightarrow Y$$

for the itemsets X and Y, such that $X \cap Y = \emptyset$. Like a single itemset, each association rule has a frequency f, defined as $f(X \rightarrow Y) = f(X \cup Y)$. Additionally, an association rule has the quantity *accuracy*, which is the frequency with which the implication holds true, that is, the relative number of times that the itemset Y occurs on a row t, given that the itemset X occurs on t. More formally

$$\mathit{accur}(X \rightarrow Y, \mathcal{D}) = \frac{f(\mathcal{D}, X \cup Y)}{f(\mathcal{D}, Y)}.$$

We denote $\mathit{accur}(X \rightarrow Y)$ when $\mathcal{D}$ is clear from the context. We call the rule $X \rightarrow Y$ *accurate*, if $\mathit{accur}(X \rightarrow Y)$ exceeds a certain threshold γ.

Definition 2.2 *Given two predefined thresholds σ and γ, such that $\sigma, \gamma \in [0, 1]$, the association rule $X \rightarrow Y$ is frequent and accurate if $f(X \rightarrow Y) \geq n\sigma$ and $\mathit{accur}(X \rightarrow Y) \geq \gamma$, where n is the number of rows in $\mathcal{D}$.*

Given $\mathcal{D}$, σ and γ we denote the collection of all frequent and accurate association rules by $\mathcal{R}(\mathcal{D}, \sigma, \gamma)$. The association rule mining problem is then the following:

Problem 2.2 *Given $\mathcal{D}$, σ and γ, compute $\mathcal{R}(\mathcal{D}, \sigma, \gamma)$.*

Problem 2.2 can be considered as a general case of the frequent itemset mining Problem 2.1. For a single itemset X, the trivial rule $X \rightarrow \{\}$ has the same frequency $f(X)$ as that of itemset X and $\mathit{accur}(X) = 1$. Respectively, the frequency of the rule $\{\} \rightarrow X$ is equal to 1, while the accuracy is equal to the frequency of X. It also turns out that there is a monotonicity property that holds for the accuracy of an association rule with respect to extending the right-hand side of the rule.

Algorithm 1 The level-wise algorithm

Input: Dataset $\mathcal{D}$ and frequency threshold σ.
Output: Collection of frequent itemsets $\mathcal{P}(\mathcal{D}, \sigma)$.

```
1: $i = 1$
2: candidate set $C_i = \{\{A\} | A$ is an attribute$\}$
3: while $C_i$ is not empty do
4:    set of size $i$ patterns $\mathcal{P}_i = \{\}$
5:    // Frequency checking
6:    for each $X$ in $C_i$ do
7:       if $f(\mathcal{D}, X) \geq \sigma$ then
8:          add $X$ to $\mathcal{P}_i$
9:       end if
10:   end for
11:   // Candidate generation
12:   for each $X, Y$ in $\mathcal{P}_i$ do
13:      candidate $Z = X \cup Y$
14:      if $|Z| = i + 1$ and $(Z \setminus A) \in \mathcal{P}_i$ for all $A \in Z$ then
15:         add $Z$ to $C_{i+1}$
16:      end if
17:   end for
18:   add $\mathcal{P}_i$ to $\mathcal{P}(\mathcal{D}, \sigma)$
19:   $i = i + 1$
20: end while
21: return $\mathcal{P}(\mathcal{D}, \sigma)$
```

Proposition 2.1 *Let X, Y, Z be three itemsets, such that $X \cap Y = \{\}$. Then*

$$\mathit{accur}(X \setminus Z \to Y \cup Z) \leq \mathit{accur}(X \to Y)$$

Proof Since $X \cup Y \subseteq X \cup Y \cup Z$, and $X \setminus Z \subseteq X$,

$$\frac{f(X \cup Y \cup Z)}{f(X \setminus Z)} \leq \frac{f(X \cup Y)}{f(X)}.$$

□

Proposition 2.1 leads naturally to a level-wise type of approach for finding the set $\mathcal{R}(\mathcal{D}, \sigma, \gamma)$. First compute $\mathcal{P}(\mathcal{D}, \sigma)$. Note that for each itemset $X \in \mathcal{P}(\mathcal{D}, \sigma)$, we have the rule $X \to \{\}$ in $\mathcal{R}(\mathcal{D}, \sigma, \gamma)$. Now, for each $X \to \{\} \in \mathcal{R}(\mathcal{D}, \sigma, \gamma)$, start extending the rule right-hand side with subsets Z of X in a level-wise manner, while removing Z from X on the left-hand side. If some itemset Y, such that $X \cap Y = \emptyset$, on the right-hand side of the rule, causes the accuracy to drop below γ, we know, according to Proposition 2.1, that no rule with a superset of Y on the right-hand side will be accurate. Hence, further extension of this rule can be stopped.

From the above it is evident, given $\mathcal{P}(\mathcal{D}, \sigma)$, that the set of association rules $\mathcal{R}(\mathcal{D}, \sigma, \gamma)$ can be computed without referring to the data at all. Hence, from the point of view of knowledge discovery the set of frequent itemsets is more interesting. Consequently, many frequent pattern mining related studies have mostly been concerned with issues related to frequent itemset mining.

2.2.4 Closed and maximal frequent itemsets

An often mentioned problem in frequent itemset mining is that in practice the number of returned patterns in $\mathcal{P}(\mathcal{D}, \sigma)$ may become very large even with relatively small values of σ. This is referred to as the *pattern explosion problem. Closed frequent itemsets* [74, 77] and *maximal frequent itemsets* [7, 10] are among the earliest and best known methods to tackle this problem.

Closed frequent itemsets

The idea behind closed frequent itemsets can be motivated by the following example borrowed from [77]. Consider a dataset with only two data rows $t_1 = \{A_1, ..., A_{100}\}$ and $t_2 = \{A_1, ..., A_{50}\}$. Setting the frequency threshold to $\sigma = 0.5$ results in the set of all possible $2^{100} - 1 \approx 10^{30}$ frequent itemsets. Clearly, $\{A_1, ..., A_{100}\}$ and $\{A_1, ..., A_{50}\}$ are the only effective patterns in the data. Any other subpattern of the two itemsets has the same frequency as one or the other, and hence conveys only redundant information to the user. The set of closed frequent itemsets is the set of frequent itemsets from which such subpatterns have been removed.

Definition 2.3 *A frequent itemset X is a* closed frequent itemset *if there exists no frequent itemset Y, such that $X \subseteq Y$ and $f(X) = f(Y)$. Given data $\mathcal{D}$ and a frequency threshold σ, we denote the set of all closed frequent itemsets by $\mathcal{P}_{closed}(\mathcal{D}, \sigma)$.*

Several efficient algorithms applying various search strategies have been proposed for mining closed frequent itemsets [74, 77, 97]. For a performance study on closed frequent itemset mining, see [91].

Maximal frequent itemsets

The reduction from all frequent itemsets to closed frequent itemsets is *lossless* in the sense that all sets and their frequencies in $\mathcal{P}(\mathcal{D}, \sigma)$ can be easily derived from $\mathcal{P}_{closed}(\mathcal{D}, \sigma)$. Maximal frequent itemsets can be considered a *lossy* and less conservative variant of the closed frequent itemsets.

Definition 2.4 *A frequent itemset X is a* maximal frequent itemset *if there exists no frequent itemset Y, such that $X \subseteq Y$. Given data $\mathcal{D}$ and a frequency threshold σ, we denote the set of all maximal frequent itemsets by $\mathcal{P}_{max}(\mathcal{D}, \sigma)$.*

Note that the second condition $f(X) = f(Y)$ imposed for closed frequent itemsets does not hold for maximal frequent itemsets. Otherwise the definitions are the same. Consequently $\mathcal{P}(\mathcal{D}, \sigma)$ can still be derived from $\mathcal{P}_{max}(\mathcal{D}, \sigma)$, however, the frequencies of the sets in $\mathcal{P}(\mathcal{D}, \sigma) \backslash \mathcal{P}_{max}(\mathcal{D}, \sigma)$[1] are lost. Algorithms for computing maximal frequent itemsets are typically based on effective depth-first look-up strategies. One of the most cited maximal frequent itemset mining approaches is the MAFIA-algorithm by Burdick *et al.* [10].

2.3 Clustering

Out of all data mining and machine learning techniques, clustering is probably one of the most studied topics (see for instance [60, 46, 33, 88, 92]). The idea of clustering is to provide a grouping on a set of rows such that the similarity between the rows within each group is as large as possible while the dissimilarity in different groups is maximized. Clustering can be used in applications such as customer profiling and segmentation in market analysis, in bioinformatics, in ecology, in text document analysis and many more.

In the following we briefly discuss a basic partition-based algorithm, *k-means*, which will be referred to later in the thesis.

2.3.1 The k-means algorithm

The k-means algorithm [60] is a basic iterative clustering procedure the can be used to partition the data in k number of clusters. Each k cluster is represented by a mean vector computed over the rows in the cluster. Hence the name k-means. More precisely, if we denote by C_i the set of rows in the i:th cluster, the corresponding center r_i is defined as

$$r_i = \frac{1}{|C_i|} \sum_{t \in C_i} t. \tag{2.1}$$

Formally the problem of k-means clustering is the following:

Problem 2.3 *Given a dataset $\mathcal{D} = \{t_1, ..., t_n\}$ find a partition of $\mathcal{D}$ into k disjoint sets (clusters) $C_1 \ldots C_k$ such that*

$$\sum_{i=1}^{k} \sum_{t \in C_i} (r_i - t)^2 \tag{2.2}$$

is minimized.

[1]It is easy to see that $\mathcal{P}_{max}(\mathcal{D}, \sigma) \subseteq \mathcal{P}_{closed}(\mathcal{D}, \sigma) \subseteq \mathcal{P}(\mathcal{D}, \sigma)$. In practice, the set $\mathcal{P}_{max}(\mathcal{D}, \sigma)$ may be up to orders of magnitude smaller than the set $\mathcal{P}_{closed}(\mathcal{D}, \sigma)$, which itself is orders of magnitude smaller than the set $\mathcal{P}(\mathcal{D}, \sigma)$ [10].

Algorithm 2 The k-means algorithm

```
Input: Dataset $\mathcal{D} = \{t_1, ..., t_n\}$ and the number of clusters $k$.
Output: Clusters $C_1, ..., C_k$ and the corresponding centers $r_1, ..., r_k$.
 1: for $i = 1, .., k$ do
 2:     initialize $r_i$ with a randomly selected data point in $\mathcal{D}$.
 3: end for
 4: while changes in $C_1, ..., C_k$ happen do
 5:     // associate rows to clusters
 6:     for $i = 1, .., k$ do
 7:         $C_i = \{t \in \mathcal{D} | r_i = \text{argmin}_{r_j} (r_j - t)^2 \text{ where } j = 1, ..., k\}$
 8:     end for
 9:     // recalculate cluster means
10:     for $i = 1, .., k$ do
11:         $r_i = \frac{1}{|C_i|} \sum_{t \in C_i} t$.
12:     end for
13: end while
14: return $C_1, ..., C_k$ and $r_1, ..., r_k$
```

Equation (2.2) is sometimes referred to as the *sum-of-squares error* of the k-means model. Depending on the application, the term $(r_i - t)^2$ can also be replaced with other error term variants.

Algorithm 2 gives a pseudo-code presentation of the procedure for minimizing Equation (2.2). The algorithm is initialized using a random assignment of cluster centers. The first step of the procedure is to take each row t in the dataset and associate it with the nearest cluster center r_i in terms of the squared error $(r_i - t)^2$. The second step is to recalculate each cluster center by assigning to it the mean of the data rows associated with it, as in Equation (2.1). By repeating these two steps sufficiently many times, the algorithm starts to converge, such that the cluster centers find a locally optimal position in the data space. The final clustering is obtained by associating each data point with the nearest converged cluster center.

2.4 Pattern validation via swap randomization

A common property of frequent pattern mining, clustering and other data mining methods is that they will always produce an outcome of some sort. The k-means algorithm is guaranteed to produce a partition of the data into k clusters, whether or not the data truly has a genuine cluster structure. Moreover, the level-wise algorithm will always produce a set of frequent patterns, even for simple random data, provided that the frequency threshold σ is assigned sufficiently low. Hence, it is important to evaluate the statistical significance of the mining results to make sure that it is due to other than just random noise in the data.

In significance testing an observed statistic is compared against a *null distribution*. The null distribution can be considered as the distribution of the statistic in the case where no relevant structure is present in the data. If the statistic tested exhibits a very unlikely (an extreme) value in the null distribution, it can be concluded that the observed value is due to something mere randomness in the result. The confidence for this is expressed using the *p-value*, which is the probability of obtaining a test statistic at least as extreme as the one that was actually observed.

The null distribution of a statistic can be derived either analytically or empirically. Generally, for more complex models and their statistics, such as clustering and frequent pattern sets, analytical derivation of the null distribution becomes very difficult. Hence, for most common data mining methods, the null distribution may only be derived empirically.

In [29] Gionis *et al.* proposed the use of *swap randomization* [16] for empirically assessing data mining results in binary data. The randomization procedure is based on repeating a basic swap operation: given data $\mathcal{D}$, take two random rows t and u and two random attributes A and B of the data, with $t(A) = u(B) = 1$ and $u(A) = t(B) = 0$, and change the values of these entries such that $t(A) = u(B) = 0$ and $u(A) = t(B) = 1$ (see illustration in Figure 2.1).

A special property of the swap operation is that it maintains the row and column sums of the data, and all datasets with the same row and column sums can be reached through a series of swaps starting from $\mathcal{D}$ [16]. Hence, given any statistic for a binary dataset $\mathcal{D}$, the swap precedure can be used to sample random data sets with the same row and column sum as $\mathcal{D}$ in order to obtain an empirical null distribution of the statistic of interest.

More precisely, let $\mathcal{S}(\mathcal{D})$ be the statistic of interest, say the number of frequent patterns in $\mathcal{D}$ or the sum-of-squares error for the k-means clustering. Furthermore, let $\hat{D} = \{\hat{\mathcal{D}}_1, \ldots, \hat{\mathcal{D}}_l\}$ be a collection of independent randomized versions of the original dataset $\mathcal{D}$. The one-tailed *empirical p-value* of $\mathcal{S}(\mathcal{D})$ is

$$\frac{|\{\hat{\mathcal{D}} \in \hat{D} \,|\, \mathcal{S}(\hat{\mathcal{D}}) \geq \mathcal{S}(\mathcal{D})\}| + 1}{l+1},$$

which is the fraction of randomized datasets whose statistic is larger than the original statistic $\mathcal{S}(\mathcal{D})$. The one-tailed empirical p-value when small values of $\mathcal{S}(\mathcal{D})$ are interesting, and the two-tailed empirical p-value are defined similarly. If the p-value obtained is less than a given threshold α, say $\alpha = 0.05$, we can regard the result $\mathcal{S}(\mathcal{D})$ as significant.

The randomized data set in the collection $\hat{D}$ can be produced by using consecutive swaps in the style of the Markov chain Monte Carlo. Starting from the original dataset $\mathcal{D}$, some number of swaps are made, thus producing a new dataset $\hat{\mathcal{D}}$. It is important, however, that the sampling is done uniformly from the set of all possible datasets in order to avoid introducing any sampling bias into the significance test. A straightforward application of swapping does not guarantee uniform sampling, as some datasets have more swappable matrix en-

$$
\begin{array}{c|c|}
 & \begin{array}{ccccc} & A & & B & \end{array} \\
\hline
 & \begin{array}{ccccc} & \vdots & & \vdots & \end{array} \\
t & \begin{array}{ccccc} \cdots & 1 & \cdots & 0 & \cdots \end{array} \\
 & \begin{array}{ccccc} & \vdots & & \vdots & \end{array} \\
u & \begin{array}{ccccc} \cdots & 0 & \cdots & 1 & \cdots \end{array} \\
 & \begin{array}{ccccc} & \vdots & & \vdots & \end{array} \\
\hline
\end{array}
\quad \Longleftrightarrow \quad
\begin{array}{c|c|}
 & \begin{array}{ccccc} & A & & B & \end{array} \\
\hline
 & \begin{array}{ccccc} & \vdots & & \vdots & \end{array} \\
t & \begin{array}{ccccc} \cdots & 0 & \cdots & 1 & \cdots \end{array} \\
 & \begin{array}{ccccc} & \vdots & & \vdots & \end{array} \\
u & \begin{array}{ccccc} \cdots & 1 & \cdots & 0 & \cdots \end{array} \\
 & \begin{array}{ccccc} & \vdots & & \vdots & \end{array} \\
\hline
\end{array}
$$

Figure 2.1: A swap in a binary dataset.

tries than others. However, this can be corrected by introducing a very simple modification into the procedure: when sampling the data matrix for swappable entries, instead of just counting an occurred swap as a valid randomization step, count also the nonswappable entries encountered while sampling. The counted nonswappable entries are sometimes called *self-loops*. Introducing self-loops leads to uniform sampling as the difference in swappable entries between datasets will be evened out by the self-loops. It is estimated that for the process to converge sufficiently, the number of steps in the randomization procedure should be of the order of 1s in the data matrix. For a more detailed discussion see [29].

2.5 Datasets

Throughout this thesis the main experimental setting will be based on three real life datasets: Mammals, MovieLens and Course. In addition to this, we supplement our experiments in Chapter 5 with a few datasets from the widely used UCI repository [17].

Mammals data

The Mammals dataset consists of presence/absence records of European mammals [66] in a geographical area defined by the latitudinal and longitudinal boundaries of 32°W, 35°E, 81°N, 30°N. The original data defines the occurrence of 194 species (attributes) within 2670 grid cells (rows) with a resolution of 50×50 km. For each grid cell and each species the dataset features an entry of 0 or 1 depending on whether the corresponding species is absent or present in the respective grid cell. The dataset used in this thesis is the preprocessed (124 species × 2183 grid cells) version used in [38]. The data were collected by Societas Europaea Mammalogica. The full version of the dataset is available for research purposes upon request from `www.european-mammals.org`. For the convenience of the reader we use here the English species names translated from the scientific names using the Wikipedia encyclopedia (`www.wikipedia.org`).

In addition to the binary observations we combine a set of numerical attributes with the Mammals data. For each grid cell we associate the geographical longitude and latitude (spatial coordinates), as well as observations of mean annual temperature, mean annual precipitation, mean annual temperature range[2] and average elevation. The climate and elevation observations were obtained from the dataset [42] available online at `www.worldclim.org`.

MovieLens data

The MovieLens data consists of 100,000 movie ratings, valued from 1 to 5, from 943 users (rows) on 1682 movies (attributes). Each user has rated at least 20 movies. Furthermore, for each user the data includes age, gender, occupation and zip code. The data were collected through the MovieLens website (`movielens.umn.edu`) during a seven-month period between September 19th, 1997 and April 22nd, 1998. The data can be download from `www.grouplens.org`. The data available for downloading have been preprocessed, that is, users who had less than 20 ratings or did not provide complete demographic information were removed from the dataset.

For the experiments in this thesis, we converted the movie ratings into binary items: for any user, a movie rated with 4 or 5 was mapped to a 1, while ratings from 1 to 3 and non-rated movies were mapped to 0. In other words, one row of the data was made to correspond to those movie items that a user most prefers. Numerical demographic attributes were also mapped to each of the users (rows). Gender was encoded such that males were assigned the value 1 and females the value 0. Age was used as recorded in the data. Occupation information was omitted.

Course data

We use two kinds of data gathered at the Department of Computer Science at the University of Helsinki. The first, Course Enrollment, includes enrollment records for courses held at the Department of Computer Science at the University of Helsinki. The dataset has 3506 rows corresponding to students and 98 attributes corresponding to courses. The second, Course Completion, consists of course completion data for a different set of students. This data has 2405 observations corresponding to students and 5021 attributes corresponding to courses.

UCI repository

In addition to the three datasets described above we take from the widely used UCI repository [17] the datasets Heart, Pen digits, Mushroom and Letter Recognition to be used in the experiments of Chapter 5. For more information about these data sets, see `archive.ics.uci.edu/ml/datasets.html`.

[2]Difference between the maximum temperature of the warmest month and the minimum temperature of the coldest month.

Chapter 3

Mining trees from unstructured binary data

3.1 Introduction

An itemset does not impose a structure of any kind on its attributes. Also, while association rules $X \rightarrow Y$ offer a slightly more structured view, the structure is presented only with respect to itemsets, not with respect to single attributes. Yet, even for unstructured data it is likely that the domain from which the observations are sampled contains a structure, such as a hierarchy, that should be expressed at the attribute level. Because of the unstructured nature of itemsets, such relationships will not be revealed.

As an example, consider movie ratings data, such as in the MovieLens dataset. It is not uncommon for successful Hollywood movies to be followed by a sequel movie. Popular themes that are box office successes, will be reused again and again, creating whole genres of niche movies. Take, for instance, the Indiana Jones hit movie Raiders of the Lost Ark (1981). Figure 3.1 shows how a genre of treasure hunt movies has followed since its première in 1981.

Looking at this from the point of view of the movie rater, the original movie(s) will usually be appreciated by a wider audience. However, due to the lack of novelty, the continuing sequel movies will lose their appeal to the general audience at some point, while a more specific audience of enthusiasts will still appreciate the niche films of the genre. Hence, if a user gives a high rating to some specific niche movie, probably he or she will give high ratings to all of the previous (including the more mainstream) films of the genre as well. Hence, a hierarchy of general versus more specific will arise from the dependencies of the movie attributes.

Consider the hierarchy in Figure 3.1. It is not hard to imagine that a user who gave a high rating to the movie Indiana Jones and the Kingdom of the Crystal Skull (2008), also gave a high rating to all the other Indian Jones movies. However, more users probably preferred the first Indiana Jones movie, Raiders of the Lost

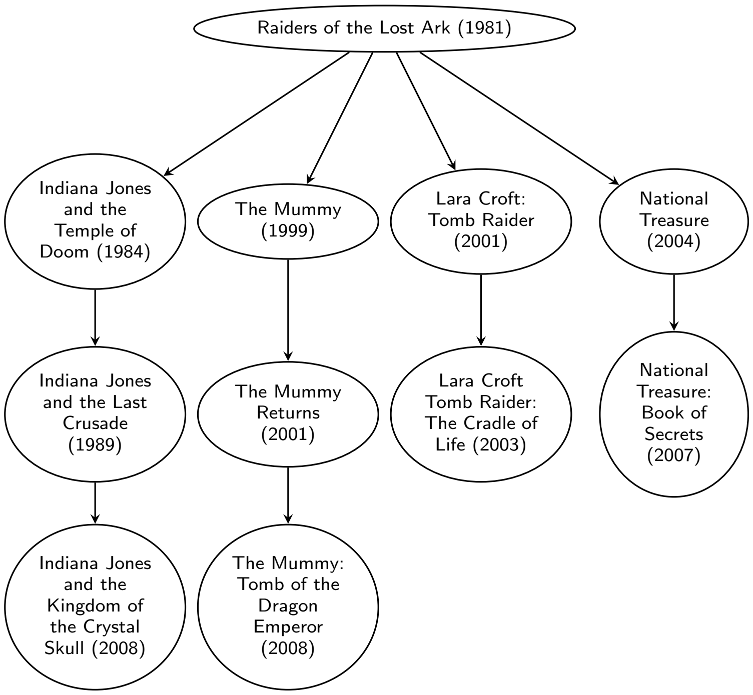

Raiders of the Lost Ark (1981)
Indiana Jones and the Temple of Doom (1984)
The Mummy (1999)
Lara Croft: Tomb Raider (2001)
National Treasure (2004)
Indiana Jones and the Last Crusade (1989)
The Mummy Returns (2001)
Lara Croft Tomb Raider: The Cradle of Life (2003)
National Treasure: Book of Secrets (2007)
Indiana Jones and the Kingdom of the Crystal Skull (2008)
The Mummy: Tomb of the Dragon Emperor (2008)

Figure 3.1: The Indiana Jones hit movie *Raiders of the Lost Ark (1981)* has been followed by several Indiana Jones sequels as well as a whole genre of treasure hunt movies.

Ark (1981), but not the sequel Indiana Jones and the Kingdom of the Crystal Skull (2008).

A similar kind of structural hierarchy of general versus more specific may be present in other types of data as well. In the course enrollment data, some courses require no previous knowledge of their subject, whereas some, more specific courses, have some introductory courses as prerequisites. In other words, if a student has enrolled in a course with some specific prerequisites, probably he or she is likely to have an earlier record of enrollment in the corresponding introductory course. However, while general courses usually are passed by many, more specific topics interest only a handful of students.

Similar phenomena can be found from other domains as well. In ecology, rare niche species may exhibit dependencies with more widespread and abundant species [34, 35]. In the domain of text mining, dependencies usually exist between semantically related terms (see [28] for a good example).

Based on these examples, a new class of co-occurrence patterns is proposed: trees. The idea is to search for frequent hierarchies of general and more specific attributes. As an example, consider a tree with item A as the root and B and C

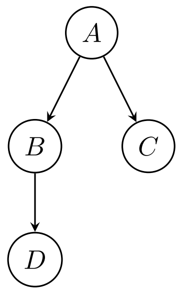


Figure 3.2: Example tree.

as the children of A, and D as the child of B (see Figure 3.2). This means that A is the general item, B and C are more specific items related to A, and D is a further specialization of B. We say that a row t follows the structure described by the tree if the items in t form a subtree of T containing the root. Figure 3.3 gives an example of this: a row with $t(A) = t(B) = t(D) = 1$ and $t(C) = 0$ satisfies this condition, but a row with $t(A) = t(D) = t(C) = 1$ and $t(B) = 0$ does not.

In this chapter we consider the problem of finding all trees T that have a sufficiently low number of violations with respect to the subtree condition. We also show that the class of patterns has advantageous properties: high quality trees are unlikely to occur in random data. The definition also allows a simple level-wise algorithm. We demonstrate with empirical results that the pattern class is feasible and that it can be used to discover interesting hierarchical relationships in real data not expressible with traditional set-type patterns such as frequent itemsets.

3.2 Problem definition

3.2.1 Graphs and trees

A graph G over a set of items X is a pair $G = (X, E)$, where X is an itemset and E is a relation defined between pairs of items in X. When referring to an item in a graph, we use the expression *node*. Similarly, the elements in E are often called *edges*.

The edges E of G can be either *directed* or *undirected*. In this chapter we are concerned with the case where edges in E are directed. Such graphs are called *directed graphs*. We denote a directed edge in E with a 2-item sequence (A, B), for two nodes $A, B \in X$. Respectively, a directed *path* in G is a sequence of nodes $S = (A_1, ..., A_k)$, such that $A_i \in X$, for $i = 1, ..., k$ and for each consecutive node pair A_i, A_{i+1} in S there exists an edge (A_i, A_{i+1}) in E.

Definition 3.1 *A* rooted directed tree *is a graph $T = (X, E)$, such that for one node $A \in X$, called the* root, *and any node $B \in X \setminus A$ there exists exactly one directed path from A to B. All edges in T are directed away from the root A.*

Example Figure 3.2 depicts a tree (X, E), for which

$$X = \{A, B, C, D\} \text{ and } E = \{(A, B), (B, D), (A, C)\},$$

and A is the root of the tree.

Trees are commonly used to represent hierarchical structure. When an edge $(A, B) \in E$ appears in a tree, we say that A is the *parent* of B and that B is a *child* of A. Furthermore, for two nodes A and C, we say that C is a *descendant* of A (or, equivalently, A is an *ancestor* of C), either if C is a child of A, or if C is a descendant of a child of A. A node with no children is called a *leaf.* A path from the root of the tree to any leaf node is called a *branch.*

Notice that removing node A from the tree in Figure 3.2 will result in two valid trees with the children B and C now as the root nodes of the two new trees. Indeed, in some cases a directed tree can be very conveniently expressed recursively using the root node and the subtrees in which its children act as a root. We have an alternative definition.

Definition 3.2 *A* rooted directed tree *is a pair $T = (A, \mathcal{C})$, where A is the root node of the tree T and $\mathcal{C}$ is a collection of rooted directed trees $\mathcal{C} = \{ T_1, T_2, \ldots, T_k \}$. Each $T_i \in \mathcal{C}$ has the form $T_i = (A_i, \mathcal{C}_i)$, such that A_i is a child of A. In the case where A has no children, the set $\mathcal{C}$ is empty.*

Example The tree in Figure 3.2 is recursively defined as

$$(A, \{T_1, T_2\}),\ T_1 = (B, \{(D, \emptyset)\}),\ T_2 = (C, \emptyset).$$

Throughout this thesis, unless stated otherwise Definition 3.1 is used as default tree definition.

3.2.2 Frequent trees

Given a tree T and a row $t \in \mathcal{D}$, we say that the row t *conflicts* with tree T if there is a node A and its descendant B in T such that $t(A) = 0$ and $t(B) = 1$. Figure 3.3 provides a example. Respectively, the *conflict count*, denoted $c(T, \mathcal{D})$, is the number of rows $t \in \mathcal{D}$ that conflict with T. We write $c(T)$ when $\mathcal{D}$ is clear from the context.

Our aim here is to consider trees with frequently occurring attributes but with the property of having a sufficiently low number of conflicts in the data. We have the following definition.

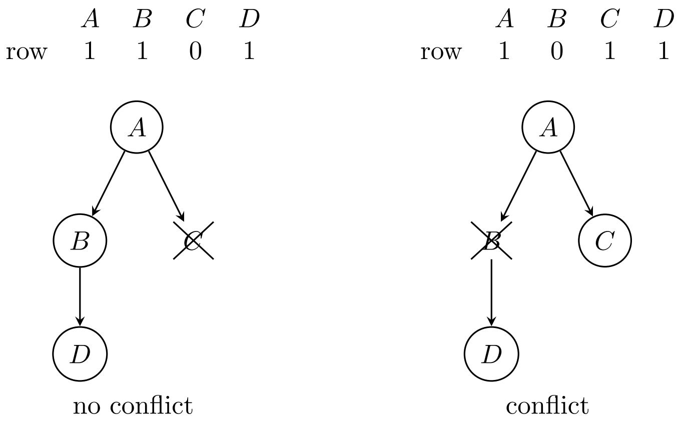


Figure 3.3: A row t follows the hierarchy described by a tree if the attributes in the tree that are 1 in t form a subtree of T containing the root. A row t *conflicts* with tree T if there is a node A and its descendant B in T such that $t(A) = 0$ and $t(B) = 1$.

Definition 3.3 *Consider a dataset $\mathcal{D}$ with n rows, and three user-defined thresholds τ, σ and β, where $\tau, \sigma \in [0, 1]$, and $\beta \in \mathbb{Z}^+$. For an itemset X, a tree $T = (X, E)$ is a* frequent tree*, if $c(T, \mathcal{D}) \leq \tau n$, $f(A) \geq n\sigma$, for all $A \in X$, and all $A \in X$ have at most β children. We denote the collection of all frequent trees in $\mathcal{D}$ with $\mathcal{TP}_F(\mathcal{D}, \tau, \sigma, \beta)$.*

We use an *upper* bound τ on the number of conflicts. Note that frequent itemsets are defined by a *lower* bound on their number of occurrences. The parameter σ has this role, preventing attributes with low frequency from being considered.

The maximum number of children a tree can have (parameter β) is related to the amount of structure a tree conveys for its nodes. If one single node has many children, the tree has short branches. This implies a low number of ancestor-descendant relations in the tree. Respectively, the number of ancestor-descendant pairs is maximal when each node has at most one child, that is the tree comprises only one single branch. In the single branch case the hierarchy is defined completely for each item pair.

The definition gives rise to the following computational problem.

Problem 3.1 *Given $\mathcal{D}$, σ, τ and β compute $\mathcal{TP}_F(\mathcal{D}, \tau, \sigma, \beta)$.*

3.3 Problem properties

Next we consider some basic properties of frequent trees and the pattern collection $\mathcal{TP}_F(\mathcal{D}, \tau, \sigma, \beta)$.

3.3.1 Monotonicity

The first observation is the simple monotonicity property typical of frequent patterns. A tree S is a rooted subtree of tree T if S can be obtained from T by a series of removals of leaves. It is quite clear that removing a leaf A from T cannot increase the number of conflicts in the tree. Hence, the tree S can have at most the same number of conflicts as T. Furthermore, the maximal number of children for a node in the tree can not grow, and the minimum node frequency cannot decrease.

Proposition 3.1 *The pattern class $\mathcal{TP}_F(\mathcal{D}, \tau, \sigma, \beta)$ is monotonic with respect to rooted subtrees, i.e., if $T \in \mathcal{TP}_F(\mathcal{D}, \tau, \sigma, \beta)$ and S is a subtree of T, then $S \in \mathcal{TP}_F(\mathcal{D}, \tau, \sigma, \beta)$.*

3.3.2 Number of tree patterns

The number of rooted labeled trees on k nodes, with an unrestricted branching factor, is k^{k-1}. This follows from the theorem of Cayley, which states that the number of labeled trees is k^{k-2}; see e.g. [57, Section 3.3] or [85, sequence A000169]. The number of possible roots is of course k, from which the result follows.

3.3.3 Expected number of conflicts

The expected number of conflicts $E[c(T)]$ can be computed recursively for a tree T, under the assumption that the attributes are independent and have the marginal frequencies observed in the data. Consider a tree $T = (A, \mathcal{C})$, defined now according to the recursive Definition 3.2. The probability that a row t does not conflict with tree T, is

$$\begin{aligned} 1 - \Pr(t \text{ conflicts with } T) &= \Pr(t(A) = 0) \prod_B \Pr(t(B) = 0) \\ &\quad + \Pr(t(A) = 1) \prod_{S \in \mathcal{C}} (1 - \Pr(t \text{ conflicts with } S)), \end{aligned}$$

where the first product is taken over all attributes B represented in T, except for the root A, and the second over all subtrees S of T, rooted by the children of A.

We call $E[c(T)]/c(T)$ the *conflict ratio* of a tree T. It is obtained by comparing the number $c(T)$ of conflicting rows in the data with its expectation $E[c(T)]$. The conflict ratio will be high for trees that have much fewer conflicts than would be expected under the independence assumption. Such trees can be argued to capture interesting co-occurrence patterns in the data that cannot be explained by mere random occurrence due to frequencies of the attributes. Conflict ratio can be used as an interestingness criterion for selecting tree patterns.

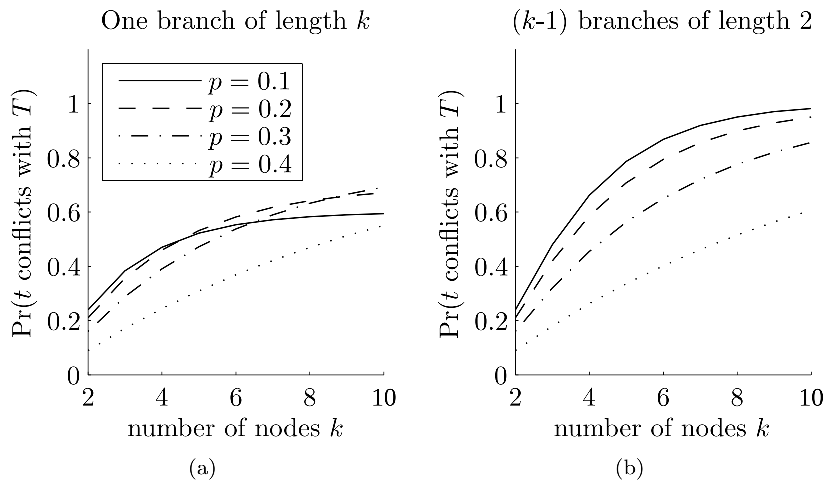


Figure 3.4: The growth of conflict probability for two different tree topologies with respect to tree size k and item occurrence probability p.

3.3.4 Number of tree patterns in random data

Although the number of all possible trees is very large, for random data (with no genuine structure) the number of interesting tree patterns should be small. To further investigate the case of random data, suppose that data $\mathcal{D}$ contains independent and identically distributed items, such that we have $t(A) = 1$ for all items $A \in \mathcal{I}$ with probability p.

Trees of size k with $k-1$ children under the root have a low expected number of conflicts. This is due to the fact that a conflict may occur only when we have $t(A) = 0$ for the root A. In fact, the conflict probability of such a tree approaches $(1-p)$ as k becomes large: from the previous subsection we have

$$\Pr(t \text{ conflicts with } T) = 1 - (1-p)^k - p,$$

where $\lim_{k\to\infty}(1-p)^k = 0$, as k approaches infinity. For trees with longer branches the expected number of conflicts is higher. Hence, it makes sense to use relatively small values of β. Figure 3.4 shows how the conflict probability grows in two different cases, with respect to tree size k and the item occurrence probability p.

With respect to the conflict ratio, it can be shown that the probability of coming across a tree with a large conflict ratio is very small in random data. Using Chernoff bounds [68, page 70], we can compute an upper bound on the probability that a tree T having a conflict ratio $E[c(T)]/c(T)$ larger than some $\delta \in \mathbb{R}_{\leq 0}$ will be encountered. From [68, page 70], we have

$$\Pr(\, c(T) < \frac{1}{\delta} E[c(T)]\,) < \frac{1}{e^{(1-1/\delta)^2 E[c(T)]/2}}. \tag{3.1}$$

Consider relating this to a dataset of m items. For such data, there are $\Gamma \leq \binom{m}{k} k^{k-1}$ trees with k nodes. Consider the case that for all of these trees T the expected conflict count is at least $E[c(T)] \geq nq$, where n is the number of rows in the data and q the conflict frequency. Thus, according to Inequality (3.1), the expected number of trees with a conflict ratio larger than, say $\delta = 2$, is bounded by

$$\frac{\Gamma}{e^{nq/8}}. \tag{3.2}$$

On the other hand, we can approximate Γ from above, as

$$\Gamma \leq \binom{m}{k} k^{k-1} \leq m^k k^k \leq m^{2k}. \tag{3.3}$$

Inserting this into (3.2), we can write the condition that the expected number of trees T with $E[c(t)]/c(T) \geq 2$ is at most 1 as

$$\frac{m^{2k}}{e^{nq/8}} \leq 1. \tag{3.4}$$

This inequality is satisfied when

$$2k \log m \leq \frac{nq}{8}. \tag{3.5}$$

As the number of tree nodes k is typically small, the inequality holds as long as we have enough data, that is, n is large enough. Hence, for a reasonable size random data set, the number of trees with large conflict ratios (interesting trees) is very small. This is a desirable property for a pattern.

Example Consider a random data set with the same dimensions as the Mammals data set, that is, with $m = 124$ attributes and $n = 2183$ rows. Say, we are interested in trees T of size $k = 4$ with conflict ratio $E[c(T)]/c(T) \geq 2$. We have

$$2 \times 4 \times \log 124 \leq \frac{2183 \times q}{8},$$

which holds when $q \geq 0.14$. Hence, assuming that the expected conflict frequency of the trees in the data is larger than 0.14 (compare Figure 3.4), the expected number of 4 size trees with conflict ratio $E[c(T)]/c(T) \geq 2$ is less then one.

Similar reasoning has been applied to frequent itemsets in [3].

3.3.5 Trees, itemsets and association rules

Next we discuss the relationship between trees, itemsets and association rules. Consider a simple 2-size tree $T = (\{A, B\}, (A, B))$ containing the root A and the child B of the root. It is easy to see that the conflict count $c(T)$ is equal to $f(B) - f(\{A, B\})$, where $f(B)$ and $f(\{A, B\})$ are the frequencies of B and

$\{A, B\}$, respectively. Also, for the association rule $B \rightarrow A$, denoting the accuracy of the rule by $\gamma = accur(B \rightarrow A)$, we have that $c(T) = f(B)(1-\gamma)$. One can ask whether other, more complex trees could be reduced to frequent itemset mining and association rules. Could we perhaps find all the trees in $\mathcal{TP}_F(\mathcal{D}, \tau, \sigma, \beta)$ just by post-processing[1] a set of frequent itemsets or association rules for the set of items in the data? This turns out not to be the case, however.

For any tree T, a row t will not conflict with T if the rule $B \rightarrow A$ is true on t for all pairs (A, B) such that B is a descendant of A in T. However, there is no simple formula for the conflict count of tree T given the accuracies of the rules $B \rightarrow A$ for item pairs of T. The reason is that a tree conflicts with a row t if at least one rule is violated: the frequency with which this happens depends on the interaction of the different rules.

As an example, consider a three-item tree with root A, and B and C as the children of A. If $f(A) = f(B) = f(C) = 0.2$ and $f(\{A, B\}) = f(\{A, C\}) = 0.1$, the accuracies of the rules $B \rightarrow A$ and $C \rightarrow A$ are both 0.5. Denoting by n the number of rows in the data, the conflict count of tree T can, however, vary between $0.2n0.5$ and $2(0.2n0.5)$. If the set of rows which do not satisfy $B \rightarrow A$ is the same as the set of rows that do not satisfy $C \rightarrow A$, the conflict count will be $(0.2n0.5)$. On the other hand, if the rows are disjoint, the conflict count will be $2 \times (0.2n0.5)$. Hence there is no algorithm for computing $\mathcal{TP}_F(D, \tau, \sigma, \beta)$ given only the collection of frequent itemsets/association rules for the items.[2]

3.4 Algorithms

Proposition 3.1 allows a standard level-wise algorithm to be used for computing trees in $\mathcal{TP}_F(\mathcal{D}, \tau, \sigma, \beta)$, as when computing frequent itemsets (recall Algorithm 1 in Section 2.2): start from single attributes, and on every pass combine trees of size k into trees of size $k+1$.

However, the combination phase for trees is not as simple as for itemsets. Fortunately however, there are several existing methods for level-wise type of tree enumeration stemming from research on frequent subtree discovery from relational databases (see [12] for an overview). In this study, we implement the approach by Zaki [96], mainly because of its simplicity with respect to other existing methods. For a more detailed description of the method, the reader is referred to [96]. Here we give a simplified overview.

Briefly, a tree is represented as a string by traversing it depth-first in preorder, recording the attribute in each node, and -1 when backtracking. Figure 3.5 provides an example of the encoding. In this encoding, it is sufficient to consider combining pairs of trees sharing the same $(k-1)$-prefix, as opposed to trying to

[1]See, e.g., [89, 58, 59, 53, 48] for interesting work on post-processing collections of association rules.

[2]The exponential collection of frequencies of *all* frequent itemsets for frequency threshold 0 specifies the distribution of the data rows uniquely, so that exponential input would also suffice to determine the collection $\mathcal{TP}_F(D, \tau, \sigma, \beta)$.

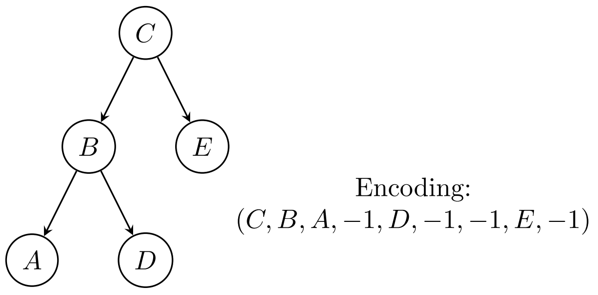


Figure 3.5: Example of tree encoding by Zaki [96].

combine all pairs of trees. Consequently, the otherwise quadratic time complexity is cut down.

A drawback of the Zaki combination method is that each $(k+1)$-tree is generated multiple times as isomorphic copies: for example, the isomorphic trees $(A, B, -1, C, -1)$ and $(A, C, -1, B, -1)$ are both generated. We optimize the database pass by only accessing the database for trees where the children of each node are in alphabetical order, and using the same information for isomorphic trees. However, it is not possible to completely prune the copies. For example, the 4-tree $(A, C, -1, D, B, -1, -1)$ can only be generated from the 3-trees $(A, C, -1, D, -1)$ and $(A, C, -1, B, -1)$, the latter of which does not have the order property. Another possibility would be to work only with canonical forms of trees, as in [5, 72, 13]. On the other hand, for canonical forms of trees the combination phase is conceptually more complex.

3.5 Experiments

To test the tree mining method in practice an implementation of the Zaki's algorithm (discussed in Section 3.4) was built using Java language. The results of the experiments are reported in this section.

3.5.1 Examples of trees

Some examples are now presented of the kind of individual tree patterns that result from real data. The data sets MovieLens, Mammals, and Course Enrollment are used in the examples.

MovieLens data

We start with the MovieLens data by exploring frequent trees in comparisons with the use of only regular frequent itemset. To do this we mined all frequent itemsets using a frequency of 100 rows (relative frequency = 0.106) and ranked the itemsets according to the ratio of their observed frequency with respect to their expected frequency (computed from marginal frequencies assuming independence of attributes). The top three itemsets are the following:

1. {Back to the Future (1985), Empire Strikes Back (1980), Indiana Jones and the Last Crusade (1989), Raiders of the Lost Ark (1981), Return of The Jedi (1983), Star Wars (1977)}

2. {Alien (1979), Aliens (1986), Blade Runner (1982)}

3. {Godfather (1972), Pulp Fiction (1994), Shawshank Redemption (1994), Silence of the Lambs (1991)}

The first itemset features three Star Wars movies and two Indiana Jones movies, all films by the producer-director George Lucas. The item Back to the Future (1985) is a movie directed by Steven Spielberg, as are the two Indiana Jones movies. The second itemset includes the movies Alien (1979) and Blade Runner (1982), both science fiction movies by Ridley Scott, with Aliens (1986) being the sequel to Alien (1979). The third itemset features the drama and crime thriller movies Godfather (1972), Pulp Fiction (1994), Shawshank Redemption (1994) and Silence of the Lambs (1991).

To see the more structural view provided by the frequent tree pattern class, we computed the best tree, according to the conflict ratio score, for each of the three itemsets. The search was restricted to trees with a maximum of 3 children per node, that is $\beta \leq 3$. The resulting trees are shown in Figure 3.6.

The tree structures bring out thematical and temporal relationships. The trees contain altogether four sequel movies, (Empire Strikes Back (1980), Return of The Jedi (1983), Indiana Jones and the Last Crusade (1989) and Aliens (1986)), all occurring as children of the original movie (Star Wars (1977), Raiders of the Lost Ark (1981) and Alien (1979)). Furthermore, movies with no sequel relations show temporal ordering; for instance, the movie Star Wars (1977) is parent to the movie Raiders of the Lost Ark (1981), which premiered four years later. None of these relationships are directly visible from the plain itemset patterns.

Figure 3.7 shows the top-3 disjoint frequent trees, now using the frequent tree mining method. For the result we used an occurrence count threshold of $\sigma \geq 0.1$, a conflict count of $\tau \leq 0.075$ and a maximum number of children of $\beta \leq 2$. The trees were ranked first according to size and then conflict ratio.

The common denominator for the frequent itemsets seemed to be more related to the makers of the movies, that is, George Lucas, Steven Spielberg and Ridley Scott. The trees in Figure 3.7, on the other hand, show perhaps more coherence in genre/theme. The tree in Figure 3.7(a) has a strong sci-fi theme with two Star Wars and two Star Trek movies, the Star Trek movies in particular being more

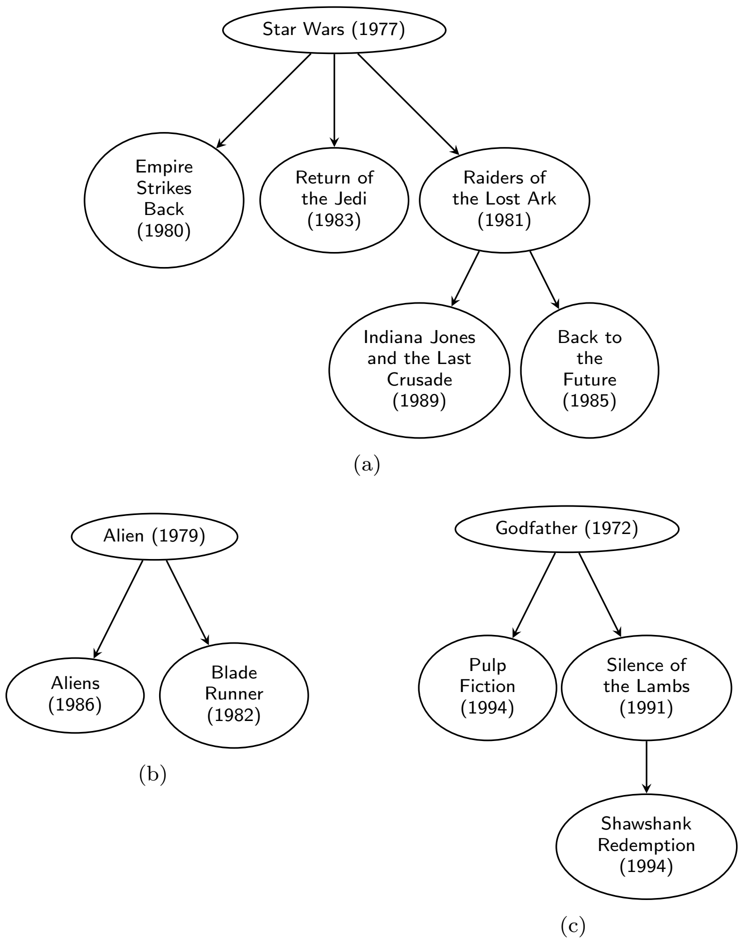

Star Wars (1977)
Empire Strikes Back (1980)
Return of the Jedi (1983)
Raiders of the Lost Ark (1981)
Indiana Jones and the Last Crusade (1989)
Back to the Future (1985)
(a)
Alien (1979)
Aliens (1986)
Blade Runner (1982)
(b)
Godfather (1972)
Pulp Fiction (1994)
Silence of the Lambs (1991)
Shawshank Redemption (1994)
(c)

Figure 3.6: Tree structure revealed for itemsets in the dataset MovieLens. The itemsets are the top-3 disjoint sets with a frequency threshold of 100 rows (relative frequency = 0.106). The tree structures for each of the itemsets have been selected by maximizing the conflict ratio for trees with at most 3 children for each node. Trees (a), (b) and (c) have conflict ratios 3.5, 2.7 and 1.8, respectively.

niche items for a specific sci-fi audience. The tree in Figure 3.7(b) features a set of more "testosterone backed" action movies, such as the war movie Full Metal Jacket (1987) by Stanley Kubrick, the high tempo action movies True Lies (1994) and The Terminator (1984) by James Cameron, and the action adventure movie Raiders of the Lost Ark (1981) by Steven Spielberg and George Lucas. Correspondingly, the tree in Figure 3.7(c) shows a common Italian-American theme with the two Mafia movies The Godfather (1972) and The Godfather: Part II (1974), the movie

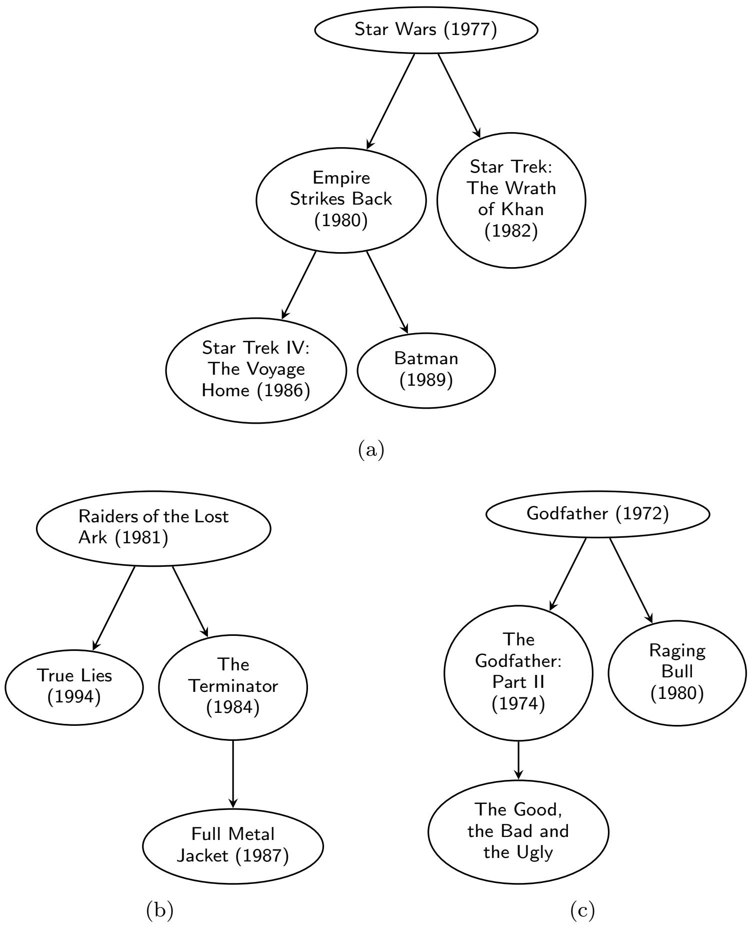
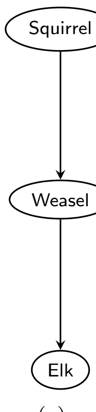
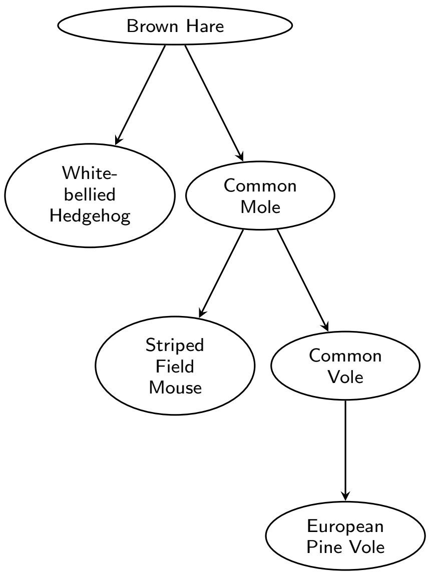

Star Wars (1977)
Empire Strikes Back (1980)
Star Trek: The Wrath of Khan (1982)
Star Trek IV: The Voyage Home (1986)
Batman (1989)
(a)
Raiders of the Lost Ark (1981)
True Lies (1994)
The Terminator (1984)
Full Metal Jacket (1987)
(b)
Godfather (1972)
The Godfather: Part II (1974)
Raging Bull (1980)
The Good, the Bad and the Ugly
(c)

Figure 3.7: Top-3 disjoint frequent trees from the MovieLens data computed from the set of frequent trees having a maximum node degree of 2, occurrence count threshold $\sigma \geq 0.1$ and conflict count threshold $\tau \leq 0.075$. The ranking is based on first size and then conflict ratio. The trees (a), (b) and (c) have the conflict ratios of 5.5, 4.0 and 3.2, respectively.

Raging Bull (1980), a story about an Italian-American boxer, and the well known spaghetti western The Good, the Bad and the Ugly. Again, sequel and temporal relationships are featured in all of the trees, with for instance, the movies Star Wars (1977) and Empire Strikes Back (1980), as well as The Godfather (1972) and The Godfather: Part II (1974).



(a)



(b)

Figure 3.8: Frequent trees found in MAMMALS. Tree (a) shows the best tree structure for the itemset {Squirrel, Weasel, Elk} according to the conflict ratio score. Tree (b) is the best tree according conflict ratio in the set of maximal trees having occurrence count $0.2 \leq \sigma \leq 0.8$, conflict count $\tau \leq 0.11$ and $\beta \leq 2$. Tree (a) has a conflict ratio of 2.72, while tree (b) has a conflict ratio of 5.09.

Mammals data

Figure 3.8(a) shows the best tree structure according to the conflict ratio score for the itemset {Squirrel, Weasel, Elk} found[3] in MAMMALS. The tree structure found again reveals something not perceivable from the plain itemset: the three species in question have a *nested* relationship [62]. That is, if Elk is present, Weasel and Squirrel are rarely absent (conflict). Furthermore, if Weasel is present, Squirrel is rarely absent. In other words, the occurrence range of Elk is more or less in a subset relationship with the occurrence range of Weasel, which is in turn in a subset relationship with the range of Squirrel. Nestedness is a phenomenon studied extensively in the ecological domain [75, 6, 20, 45].

Figure 3.8(b) shows the best tree according to conflict ratio, for trees mined using an occurrence count threshold of $0.2 \leq \sigma \leq 0.8$, a conflict count threshold

[3]In the experiments of Chapter 6, Section 6.5.1, we present a clustering of the dataset MAMMALS, based on real-valued environmental observations and a subset of itemsets computed from the binary observations of the data. The itemset {Squirrel, Weasel, Elk} turns out to characterize a Nordic cluster defining a region covering the geographical areas of Finland and Sweden (see Figure 6.3 and Table 6.1).

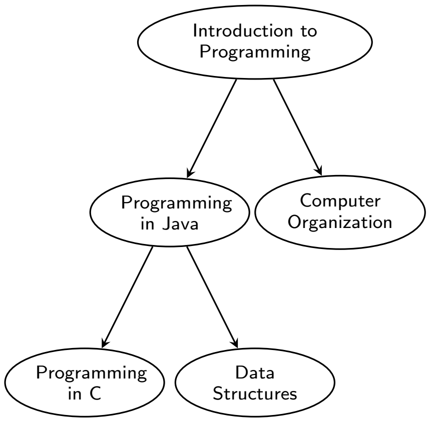


Figure 3.9: Best tree according conflict ratio and size in the dataset Course Enrollment. The result was obtained using occurrence count $\sigma \geq 0.09$, conflict count $\tau \leq 0.165$ and $\beta \leq 2$. The tree has a conflict ratio of 2.20.

of $\tau \leq 0.11$ and a branching constraint of $\beta \leq 2$. The root Brown Hare is a common species all around Europe, while the leafs feature White-bellied Hedgehog and Striped Field Mouse, both species occurring mostly only in eastern Europe. Moreover, as their English language species names suggest, the species European Pine Vole is a more habitat-specified species than the more general Common Vole.

Course enrollment data

For course enrollment there is an ordering in which the Department of Computer Science recommends students to take certain courses. For instance, some courses require only a basic understanding of programming concepts, whereas other courses have more specific prerequisites. Figure 3.9 shows the best tree with the largest number of items together with the best conflict ratio among trees having occurrence count $\sigma \geq 0.09$, conflict count $\tau \leq 0.165$ and $\beta \leq 2$. The tree nicely reflects the fact that the more advanced courses Data structures and Programming in C have Java programming as a prerequisite, whereas for Computer Organization, the course Introduction to Programming suffices. The order presented in the example tree is also the order in which the department recommends these courses to be taken.

3.5.2 Tree pattern validation

A good pattern model should find structure where it exists, and produce little or no patterns for attribute sets for which no structure is present. To validate our method, a generated dataset with specific "planted" trees is used to see whether the artificially placed structure can be found using our tree pattern model. The

Table 3.1: Results from generated data. m: number of attributes; $|\mathcal{S}|$: number of trees used in the generating process; σ, τ: thresholds for frequency and conflicts; $|\mathcal{TP}_F|$: size of the output set; $|\mathcal{R}|$: size of the final result set obtained by taking enough trees to cover $\mathcal{S}$.

m	$\|\mathcal{S}\|$	σ	τ	$\|\mathcal{TP}_F\|$	$\|\mathcal{R}\|$
10	2	0.2	0.30	1343	5
14	3	0.2	0.30	1172	8
18	4	0.2	0.30	1965	21
20	5	0.2	0.30	2208	27
23	6	0.2	0.30	1674	45
28	7	0.2	0.30	5469	43

method was also tested with swap randomization [29] using the MovieLens dataset. The randomization procedure preserves the row and column margins of the given dataset, but obscures the internal dependencies of the data. The idea is to see whether the number and quality of patterns between the original data and the swap randomized data differ. If the true structure in the data is captured by the pattern class, there should be significant differences between the models found on the original and randomized datasets.

Generated data

Data were generated using the following procedure. First, a number of disjoint trees with different values of the specificity measure were created by hand. The number of trees used for the experiment was varied by taking different subcollections of the trees. Given such a collection $\mathcal{S}$, data was produced as follows. A row t was generated by first making all attributes of t equal to 0. Then, each tree $T \in \mathcal{S}$ was selected with probability p. If T was selected, we sampled a subset X of the nodes of T by taking each node with probability q. Letting Y be the set of all ancestors of nodes in X, we let $t(A) = 1$ for all $A \in X \cup Y$. Finally, each bit in the dataset was flipped independently with probability r to create noise. The parameter values used were $p = q = 0.5$ and $r = 0.1$, and 1000 data rows were generated.

From the data generated, all the frequent trees were mined. The parameter σ was chosen to be 0.2, since each tree has a $p = 0.5$ chance of occurring, and each attribute in the tree may have as low as a $pq = 0.25$ chance of occurring. The parameter τ was chosen as large as possible so that a reasonable number of trees was still obtained; typically $\tau = 0.3$ was close to the limit.

The result set $\mathcal{TP}_F$ was reasonably small, while containing all the trees in $\mathcal{S}$. In order to test how well these trees were positioned in the results with respect to the two interestingness measures, We partitioned the mined trees into classes by their number of ancestor-descendant pairs ϕ, and each class was sorted by the conflict ratio. From each class ϕ, trees were selected into the final result set $\mathcal{R}$ in decreasing order of conflict ratio until all trees in $\mathcal{S}$ had been selected. The size $|\mathcal{R}|$ of the final result set is thus a measure of how close to the top the generating trees in $\mathcal{S}$ were. The results are shown in Table 3.1.

We see that in every case the number of trees one needs to examine in order to find the generating trees is fairly low. In fact, most of the extra trees are variations of the generating trees: for example, in the smallest dataset, one of the two generating trees is found immediately, and three of its simple variants precede the other generating tree in conflict ratio order. The sensitivity to the parameter is evident in that, while we have $\sigma = 0.2$ and $\tau = 0.3$ in all the cases shown, the size of the output $|\mathcal{TP}|$ varies non-monotonically in m.

Swap randomized data

We used the MovieLens dataset in the swap randomization test. The number of swaps in the procedure was equivalent to the number of rows multiplied by the number of columns in the original data. A set of 100 swap randomized data instances was produced. We generated a collection of frequent trees using the occurrence count threshold $\sigma \geq 0.2$ and the conflict count threshold $\tau \leq 0.15$ and $\beta \leq 2$, for the original data and each of the 100 swap randomized data instances.

Figure 3.10 shows the results of the swap randomization tests. Figure 3.10(a) shows the distribution of conflict ratios in the original MovieLens dataset. To compare this with the patterns found in the swap randomized data we took a 1% random sample of the entire set of patterns generated from 100 swap randomized dataset instances. The distribution of conflict ratios resulting from these patterns is shown in the same subfigure with the dashed outline. A clear difference can be noticed in these two distributions.

To obtain an empirical p-value [29, page 4] we compared two pattern statistics in the original data with the corresponding statistics in the 100 swap randomized data instances: the 100th best conflict ratio and the total size of the pattern set. Figure 3.10(b) shows the histogram of the 100th best confliction ratio in the swap randomized datasets, while Figure 3.10(c) shows the histogram for the size of the pattern output set. The values of the corresponding statistics in the original dataset are marked with an arrow. In both cases an empirical p-value of 0.001 is obtained.

3.5.3 General statistics

Finally some general mining statistics are given in Table 3.2 for the three datasets: MovieLens, Mammals and Course Enrollment. We see that the number of elements in the answer increases rapidly with decreasing frequency threshold σ and increasing conflict threshold τ. It should be noted, however, that it is quite easy to iteratively find values of σ and τ that produce outputs of the desired size. Also, retaining trees with a low branching factor efficiently prunes the answer set and search space. The run-times are feasible.

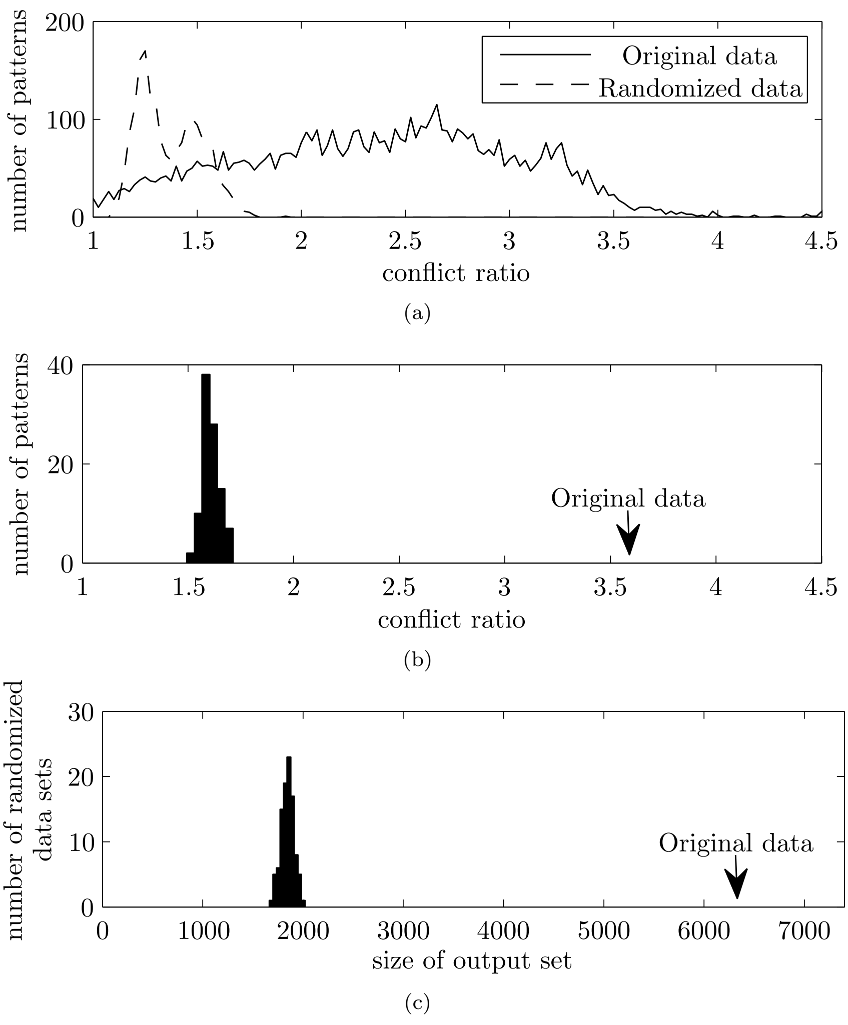


Figure 3.10: Comparison between the original MovieLens data and corresponding swap randomized data. Figure (a) shows the empirical distribution of conflict ratios of all patterns generated from the original data, in comparison to a 1% random sample on the set of patterns generated from 100 swap randomized dataset instances. Figure (b) shows the histogram of the 100th best conflict ratio over the 100 swap randomized datasets. Figure (c) shows the histogram for the corresponding sizes of the pattern output sets.

Table 3.2: Frequent tree mining statistics for the datasets MovieLens, Mammals and Course Enrollment. The number of trees in the collection $\mathcal{TP}(D, \tau, \sigma)$ for various values of τ, σ and the branching factor β. $k = \max |T|$, the largest tree in the pattern set. Time/sec = algorithm running time in seconds. Gen Cands = number of generated candidate tree patterns. *For the Mammals data an occurrence count threshold of $0.2 \leq \sigma \leq 0.8$ was used.

Dataset	σ	τ	β	$\lvert\mathcal{TP}\rvert$	k	Time/sec.	Gen Cands
Mammals	0.2*	0.075	2	1023	5	7	9124
Mammals	0.2*	0.075	3	2198	6	36	20266
Mammals	0.2*	0.1	2	2984	5	25	22145
Mammals	0.2*	0.1	3	8811	7	867	80729
Mammals	0.2*	0.11	2	4651	6	46	32887
Mammals	0.2*	0.11	3	15644	7	2773	148483
MovieLens	0.2	0.075	2	219	3	2	6050
MovieLens	0.2	0.075	3	255	4	3	8036
MovieLens	0.1	0.05	2	4763	4	158	149032
MovieLens	0.1	0.05	3	9329	4	2159	390076
MovieLens	0.1	0.055	2	7384	4	433	203800
MovieLens	0.1	0.055	3	18753	5	10745	686266
MovieLens	0.1	0.075	2	31935	5	13602	869238
Course E.	0.085	0.14	2	888	4	17	9560
Course E.	0.085	0.14	3	1023	5	16	13080
Course E.	0.08	0.15	2	4895	5	76	52237
Course E.	0.08	0.15	3	6348	6	234	88607
Course E.	0.07	0.145	2	34946	6	3056	528704
Course E.	0.07	0.165	2	127381	6	31432	2036887

3.6 Related work

While a vast amount of research has been conducted into finding trees from relational tree- or graph-structured data [12], the crucial difference here is that we start from unstructured 0-1 data, as in mining frequent sets and association rules [3]. Hierarchical clustering [49, 46, 92] or finding phylogenetic trees [23] seeks to find trees from 0-1 data, but typically the goal is to find one tree containing all the attributes of the data. The same is true for finding tree-structured Bayes nets [18, 56, 37, 84] from data. Another difference is that in a hierarchical clustering or phylogenetic trees all attributes of the data are in the leaves of the tree. We seek trees where all nodes of the tree are attributes from the data.

Another somewhat analogous class of patterns are approximate itemsets, such as error-tolerant or dense itemsets [94, 80, 78]. These are relaxed versions of frequent itemsets: the set is considered to occur in a row provided most of the attributes of the set are 1 in the row. Similarly to these patterns, in this work a tree can be supported by rows that do not have all of the items of the tree. The tree structure reflects more closely the kinds of co-occurrence that are in fact present in the data. Fragments of order [28] are a type of directed itemset: a fragment is violated by rows having two of its items but lacking at least one item that appears between the two. Fragments of order can be viewed as simple unrooted trees having only one branch.

3.7 Conclusions

In this chapter we introduced the idea of mining trees from unordered binary data and showed that this pattern class is distinct from traditional frequent itemsets or association rules. We also showed that the method has advantageous properties: high quality tree patterns are unlikely to occur in random data. The definition allows a simple level-wise algorithm for mining all frequently occurring trees. Empirical results prove that the level-wise algorithm can find interesting trees in real data, and relevant patterns in generated data. The extra structure provided by the new pattern class can help the data analyst to make further conclusions on the relationships between the attributes not possible with traditional frequent itemsets.

Chapter 4

Low-entropy sets and trees

4.1 Introduction

Pattern classes such as frequent itemsets stress the co-occurrence of the value 1 in the data. Take for instance the itemset $X = \{A, B, C, D\}$. We say that X is frequent if there are sufficiently many data rows t that have a value combination in the attributes such that $t(A){=}t(B){=}t(C){=}t(D){=}1$. But, why concentrate only on the co-occurrences of the value 1? The data could hide a potentially interesting subset of attributes that have some other type of dependency structure.

As an example, consider the dataset Mammals. An ecologist might be interested in species occurrence dynamics from some other point of view than just plain co-occurrence. It could be relevant, for instance, to conclude that X comprises two (or more) prominent value combinations. The case could be that when A and B occur, C and D almost never occur, and when C and D occur, only very rarely do A and B occur. Occurrence behavior like this could stem from competition over scarce resources, or some other environmental factors, and would not be possible to detect with frequent itemsets. Table 4.1 gives an example of a distribution in which the value combinations occur like this.

Instead of concentrating on attribute sets with one frequent value combination, in this chapter we discuss a more general approach. Given an itemset X, we say that X is interesting if the attributes of X have low overall complexity. As a measure of complexity we take *entropy*, which can be used to describe the skewness of the distribution according to which the value combinations of X occur. The lower the entropy, the more structured and more concentrated the value combinations are. We call attribute sets with entropy below some predefined threshold *low-entropy sets*.

Low-entropy sets can be viewed as a generalization of frequent itemsets in the sense that they are not restricted to co-occurrences of the value 1 in the data. Compared with frequent itemsets they are also more expressive in that they can locate subsets that have a number of different value combinations, such as in the example of Table 4.1. Also, unlike frequent itemsets, low-entropy sets are symmetric with respect to 0 and 1, that is, flipping 0s to 1s and vice versa will not change the score.

While it can be argued that the property of having low entropy is interesting in itself, it is also beneficial to have an explanation of how the attributes in the set are connected to each other, as already discussed in Chapter 3. Fortunately, it is easy to extend the notion of entropy to structural patterns such as trees. A *low-entropy tree* is a subclass of a tree-structured network with an entropy score below a predefined threshold. Low-entropy trees can be seen as Bayes trees [37], Markov trees [76], dependence trees [64], or Chow-Liu trees [14], but an important distinction is that we do not attempt to model the complete joint distribution but to find interesting local patterns.

In this chapter we give formal definitions to two new pattern classes: low-entropy sets and low-entropy trees. We show that entropy has the monotonicity property, and thus a level-wise approach can find all low-entropy sets. We also show that the low-entropy trees are bounded above by the entropy of the corresponding set, allowing similar algorithms to be used for finding low-entropy trees. We describe algorithms for finding all patterns satisfying an entropy condition for both pattern types. We give an empirical evaluation of patterns found by these methods and compare their properties with results with frequent itemsets. We show that entropy allows to express structure related not only with single types of co-occurrence but also with more general occurrence patterns.

4.2 Problem definitions

4.2.1 Low-entropy sets

Given a set of items X, we denote by $\pi_X(t)$ the projection of the transaction t onto X. In other words, $\pi_X(t)$ is a 0–1 vector of values $t(A)$ defined by the attributes $A \in X$.

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

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

Or, simply put, the fraction of transactions in $\mathcal{D}$ where i fits. The entropy of an itemset X in $\mathcal{D}$ is

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

where $0 \log_2 0$ is assigned the value 0 by convention. For readability, we write $p_X(i)$ and $H(X)$ wherever $\mathcal{D}$ is clear from the context.

Table 4.1: Example low-entropy set $\{A, B, C, D\}$. The distribution of the set is concentrated on two frequent value combinations.

A	B	C	D	value counts
1	1	0	0	434
0	0	1	1	398
0	0	0	0	88
0	1	0	0	34
0	0	0	1	22
1	0	0	1	14
1	1	1	1	10
Number of rows in data				1000

The entropy of X can be seen as a measure of uncertainty about the value of $\pi_X(t)$ if we were to randomly sample a row t from $\mathcal{D}$. Intuitively, entropy is increased by outcomes that are unlikely, as $-\log_2 p_X(i)$ will be high in this case. However, since these contributions are weighted by their frequencies, a single unlikely outcome contributes little. Hence, entropy is only maximized when each outcome is equally unlikely. Indeed, the entropy function $H(X)$ has its maximum value when all probabilities $p_X(i)$ are equal. In this case $H(x) = \log_2 |\Omega_X| = |X|$.

Correspondingly, entropy is small when most outcomes are not surprising. The lower the entropy, the more structured and more concentrated the instantiations are. The minimum value $H(X) = 0$ is achieved when only a single instantiation occurs in the data, i.e., when $p(i) = 1$. This can be contrasted with frequent itemsets: an itemset X has maximum frequency if $p_X(i) = 1$ for $i = \vec{1}$. In this case the entropy of X is also minimized. More generally, high frequency implies low entropy, but not necessarily vice versa. From a pattern mining point of view, attribute sets exhibiting structure in the form of low entropy can be deemed interesting. For more on the basic properties of entropy, see e.g. [19, Chapter 2].

Definition 4.1 *Given an entropy threshold ϵ, a low entropy set X (LE-set) is an itemset that has entropy lower than ϵ in $\mathcal{D}$, that is $H(X, \mathcal{D}) \leq \epsilon$. The collection of all low-entropy sets in $\mathcal{D}$ is denoted by $\mathcal{P}_{LE}(\mathcal{D}, \epsilon)$.*

Problem 4.1 *Given $\mathcal{D}$ and ϵ, compute $\mathcal{P}_{LE}(\mathcal{D}, \epsilon)$.*

4.2.2 Low-entropy trees

A potential drawback of mining low-entropy sets is that the sets do not have any structure that would explain how the attributes in the set are connected to each

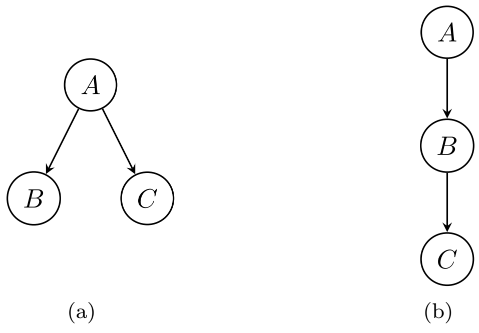


Figure 4.1: Two example trees.

other, a topic already discussed in Chapter 3. In this subsection we consider replacing itemsets by a tree-type pattern; the pattern imposes a model on the attribute sets, and it is the entropy of the model that we seek to minimize.

Consider the tree $T = (X, E)$. Furthermore, denote the root of T with R. We define the entropy of T as

$$H_{\mathcal{T}}(T) = H(R) + \sum_{(A,B) \in E} H(B \mid A), \tag{4.1}$$

where $H(B \mid A)$ is the *conditional entropy* of B given A, that is,

$$H(B \mid A) = H(\{A, B\}) - H(A).$$

Conditional entropy may be interpreted as the amount of uncertainty that remains about the value of $t(B)$ if the value of $t(A)$ is known, when randomly sampling a row t from $\mathcal{D}$.

Example Consider Figure 4.1. The tree in Figure 4.1(a) has entropy

$$H(A) + H(B \mid A) + H(C \mid A),$$

while the tree in Figure 4.1(b) has entropy

$$H(A) + H(B \mid A) + H(C \mid B).$$

The tree in Figure 4.1(a) will have lower entropy than the tree in Figure 4.1(b) if $H(C \mid A) < H(C \mid B)$. The interpretation of this would be that knowing $t(A)$ reduces one's uncertainty about $t(C)$ more than knowing $t(B)$ would. Indeed, in general, we would like to have trees, in which each child node is as closely determined by its parent as possible. In other words trees that capture the interaction structure of the attributes as well as possible. We say that tree $T = (X, E)$ is *optimal* for itemset X if the set of edges E minimizes $H_{\mathcal{T}}$ for the itemset X. In other words, $H_{\mathcal{T}}(T) \leq H_{\mathcal{T}}(T')$, for all trees $T' = (X, E')$, such that $E' \neq E$.

Definition 4.2 *Given an entropy threshold ϵ, a* low-entropy tree *(LE-tree) is an optimal tree $T(X, E)$ for itemset X that has entropy lower than ϵ in $\mathcal{D}$, that is $H_{\mathcal{T}}(T) \leq \epsilon$ and $H_{\mathcal{T}}(T) \leq H_{\mathcal{T}}(T')$ for all trees $T' = (X, E')$ such that $E' \neq E$. The collection of all low-entropy trees in $\mathcal{D}$ is denoted by $\mathcal{TP}_{LE}(\mathcal{D}, \epsilon)$.*

We have the following computational problem:

Problem 4.2 *Given $\mathcal{D}$ and ϵ, compute $\mathcal{TP}_{LE}(\mathcal{D}, \epsilon)$.*

4.3 Problem properties

Next we consider the basic properties of Problems 4.1 and 4.2. We start with monotonicity.

4.3.1 Monotonicity

Proposition 4.1 *For all datasets $\mathcal{D}$,*

$$H(X) \geq H(X \setminus A)$$

for all $A \in X$ and $X \neq \emptyset$.

Proof We use $H(X \mid Y) = H(X \cup Y) - H(Y)$, and the fact that entropy is always nonnegative.

$$H(X) = H(X \setminus A) + H(A \mid X \setminus A) \geq H(X \setminus A)$$

Note that the equality $H(X) = H(X \setminus A)$ holds only when $H(A \mid X \setminus A) = 0$, i.e., when there is a functional dependency $X \setminus A \to A$. $\square$

4.3.2 LE-trees versus LE-sets

The following states that any low-entropy tree is necessarily a low-entropy set.

Proposition 4.2 *Given any tree $T = (X, E)$, we have $H(X) \leq H_{\mathcal{T}}(T)$.*

Proof To see this, consider marking each node $A \in X$ with an index $i(A) \in 1, \ldots, |X|$, such that for each edge $(A, B) \in E$ we have $i(A) < i(B)$. This can be done easily by traversing the nodes in the tree in a breath-first manner starting from the root, and at each step assigning index i to the ith node visited. For readability we write A_i when we have $i(A)$. The entropy of tree T can now be written out as

$$H_{\mathcal{T}}(T) = H(A_1) + \sum_{i=2}^{|X|} \sum_{(A_i, A_j) \in E} H(A_j \mid A_i). \tag{4.2}$$

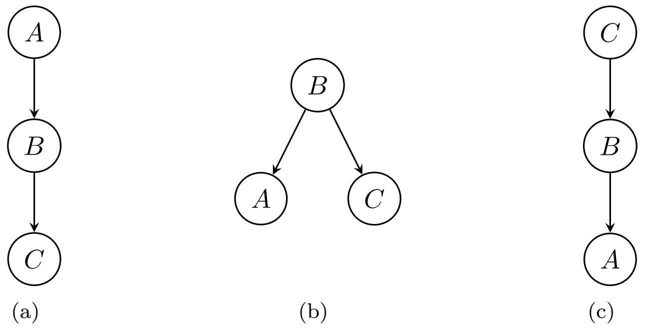


Figure 4.2: All trees in the figure are equal in terms of $H_{\mathcal{T}}$.

Conditioning reduces entropy [19, Theorem 2.6.5]. Hence, by replacing each $H(A_j \mid A_i)$ with $H(A_j \mid \{A_{j-1}, \ldots, A_1\})$, we can only decrease the value of the sum. Note that as $(A_i, A_j) \in E$ we have $A_i \in \{A_{j-1}, \ldots, A_1\}$. Thus,

$$\begin{aligned} H_{\mathcal{T}}(T) &\geq H(A_1) + H(A_2 \mid A_1) + \ldots \\ &\quad + H(A_{|X|} \mid \{A_{|X|-1}, \ldots, A_2, A_1\}) \\ &= H(X) \end{aligned}$$

The final equality follows from the chain rule of entropy [19, Theorem 2.5.1].

□

The proposition implies that one possible way to mine low-entropy trees is to mine first the low-entropy itemsets using e.g. the level-wise search and then fit tree structures into each itemset.

Notice also that in the proof of Proposition 4.2 we have equality only for trees of size two. In that case $Y = \{A_i\}$. For equality to hold in the general case it would require a fully connected Bayesian network.

Corollary 4.1 *Let $G = (X, E)$ be a fully connected directed acyclic graph, interpreted as a Bayesian network whose every node has the maximum-likelihood distribution from the data $\mathcal{D}$. Then we have $H(X) = H(G)$.*

4.3.3 Symmetry of entropy

A known property of entropy is symmetry, from which it follows that the score $H_{\mathcal{T}}(T)$ is not dependent on the choice of the root for the tree $T = (X, E)$. Picking any node as root and then redirecting links away from this root will results in an equivalent entropy value $H_{\mathcal{T}}$. As an example, consider the trees in Figure 4.2. Due to symmetry, all the trees in the figure are equal in terms of $H_{\mathcal{T}}$. Consequently, the undirected tree in Figure 4.3 can be used to depict the common structure of these trees.

To see this, consider a tree $T = (X, E)$. We define the *degree* $deg(A)$ of node $A \in X$ as the number of edges incident to A, that is

$$deg(A) = |\{A \in X \mid (A, B) \in E \vee (B, A) \in E\}|.$$

It turns out that $H_{\mathcal{T}}(T)$ can be expressed as a sum depending only on the degrees of the nodes $A \in X$ and the joint entropies $H(\{A, B\})$ for all edges $(A, B) \in E$. Clearly, neither $deg(A)$ nor $H(\{A, B\})$ are quantities that depend on whether we have $(A, B) \in E$ or $(B, A) \in E$.

Proposition 4.3 *Consider the tree $T = (X, E)$. The entropy of T can be expressed as*

$$H_{\mathcal{T}}(T) = \sum_{(A,B)\in E} H(\{A, B\}) - \sum_{A\in X} (deg(A) - 1)H(A).$$

Proof Denoting the root of T by R, the entropy $H_{\mathcal{T}}(T)$ of T is

$$\begin{aligned} H_{\mathcal{T}}(T) &= H(R) + \sum_{(A,B)\in E} H(B \mid A) \\ &= H(R) + \sum_{(A,B)\in E} H(\{A, B\}) - \sum_{(A,B)\in E} H(A) \\ &= H(R) + \sum_{(A,B)\in E} H(\{A, B\}) - \sum_{A\in X} \sum_{(A,B)\in E} H(A) \\ &= H(R) + \sum_{(A,B)\in E} H(\{A, B\}) - \sum_{A\in X} (H(A) \sum_{(A,B)\in E} 1) \end{aligned}$$

For each node $A \in X \setminus R$, we have exactly 1 edge connection to its parent and $deg(A) - 1$ connections to each of its child nodes B. Thus, for a fixed node $A \neq R$,

$$\sum_{(A,B)\in E} 1 = |\{\, (A, B) \mid (A, B) \in E \,\}| = deg(A) - 1.$$

As the root R has no parents, we have $\sum_{(R,B)\in E} 1 = deg(R)$. From this it follows that

$$\begin{aligned} H_{\mathcal{T}}(T) &= \sum_{(A,B)\in E} H(\{A, B\}) - \sum_{A\in X\setminus R} (deg(A) - 1)H(A) - deg(R)H(R) + H(R) \\ &= \sum_{(A,B)\in E} H(\{A, B\}) - \sum_{A\in X} (deg(A) - 1)H(A). \end{aligned}$$

□

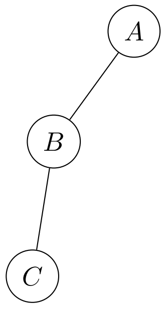


Figure 4.3: An undirected version of the trees in Figure 4.2.

4.4 Algorithms

4.4.1 Low-entropy sets

Proposition 4.1 implies that low-entropy sets can be found using techniques similar to those used in frequent itemsets mining (recall Algorithm 1 in Section 2.2). Still, in practice, one seeming complication in mining low-entropy sets is that the entropy $H(X)$ is a sum over all possible instantiations that the attributes in X can have, which involves $2^{|X|}$ different values. However, because of the convention $0 \log 0 = 0$ used in the definition of entropy, we only need to count those instantiations that appear in the data. Thus the database pass can be done using $O(c\ell n \log n)$ auxiliary space, where c is the number of candidate itemsets being examined, ℓ is the size of the candidates, and n is the number of records in the data. For each candidate X, we keep track of the instantiations seen and a count for each; there are at most n instantiations, each requiring space $O(\ell)$, and each count takes $O(\log n)$ space. If the data fits into main memory, then the entropy computation can easily be performed individually for each itemset.

An additional noteworthy observation is that the pruning strategies used in maximal frequent itemset mining algorithms (recall Subsection 2.2.4) may also be used as an alternative. Storing the instantiation counts of each maximal LE-set, and then by count marginalization, it is possible to generate the full pattern lattice without full database passes for the submaximal patterns. Indeed, for many real-life mining tasks, the maximal pattern set may suffice as satisfactory output result.

4.4.2 Low-entropy trees

We next turn to the question of how to mine low-entropy trees. As noted above in Proposition 4.2, the entropy of the set gives a lower bound for the entropy of the best LE-tree. However, the lower bound can be rather loose.

As we look for the optimal tree for a given set of attributes X, it turns out that this can be obtained by adding an edge to the optimal tree for $X \setminus A$ for some $A \in X$.

Proposition 4.4 *Let $T = (X, E)$ be the optimal tree for the itemset X. Furthermore, let S be the tree resulting from removing some leaf node $B \in X$ and the corresponding edge $(A, B) \in E$ from T. For all leaf nodes $B \in X$, the tree S is the optimal tree for the itemset $X \setminus B$.*

Proof Let $T = (X, E)$ be the optimal tree for itemset X, that is, the edges E minimize $H_{\mathcal{T}}$ for the itemset X. Furthermore, let S be the tree resulting from removing some leaf node B and an edge $(A, B) \in E$ from T. In other words, $S = (X \setminus B, F)$, where $F = E \setminus (A, B)$.

We can compute the entropy of T by

$$H_{\mathcal{T}}(T) = H_{\mathcal{T}}(S) + H(B \mid A).$$

Now assume that there exists some other tree $S' = (X \setminus B, F')$, such that $F \neq F'$ and $H_{\mathcal{T}}(S') < H_{\mathcal{T}}(S)$. Adding the node B and edge (A, B) to S' will result in a tree $T' = (X, F' \cup (A, B))$, such that

$$H_{\mathcal{T}}(T') = H_{\mathcal{T}}(S') + H(B \mid A) < H_{\mathcal{T}}(S) + H(B \mid A) = H_{\mathcal{T}}(T).$$

Hence, we have $H_{\mathcal{T}}(T') < H_{\mathcal{T}}(T)$, a contradiction. $\square$

One way to find low-entropy trees is therefore to conduct a level-wise breadth-first search. For each k-size tree $T = (X, E)$, we extend T by trying to connect each $A \in \mathcal{I} \setminus X$ to each X in T. Hence, there are $|\mathcal{I} \setminus X|\,|X|$ possible extensions of T. Note also that as the tree entropy score consists only of pairwise entropies, after level 2 the score evaluation can be done in constant time without looking into the data. Algorithm 3 gives a pseudo-code description of the level-wise search for finding low-entropy trees.

4.5 Experiments

To test the low-entropy mining methods discussed above in practice, implementations of the algorithms described in Subsection 4.4 were built using Perl language. Also, a depth-first search-based low-entropy set miner implemented by Jilles Vreeken at the Universiteit Utrecht was used. The results are reported in this section. Section 5.4 contains more experiments on low-entropy sets.

4.5.1 High frequency versus low entropy

We start by comparing statistics on a collection of frequent sets and a collection of low-entropy sets both mined from Mammals. For the comparison we generated

Algorithm 3 level-wise search for low-entropy trees.

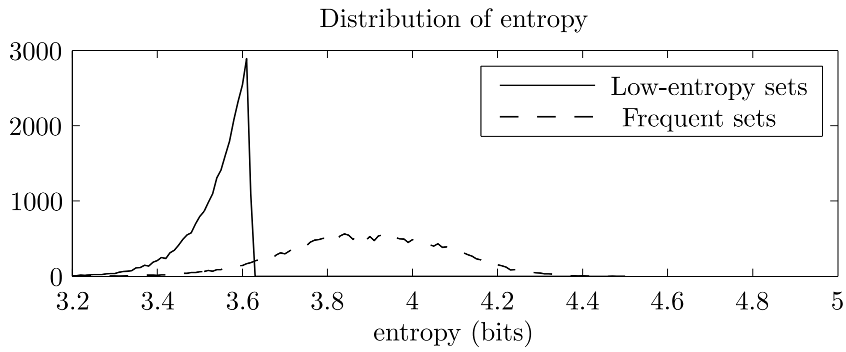
Input: Dataset $\mathcal{D}$ and entropy threshold ϵ.

Output: Collection of low-entropy trees $\mathcal{TP}_{LE}(\mathcal{D}, \epsilon)$.

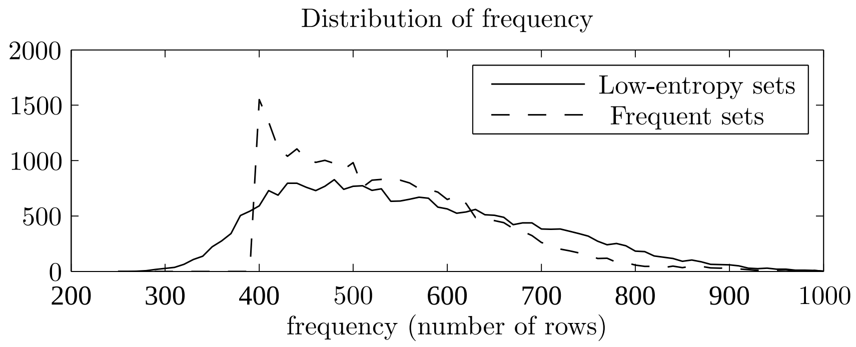
```
1: $i = 1$
2: candidate collection $C_i = \{T = (A, \emptyset)$— $A$ is an attribute$\}$
3: while $C_i$ is not empty do
4:    set of size $i$ patterns $\mathcal{P}_i = \{\}$
5:    //entropy checking
6:    for each $T = (X, E)$ in $C_i$ do
7:       if $H_{\mathcal{T}}(T) \leq \epsilon$ then
8:          add $T$ to $\mathcal{P}_i$
9:       end if
10:   end for
11:   // candidate generation
12:   for each $T = (X, E)$ in $\mathcal{P}_i$ do
13:      for each $(A, B)$ such that $A \in \mathcal{I} \setminus X$ and $B \in X$ do
14:         $T_{cand} = (X \cup A, E \cup (A, B))$
15:         if $\exists U = (Y, F) \in C_{i+1}$ such that $Y = X \cup A$ and $F \neq E \cup (A, B)$
            then
16:            if $H_{\mathcal{T}}(T_{cand}) < H_{\mathcal{T}}(U)$ then
17:               $C_{i+1} = (C_{i+1} \cup T_{cand}) \setminus U$
18:            end if
19:         else
20:            add $T_{cand}$ to $C_{i+1}$
21:         end if
22:      end for
23:   end for
24:   add $\mathcal{P}_i$ to $\mathcal{TP}_{LE}(\mathcal{D}, \epsilon)$
25:   $i = i + 1$
26: end while
27: return $\mathcal{TP}_{LE}(\mathcal{D}, \epsilon)$
```

a set of size-5 frequent sets using a frequency threshold of 400 rows. This resulted in a set of 26106 size-5 frequent itemsets. After this a matching number of size-5 low-entropy sets was generated by iteratively setting the entropy threshold to 3.623673. Only attributes with a frequency larger then 0.2 but smaller then 0.8 were included in the experiment.

Figure 4.4(a) shows the distribution of entropy for both pattern sets. We see that for frequent sets, the entropy is distributed more widely and towards higher values. This implies that most attribute sets found by the frequent itemsets mining algorithm are less structured in terms of the entire data. In contrast, the attribute sets of the low-entropy mining algorithm are more concentrated in their attribute combination throughout the data.



(a)



(b)

Figure 4.4: Comparison of the distribution of entropy and frequency for size-5 frequent sets ($\sigma \geq 0.1832$) and the same number of similar sized low-entropy sets ($\epsilon \leq 3.623673$) in Mammals. The frequency for a low-entropy set in Figure (b) is defined as the number of rows supported by its most frequent instantiation.

Figure 4.4(b) depicts the situation from the point of view of the frequency of a single instantiation. For frequent itemsets this is naturally the frequency of the instantiation of all attributes having the value 1. For low-entropy sets we define this as the number of rows supported by its most frequent instantiation. This time we see that the distribution of frequency is quite similar for both pattern sets. Hence, low-entropy sets are also able to find frequent attribute combinations similarly to frequent itemsets.

Furthermore, Figure 4.5 shows that a number of low-entropy sets have several instantiations that count as frequent according to the 400 rows frequency. This is a natural feature of low-entropy sets, but not to frequent itemsets that concentrate only on the count of one instantiation. In the next subsection we show an example pattern with several frequent instantiations.

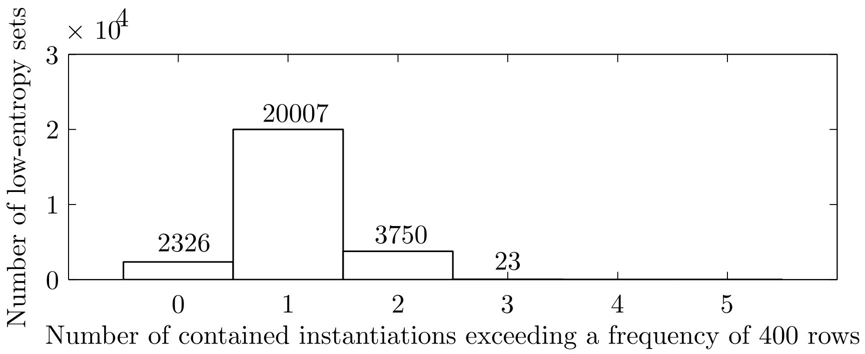


Figure 4.5: The distribution of patterns when classified according to the number of contained instantiations exceeding a frequency of 400 rows. The distribution is computed over a set of 26106 low-entropy sets of size 5 in Mammals.

4.5.2 Examples of low-entropy patterns

Mammals data

Next we look at some examples of the low-entropy sets and trees found in real data. Recall that using the 3.623673 bit threshold we found 23 LE-sets from the mammal data that had three instantiations with a frequency count of over 400 rows (see Figure 4.5).

Figure 4.6 shows the best pattern from these 23 sets, with a table of instantiation counts and expected counts according to the independence assumption. The attributes show an interesting pattern of negative correlation between the couple Blue Hare and Elk and the triplet Garden Dormouse, Wood Mouse and Greater White-toothed Shrew. Notice also that this occurrence behavior is very unexpected according to the independence assumption, and may hence reflect some significant structure in the data. A mutual exclusion pattern like this may derive from scarcity of resources such as food or other environmental factors, and could be hard to detect with regular frequent itemsets. Indeed, we notice that the count of all ones instantiation for the attributes in the pattern is zero.

On the right of Figure 4.6 we see the corresponding low-entropy tree for these attributes. The tree suggests that the presence and absence of Elk is the most central for determining the occurrence of the other species in the set.

For comparison, Figure 4.7 shows the two best (in terms of entropy) disjoint size-5 low-entropy trees in the same Mammals dataset. The tree in Figure 4.7(a) is an interesting subset of north European species with two large predators, Eurasian Lynx and Brown Bear, and three prey species, Wood Mouse, Blue Hare and Elk. Notice that the three last species are the same as in the LE-set in Figure 4.6. Here again the structure of the tree suggests that the presence and ab-

count	Blue Hare	Elk	Garden Dormouse	Wood Mouse	Greater White-toothed Shrew	expected count
554	0	0	0	1	0	391.9
412	1	1	0	0	0	30.2
404	0	0	1	1	1	57.5
144	0	0	0	1	1	141.4
135	0	0	1	1	0	159.3
117	1	0	0	1	0	193.2
112	1	1	0	1	0	82.1
103	0	0	0	0	0	144.1
79	0	1	0	1	0	166.5
24	1	1	1	0	0	12.3
24	1	0	1	1	0	78.6
21	1	0	1	1	1	28.4
18	0	1	0	0	0	61.2
10	0	0	1	0	0	58.6
7	1	0	0	0	0	71.1
5	0	1	1	1	0	67.7
5	0	0	0	0	1	52.0
4	0	0	1	0	1	21.2
3	1	0	1	0	0	28.9
1	0	1	1	0	0	24.9
1	1	0	0	1	1	69.8

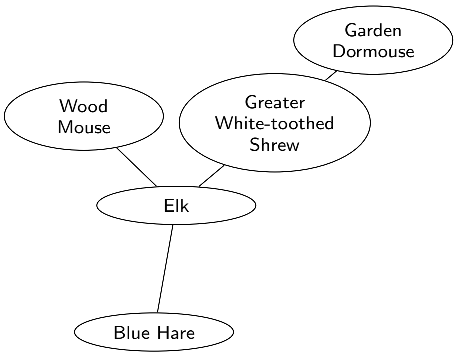


Figure 4.6: The table of instantiation counts for the best low-entropy set (according to entropy) that has at least three instantiations with frequency > 400. The tree on the right is the corresponding LE-tree for these attributes. The low-entropy set has an entropy score of 3.15, while the tree has an entropy score of 3.28. The patterns are from the data set MAMMALS.

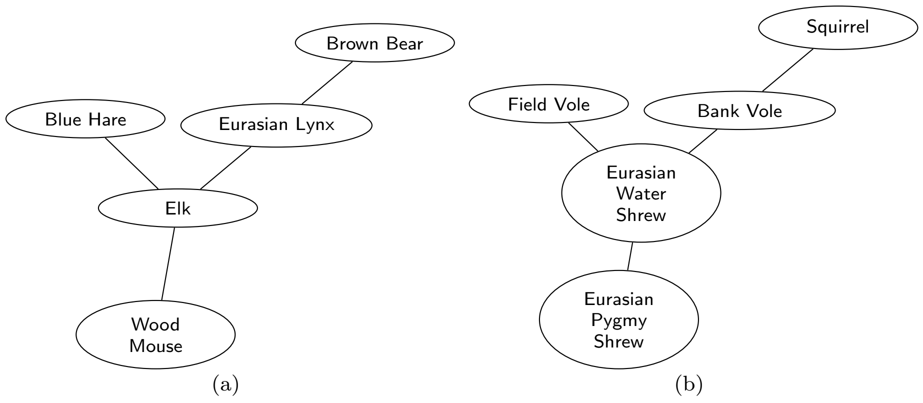


Figure 4.7: The top 2 disjoint size-5 low-entropy trees in Mammals. The entropies of the trees in Figure (a) and (b) are 3.16 and 3.31, respectively.

sence of Elk is the most central factor for determining the occurrence of the other species in the set. The other tree in Figure 4.7(b) shows a subset of small rodent or rodent-like mammals typically found in north European woodland areas.

MovieLens data

In Figure 4.8 we see the best size-5 low-entropy set in MovieLens after removal of attributes with a frequency < 0.2. The set features five thematically similar futuristic science fiction movies, Blade Runner (1982), Alien (1979), Aliens (1986), The Terminator (1984) and Terminator 2: Judgment Day (1991), with Aliens (1986) and Terminator 2: Judgment Day (1991) being sequels to the movies Alien (1979) and The Terminator (1984), respectively. From the instantiation counts we notice that this LE-set behaves more like a normal frequent itemset in comparison with the example in Figure 4.6: the two most frequent instantiations are the all-zeros and the all-ones cases. Similar behavior of instantiation counts was detected in most of the other LE-sets found in the MovieLens data.

On the right side of Figure 4.8 we see the best low-entropy tree for these attributes. The sequel movies are positioned consecutively with respect to the original movie in the tree. Furthermore, a time line is present from the early Blade Runner (1982) via the Aliens movies, and to the Terminator films. Note that temporal attribute ordering is also a familiar phenomenon in the results of frequent trees in Section 3.5.

Figure 4.9 shows the second and the third best disjoint size-5 low entropy trees in MovieLens. The tree in Figure 4.9(a) features a variety of drama movies from E.T the Extra-Terrestrial (1982) to Braveheart (1995). The tree in Figure 4.9(b) features action movies with Harrison Ford starring in all except the film Back to

count	Terminator 2: Judgment Day (1991)	Alien (1979)	Blade Runner (1982)	The Terminator (1984)	Aliens (1986)	expected count
541	0	0	0	0	0	251.5
67	1	1	1	1	1	0.6
29	0	0	1	0	0	76.1
27	1	0	0	0	0	79.3
26	1	0	0	1	0	23.7
19	0	1	0	0	1	22.3
18	0	1	0	0	0	77.9
18	0	1	1	0	0	23.6
17	0	0	0	1	0	75.2
17	0	0	0	0	1	72.1
17	1	1	0	1	1	2.1
14	1	1	1	0	1	2.1
14	0	1	1	1	1	2.0
13	1	0	1	1	0	7.2
13	0	1	1	0	1	6.8
10	1	1	1	1	0	2.2
10	1	0	0	1	1	6.8
9	1	1	0	1	0	7.3
9	1	0	1	1	1	2.1
⋮			⋮			⋮

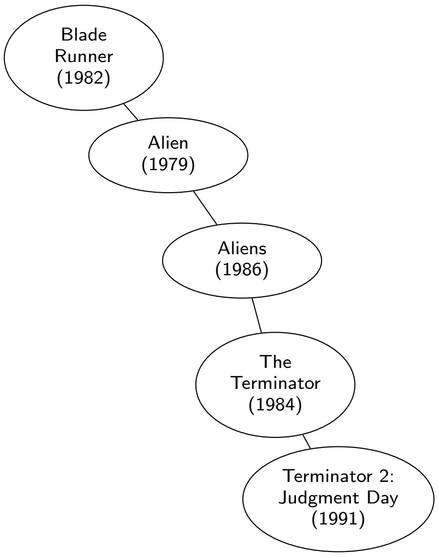


Figure 4.8: The table of instantiation counts for the best size-5 MovieLens LE-set in terms of entropy. The tree on the right is the corresponding low-entropy tree for these attributes. The low-entropy set has an entropy score of 2.89 while the tree has an entropy score of 3.04. Because of space constraints, 15 instantiations with a frequency less than 10 rows have been removed.

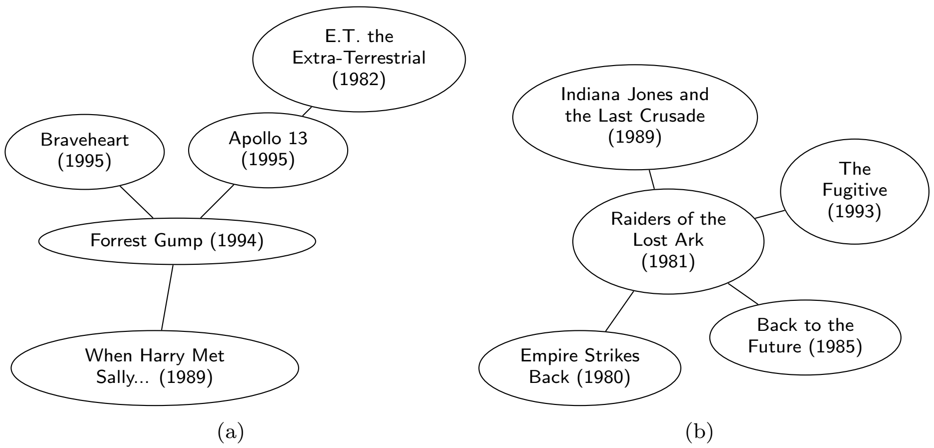

E.T. the Extra-Terrestrial (1982)
Braveheart (1995)
Apollo 13 (1995)
Forrest Gump (1994)
When Harry Met Sally... (1989)
(a)
Indiana Jones and the Last Crusade (1989)
The Fugitive (1993)
Raiders of the Lost Ark (1981)
Back to the Future (1985)
Empire Strikes Back (1980)
(b)

Figure 4.9: The top 2 and 3 disjoint size-5 low-entropy trees in MovieLens. The top 1 tree is shown in Figure 4.8. The entropies of the trees in Figure (a) and (b) are 3.23 and 3.40 respectively.

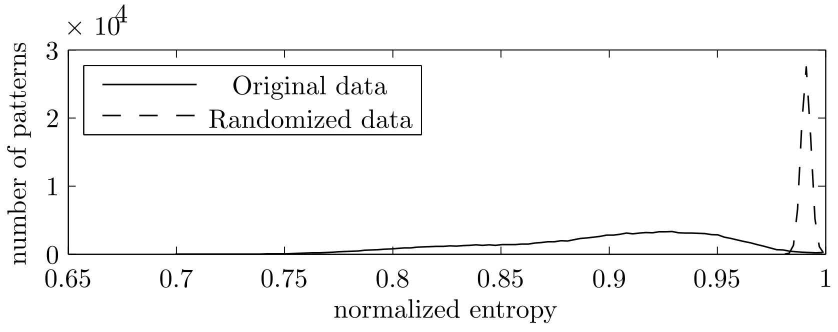
the Future (1985). For the latter tree the structure suggests that Raiders of the Lost Ark (1981) is a central attribute in relation to the preference of the movie database users with respect to the other films in Figure 4.9(b), something that could not have been seen from the corresponding set-type pattern.

4.5.3 Pattern validation using swap randomization

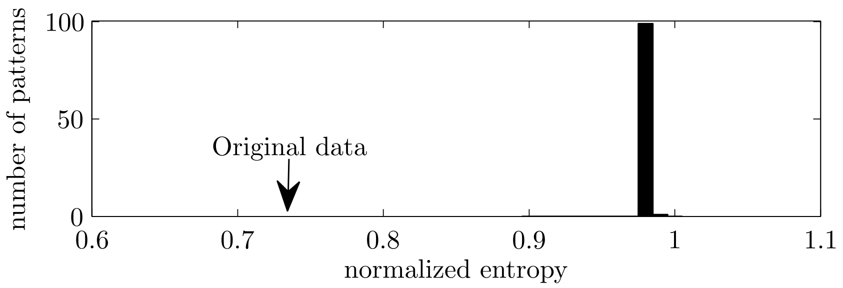
We used the Mammals dataset to perform a swap randomization test in order to evaluate the extent of the relevant structure captured by the low-entropy sets. The number of swaps in the procedure was equivalent to the number of rows multiplied by the number of columns in the original data. Using this procedure we produced a set of 100 swap randomized data instances. After this we generated a set of LE-sets using an entropy threshold of 3.623673 for the original data and each of the 100 swap randomized data instances. Only attributes with $0.2 \leq$ frequency ≤ 0.8 were included in the experiment.

To be able to evaluate the entire set of patterns resulting from the procedure, we used a normalized version of the score as a test statistic, in order to rule out the differences in the entropy score resulting from varying pattern sizes. For each LE-set we divided the entropy score with the sum of the entropies of the attributes in the set. Notice that the idea of the statistic is similar to that of the conflict ratio introduced in Chapter 3; the real score is compared with the expected score assuming independence of attributes in the data.

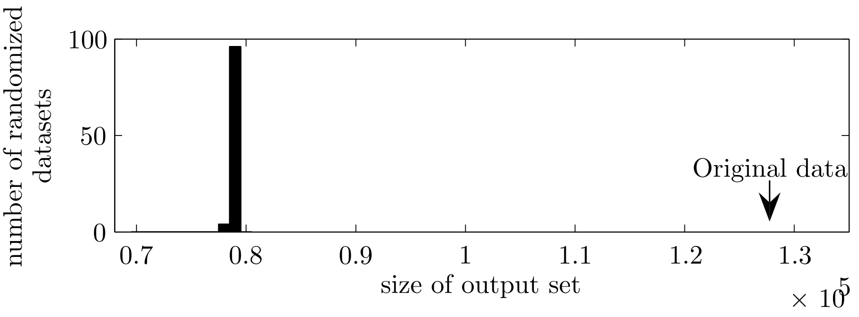
Figure 4.10 shows the results of the swap randomization tests. In Figure 4.10(a) we see the distribution of normalized entropy scores in the original Mammals. To compare this with the patterns found in swap randomized data we took a 1% random sample of the entire set of patterns generated from 100 swap randomized



(a)



(b)



(c)

Figure 4.10: Comparison between the original MAMMALS data and the corresponding swap randomized data. Figure (a) shows the empirical distribution of normalized entropy ratio of all patterns generated from the original data, in comparison with a 1% random sample on the set of patterns generated from 100 swap randomized dataset instances. Figure (b) shows the histogram of the 100th best normalized entropy ratio in the swap randomized data sets. Figure (c) shows the histogram for the corresponding sizes of the pattern output sets.

dataset instances. The distribution of normalized entropies resulting from these patterns is shown in the same figure with the dashed outline. A clear difference can be noticed in these two distributions.

To obtain an empirical p-value we compared two other pattern statistics in the original data with the corresponding statistics in the 100 swap randomized data instances: the 100th best normalized entropy score and the total size of the pattern set. In Figure 4.10(b) we see the histogram of the 100th best normalized entropy in the swap randomized datasets, and in Figure 4.10(c) the histogram for the size of the pattern output set. The values of the corresponding statistics in the original dataset are marked with an arrow in the histograms. In both cases an empirical p-value of 0.0004 is obtained.

4.5.4 General mining statistics

Finally we look at some general mining statistics in Table 4.2 for the three datasets: MovieLens, Mammals and Course Completion. We see that the run-times of the algorithms stay feasible even when the number of elements in the answer increasing as the entropy threshold ϵ is set higher.

4.6 Related work

The idea of low-entropy sets can be considered as an entropy-based alternative to frequent itemset mining [3]. On the other hand, in Chapter 3 we defined a tree type of pattern that is considered to be present in a row of data if the attributes of a rooted subtree are present simultaneously. Respectively, low-entropy trees can be considered as a general version of this type of pattern by allowing the distribution to be concentrated on combinations other than positive conjunctions.

Decision trees [36, Chapter 9.2] are similar to low-entropy trees in that low classification error implies low entropy. In [71, 70] Nijssen *et al.* proposed the idea of mining optimal decision trees from the itemset lattice. Traditionally, decision tree algorithms use a greedy search approach. One difference with respect to decision tree methods here is that we do not split the data into groups corresponding to different values of an attribute, but all data are considered in each node. In other words, $H(A|B)$ is used instead of e.g. $H(A|B = b)$.

The task of finding interesting itemsets has been addressed mainly in the context of frequent itemset mining. Morishita and Sese have presented a branch-and-bound method for finding association rules like $X \to Y$ where the itemsets X and Y are not required to have high support but high correlation [67]. They also count those data rows where X and Y appear completely, whereas our entropy-based method counts arbitrary combinations.

Strategies similar to that of [67] have also been employed by Zimmermann *et al.* [99]. In [9] they describe a sequence of graph pattern classes, and study the strategy of discovering structured patterns, say trees, using the subset of corresponding less complex patterns such as sequences or sets.

Table 4.2: Low-entropy mining statistics for the datasets MovieLens, Mammals and Course Completion. The run times between the set and the tree pattern results are not directly comparable: the sets implementation uses C++ while the tree implementation is done with Perl.

Dataset	ϵ	Pattern Type	$\lvert\mathcal{P}\rvert$	$\lvert X\rvert$	time/sec.	Gen Cands
MovieLens	2.0	Set	1257	3	<1	19652
MovieLens	2.0	Tree	1241	3	2	20332
MovieLens	2.75	Set	22967	4	1	230875
MovieLens	2.75	Tree	20690	4	34	267217
MovieLens	3.5	Set	348902	6	18	2303166
MovieLens	3.5	Tree	259037	6	488	2888023
Mammals	2.0	Set	821	3	<1	10700
Mammals	2.0	Tree	821	3	1	10737
Mammals	2.75	Set	10396	4	1	90548
Mammals	2.75	Tree	10209	4	14	102918
Mammals	3.8	Set	250628	6	17	989597
Mammals	3.8	Tree	148904	6	246	1540910
Course C.	1.0	Set	1323	3	<1	11165
Course C.	1.0	Tree	1314	3	6	77499
Course C.	1.5	Set	34936	5	2	203437
Course C.	1.5	Tree	33722	5	149	1675337
Course C.	1.8	Set	220087	6	12	1113800
Course C.	1.8	Tree	205008	6	894	9210538

Closer to our approach are Knobbe and Ho's maximally informative k-itemsets, i.e., itemsets with as high an entropy as possible [54]. Here we study itemsets with low entropy. Another difference is that Knobbe and Ho restrict their itemsets to a fixed number, k, of elements.

In real-valued data, the task of finding interesting subsets has been addressed in research areas such as subspace clustering [1, 73] and projection pursuit [24, 44]. Another method related to our task is the problem of learning the structure of a Bayesian network. This problem is computationally challenging, and the main methods are probabilistic [18, 37]; the best current exact algorithms are of the order $O(n2^n)$ [56, 84]. The key difference between Bayesian network structure learning and our approach is that we seek interesting subsets of the data, not complete models; also, we investigate only fully connected or tree-structured networks and not arbitrary graphs, and we do not try to incorporate any prior knowledge into the patterns.

4.7 Conclusions

We have considered the problem of finding low-entropy sets and trees from binary data. The approach we have chosen is a natural generalization of the discovery of frequent sets: low-entropy sets are not restricted to the all-one value combination and can locate subsets that have a number of different dominant value combinations instead of just one. The experiments show that the methods are able to discover interesting sets and trees and that they can also be used on large datasets. Our approach searches for local structure: small subsets of variables for which the data can be modeled well, in the sense of having low entropy or being amenable to description by a tree.

Chapter 5

Pattern selection for low-entropy sets

5.1 Introduction

In the previous two chapters we concentrated on the task of enumerating a complete set of patterns that satisfy certain conditions or constraints. For instance, with low-entropy sets we were interested in the entire set of patterns that yield an entropy value that is below a certain predefined threshold ϵ. In a way this is a very natural goal. The user defines the pattern set that he or she is interested in and makes a query that returns all patterns.

However, receiving all possible patterns may be impractical as the number of patterns returned may be prohibitively large. Indeed, in many cases the pattern set found may be larger than the entire number of observations in the data. Hence, giving exactly what the user wants may not be what he really needs. Instead, perhaps more beneficial for the user would be to give a small subset of the most essential patterns with some desired property (like low entropy or high frequency). This problem is often referred to as the *pattern selection problem*, and has recently attracted a significant amount of research [65, 83, 55, 8, 93].

In 2006 Siebes *et al.* [83] introduced an interesting approach to the pattern selection problem by promoting the use of the MDL (minimum description length) principle. MDL is a general model selection criterion introduced originally by Rissanen [79] that states that the best model for a given set of data is the one that leads to the largest compression of the data. Following from this, the argument of Siebes *et al.* was that the best collection of frequent itemsets should be the one that requires the fewest bits to describe all the data. Furthermore, a row of data can be described simply by telling which itemsets together (i.e. their union) form the data row.

The original work by Siebes *et al.* was for selecting frequent itemsets. However, low-entropy sets offer an interesting possibility to study this idea further.

Indeed, in terms of information theory, entropy is the average number of bits needed to encode the values of the attributes of the set X on a single row of data. Hence, for low-entropy sets this quantity is naturally low and therefore they make good candidates for minimizing overall data description length, and are hence well suited for the pattern selection problem using MDL.

There is also another property that gives low-entropy sets an advantage. This is the fact that low-entropy sets may capture multiple instantiations at once for the same attribute set, while a frequent set has only one instantiation that it can use. Take for instance the low-entropy set depicted in Figure 4.6. If the same instantiation occurrence had to be described using regular itemsets, many separate sets would be required, instead of the single low-entropy set required in this case. Low-entropy sets can be seen as providing a *two-level abstraction.* The higher, pattern-level abstraction, provides those attributes that have dependencies between each other. The lower, instantiation-level abstraction, specifies in detail the attribute value combinations occurring in the data. Hence, at the pattern level, less is needed to describe more complex phenomena in the data.

Based on these ideas, this chapter discusses the pattern selection problem using low-entropy sets and the MDL principle. We study describing each row in the data with a subset of low-entropy sets using the maximum likelihood principle. Furthermore, two algorithms for data row encoding are provided. Finally, the overall pattern selection problem is solved using a heuristic algorithm. Experiments demonstrate that the end results yield an easily interpretable, small collection of low-entropy sets, that in most cases are an order of magnitude smaller than with a similar method using frequent itemsets.

5.2 Problem definition

5.2.1 The MDL model for low-entropy sets

The MDL approach can be summarized by the following statement: *the best model is the one that yields the best compression for the data.* To put it slightly more formally, given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ is the one that minimizes

$$L(\mathcal{D}|M) + L(M),$$

in which

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

In our case a model M defines a collection S of low-entropy sets. Respectively, a description of the data is obtained so that for each row $t \in \mathcal{D}$, the model M tells which LE-sets in S and which corresponding instantiations are used to obtain the values of $t(A)$ for each attribute $A \in \mathcal{I}$.

Example Given a row t we describe it by a sequence of LE-sets and instantiations. As a simple example, consider the attributes $\{A, B, C, D\}$ and a data row $t = (1, 0, 0, 1)$, with attributes A and D having the value 1 and B and C the value 0. This row can be described by the LE-sets $\{A, B\}$ and $\{C, D\}$ and instantiations $(1, 0)$ and $(0, 1)$ respectively.

The compression is done using two *code tables*. The first code table contains the collection of the LE-sets S and the codes used to encode them. The second code table contains the corresponding instantiations and their respective codes. The data are then described using the codes in the code tables.

Example Let the codes associated with $\{A, B\}$ and $\{C, D\}$ be c_1 and c_2, respectively. Furthermore, let l_1 be the code associated with $(1, 0)$ and l_2 the one with $(0, 1)$. The row $\{A, D\}$ is encoded by the pair of codes c_1c_2 and l_1l_2.

In the following we give a more formal definition for a code table.

Definition 5.1 *Consider a dataset $\mathcal{D}$, the entropy threshold ϵ, and a subset $S \subseteq \mathcal{P}_{LE}(\mathcal{D}, \epsilon)$. We define the* low-entropy set code table *as the set of pairs*

$$CT_{LE} = \{(X, c) \mid X \in S \cup \mathcal{I}\},$$

where X is a low-entropy set and c the respective code for X. Correspondingly, we define the instantiation code table *as the set of pairs*

$$CT_I = \{(i, l) \mid i \in \bigcup_{Y \in S \cup \mathcal{I}} \Omega_Y\},$$

where i is a low-entropy set instantiation, l is its respective code, and Ω_Y the set of all instantiations for the low-entropy set Y. We write $X \in CT_{LE}$ (and similarly for CT_I), to indicate that for a low-entropy set X, there exists an entry $(X, c) \in CT_{LE}$

Notice that CT_{LE} is required to contain at least the singleton attribute sets, such that all possible rows can be encoded using at least the single-item low-entropy sets. Similarly, if the largest LE-set in S contains k attributes, CT_I is defined to contain entries for all possible instantiations from one-element LE-sets up to those for k-element LE-sets.

The pair of code tables results in a pair of encodings for the dataset $\mathcal{D}$. The first, $\mathcal{D}_{LE}$, contains the codes from CT_{LE}, the second, $\mathcal{D}_I$, contains the codes from CT_I. We denote the lengths of the two encodings of the data by $L(\mathcal{D}_{LE} \mid CT_{LE})$ and $L(\mathcal{D}_I \mid CT_I)$. The number of bits required to encode CT_{LE} and CT_I is denoted by $L(CT_{LE})$ and $L(CT_I)$, respectively.

Now, given data $\mathcal{D}$, and a set of low-entropy sets $\mathcal{P}_{LE}(\mathcal{D}, \epsilon)$ for some entropy threshold ϵ, our objective will be to find the code tables CT_{LE} and CT_{LE} that minimize $L(\mathcal{D}|CT) + L(CT)$, where

$$\begin{aligned} L(\mathcal{D}|CT) &= L(\mathcal{D}_{LE} \mid CT_{LE}) + L(\mathcal{D}_I \mid CT_I), \\ L(CT) &= L(CT_{LE}) + L(CT_I). \end{aligned}$$

In the following Subsections 5.2.2 and 5.2.3, we discuss $L(\mathcal{D} \mid CT)$ and $L(CT)$ in detail. The formal overall problem statement is given in Subsection 5.2.4.

5.2.2 $L(\mathcal{D} \mid CT)$: size of the encoded dataset

The size of the encoded dataset will be the sum of lengths of the codes used to encode each data row. To determine the appropriate codes, we need to know how often an LE-set and its instantiations are used in the encoding. That is, we have to define which low-entropy sets are used to cover a row t and which instantiations of those elements are used. For this purpose we use a *cover function* that provides a set of non-overlapping LE-sets such that they describe all attributes $\mathcal{I}$ of a row t of the data $\mathcal{D}$. More formally:

Definition 5.2 *Given a low-entropy set code table CT_{LE} and a row $t \in \mathcal{D}$, the collection C of low-entropy sets is a* cover *of t, iff $\bigcup_{X\in C} = \mathcal{I}$, $\bigcap_{X\in C} = \emptyset$, and for each $X \in C$ we have $X \in CT_{LE}$. A* cover function *is a function*

$$cover : CT_{LE} \times t \rightarrow C,$$

where C is a cover of t.

Since the elements in a cover C are non-overlapping, $cover(CT_{LE}, t)$ only needs to return a set of LE-sets from CT_{LE}. The associated instantiations can easily be reconstructed by using the values of row t. We omit CT_{LE} and write $cover(t)$ when CT_{LE} is clear from the context.

The number of times a CT_{LE} element X is used in the cover of each row $t \in \mathcal{D}$ is called its *cover frequency* of X, denoted by $f_c(\mathcal{D}, X)$. The cover frequency of an instantiation $i \in CT_I$ is defined similarly:

$$\begin{aligned} f_c(\mathcal{D}, X) &= |\{t \in \mathcal{D} |\ X \in cover(t)\}|, \\ f_c(\mathcal{D}, i) &= |\{t \in \mathcal{D} |\ X \in cover(t) \wedge i = \pi_X(t)\}|. \end{aligned}$$

We denote $f_c(X)$ and $f_c(i)$ when $\mathcal{D}$ is clear from the context.

The probability that X, or respectively i, is used in the cover of a randomly selected row t is thus

$$\begin{aligned} \Pr(X|\mathcal{D}) &= \frac{f_c(X)}{\sum_{Y\in CT_{LE}} f_c(Y)}, \\ \Pr(i|\mathcal{D}) &= \frac{f_c(i)}{\sum_{j\in CT_I} f_c(j)}. \end{aligned}$$

To compress the dataset optimally, we use a Shannon code [30] for both code tables. The more often a set or an instantiation is used, the shorter its associated code. The lengths[1] (in bits) of the codes c and l for the code table elements $(X,c) \in CT_{LE}$ and $(i,l) \in CT_I$, are

$$\begin{aligned} L(c) &= -\log(\Pr(X|\mathcal{D})), \\ L(l) &= -\log(\Pr(i|\mathcal{D})). \end{aligned}$$

Finally, the sizes of the encoded datasets $\mathcal{D}_{LE}$ and $\mathcal{D}_I$ are

$$\begin{aligned} L(\mathcal{D}_{LE} \mid CT_{LE}) &= \sum_{(X,c)\in CT_{LE}} f_c(X)L(c), \\ L(\mathcal{D}_I \mid CT_I) &= \sum_{(i,l)\in CT_I} f_c(i)L(l). \end{aligned}$$

The encoded size of the full dataset is then

$$L(\mathcal{D} \mid CT) = L(\mathcal{D}_{LE} \mid CT_{LE}) + L(\mathcal{D}_I \mid CT_I). \tag{5.1}$$

5.2.3 $L(CT)$: size of the code table

As stated earlier the two code tables, CT_{LE} and CT_I, comprise pairs (X,c) and (i,l), respectively. For the codes we already know their lengths, that is, $L(c)$ and $L(l)$ as defined above. However, to compute the full size of the code tables, we need to include the encoding lengths of the low-entropy sets X and the instantiations i as well. In other words, we must define $L(X)$ and $L(i)$.

For CT_{LE}, we encode each LE-set X with the encoding defined by the *standard code table*

$$ST = \{(A,c)|A \in \mathcal{I}\},$$

where ST contains only the singleton attribute sets $A \in \mathcal{I}$, and the codes c resulting from encoding the data only with them. We have

$$L(X) = \sum_{(A,c)\in ST: A\in X} L(c)$$

The size of CT_{LE} is then

$$L(CT_{LE}) = \sum_{(X,c)\in CT_{LE}: f_c\neq 0} L(X) + L(c).$$

For CT_I, we simply use the bit-representation of the instantiations. That is, the instantiation $(0,1)$ is represented by 01. We denote the bit-representation of

[1]Note that we are only interested in the lengths of the codes c and l, not the actual codes themselves.

$i \in CT_I$ by $bit(i)$, and the number of bits in $bit(i)$ by $L(bit(i))$. Hence, the size of CT_I is

$$L(CT_I) = \sum_{(i,l) \in CT_I : f_c \neq 0} L(bit(i)) + L(l).$$

We have as the total size for the code tables,

$$L(CT) = L(CT_{LE}) + L(CT_I).$$

5.2.4 The problem statement

Now that we have given the details of our MDL model class, we define the overall computational problem as follows.

Problem 5.1 *Given data $\mathcal{D}$, and a set of low-entropy sets $\mathcal{P}_{LE}(\mathcal{D}, \epsilon)$ for some entropy threshold ϵ, find the code table CT_{LE} and the cover function $cover(CT_{LE}, t)$ minimizing the total encoded size*

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

Note that given CT_{LE} and the cover function $cover(CT_{LE}, t)$, we can determine CT_I.

5.3 Algorithms

The search space of the problem consists of all possible code tables CT_{LE}, that is, all possible subsets of $\mathcal{P}_{LE}$ that contain at least the singleton sets $\mathcal{I}$. So, there are

$$\sum_{k=0}^{|\mathcal{P}_{LE}|-|\mathcal{I}|} \binom{|\mathcal{P}_{LE}| - |\mathcal{I}|}{k}$$

possible code tables. In order to determine which one minimizes the total encoded size, the naive approach is to consider these using every possible cover function. This means constructing the row covers using every possible cover order for each row. Since there are $m!$ possible orders for a set of size m, the total size of the search space is

$$\sum_{k=0}^{|\mathcal{P}_{LE}|-|\mathcal{I}|} \left(\binom{|\mathcal{P}_{LE}| - |\mathcal{I}|}{k} \times (k + |\mathcal{I}|)! \times |\mathcal{D}| \right).$$

In short, the search space is very large. To make matters worse, there is no known structure that allows us to prune it. Hence, we must resort to a heuristic overall solution.

5.3.1 Solution outline

Our approach to finding the best possible code table can be divided into two main elements:

- *row encoding phase*, where a good compression for each row (the cover function) is found using the patterns in the code table.
- *code table search strategy*, which is the way in which the search space of all possible pattern subsets is traversed to find a good code table CT_{LE}.

The method follows the general framework of [83]. However, we apply different technical solutions within the different parts of the approach. We will discuss row encoding in Subsection 5.3.2 and the search strategy in 5.3.3.

5.3.2 Row encoding phase

As discussed in Section 5.2, when encoding the dataset the task is to compress the data as well as possible using the low-entropy sets in the existing code table. This is done by covering each row t with a set of patterns from the code table using some cover function $cover(t)$. Until now we have not explicitly defined what a cover function might look like.

An LE-set always has an instantiation that can be used to describe a row, so any disjoint subset of LE-sets covering $\mathcal{I}$ can be used. However, we know that we need a function that yields covers that are favorable for the overall task at hand.

Our strategy is to take advantage of the statistical nature of low-entropy sets and use the maximum likelihood (ML) principle. The idea is to define the cover function such that each cover $C = cover(t)$ for each row t maximizes the conditional probability $p(t|C)$ (likelihood) of the row. Recall that we denote with $\pi_X(t)$ the instantiation of the LE-set X that fits row t. We define the likelihood of row t as follows:

Definition 5.3 *Let t be a row and C a cover of t.*

$$\mathcal{L}(t, C) = \sum_{X \in C} \log p(\pi_X(t)) \tag{5.2}$$

is the loglikelihood *of t given C.*

Using the notion of likelihood we define $cover(t)$ as the function

$$cover(t) = C^* = \arg\max_C \mathcal{L}(t, C). \tag{5.3}$$

That is, for each row t the function $cover(t)$ yields a cover C^* such that the likelihood function $\mathcal{L}$ is maximized with respect to all possible covers C. We call C^* the *optimal cover* of t.

Maximum likelihood is a widely used and well principled way of fitting the best model to the data. Moreover, Equation (5.2) has an intuitive connection to the total encoded length of the data.

In more detail, the connection is as follows. Given row t and its cover C, let's assume that for all the rest of the rows $u \in \mathcal{D} \setminus t$, each LE-set $X \in C$ is used to cover u only whenever $\pi_X(t)$ fits u[2]. This implies that for each $X \in C$, we have a cover frequency of

$$f_c(X) = |\mathcal{D}| \cdot p(\pi_X(t)),$$

where $|\mathcal{D}|$ is the number of rows in the data, and $p(\pi_X(t))$ the relative frequency of the rows for which $\pi_X(t)$ fits.

From this it follows that for the code c, such that $(X, c) \in CT_{LE}$, the length $L(c)$ will be strictly proportional to the negative loglikelihood of X on row t. That is,

$$L(c) = -\log \frac{f_c(X)}{\sum_{Y \in CT_{LE}} f_c(Y)} \propto -\log \frac{f_c(X)}{|\mathcal{D}|} = -\log p(\pi_X(t)), \tag{5.4}$$

where $\sum_{Y \in CT_{LE}} f_c(Y)$ and $|\mathcal{D}|$ can be considered as constants. It follows that the encoding length $L(t)$ of row t will be

$$L(t) \propto -\sum_{X \in C} \log p(\pi_X(t)),$$

that is, the negation of Equation (5.2).

Finding the optimal cover

Finding the optimal cover for a row using patterns of size 1 and 2 is equivalent to the 2-dimensional matching problem and hence solvable in polynomial time. The general case where the code table includes patterns of size 3 or larger is as hard as or harder than the NP-complete 3-dimensional matching problem [26]. Therefore, the cover function defined in Equation (5.3) is not known to be computable in the general case for large datasets. Fortunately, many covering problems can be well approximated by using greedy heuristics.

In this subsection, we first study covering a row optimally according to the maximum likelihood principle, with an exhaustive search strategy using pruning. In the next subsection, we discuss a greedy heuristic that can be applied for larger datasets.

To compute the optimal cover for row t, consider assigning a weight $w(t, X)$ to each LE-set $X \in CT_{LE}$, such that

$$w(t, X) = \log\big(p(\pi_X(t))\big)/|X|, \tag{5.5}$$

[2]It is clear that these assumptions will not strictly hold in every case. However, patterns with a few high frequency instantiations are likely to behave approximately like this. Such patterns are also more likely to be chosen for the cover by the maximum likelihood principle.

where $w(t, X)$ is equal to the per-attribute addition in the likelihood that X would give if it were to be added to the cover of t.

Now, start with a code table that is ordered according to $w(t, X)$. Given this ordered code table, we need to enumerate all possible covers in a depth-first manner. We start from the set X with the largest weight $w(t, X)$ and greedily continue to add non-overlapping sets to the cover in decreasing order, backtracking the search at each time when reaching a full cover.

By taking this order into account, we can cut the search space down considerably using the following proposition.

Proposition 5.1 *Consider covering row t with disjoint attribute sets in strictly decreasing order according to the weight function w. Now, when at the ith attribute set X_i, already having covered attributes Y with cover C_Y, we know that the final resulting cover will have a likelihood of at most*

$$\mathcal{L}(t, C) \leq k \cdot w(t, X_i) + \mathcal{L}(t, C_Y), \tag{5.6}$$

where k is the number of uncovered attributes.

The proposition follows straightforwardly from the fact that, if X_i covers all the previously uncovered attributes without overlapping Y, the likelihood of the resulting covering will be exactly the right-hand side of inequality 5.6. Otherwise, we will have to include some other set, which by the ordering on w will provide an equal or smaller addition in likelihood per attribute. Hence, this will result in a smaller overall likelihood for the row. Thus, Proposition 5.1 defines an upper bound that we can compare with the best solution found so far; and thus to decide whether it makes sense to continue building the current cover or to start backtracking.

Written in pseudo-code, this optimal covering approach is depicted in Algorithm 4. However, as the optimal covering method considers a prohibitively large search space, it only makes sense to apply it to moderately sized datasets of up to about 25 attributes. To allow for more practical applications, we present a fast, heuristic alternative that follows very naturally from the minimum encoding length/maximum likelihood principle.

Approximating the optimal cover

Recall that our goal is to cover rows using LE-sets that provide as high a gain as possible in the overall likelihood. The initial order used by the optimal algorithm provides us with an approximation, as it orders the elements on the gain in likelihood per attribute. If we use this order in a greedy fashion (without overlap), the resulting cover is the same as the first full cover the optimal cover strategy considers. We present, as Algorithm 5, the translation of this simple scheme into pseudo-code.

Algorithm 4 Optimal row covering algorithm

Input: Row t for attributes $\mathcal{I}$ and code table CT_{LE}.

Output: Optimal cover C^* as defined in Equation (5.3).

```
1:  return Optimal($\mathcal{I}$, $CT_{LE}$, $\emptyset$, $\emptyset$)
2:
3:  Optimal($\mathcal{I}$, $CT_{LE}$, $C_Y$, $C^*$) :
4:  if $|C_Y| = |\mathcal{I}|$ then
5:      return $C_Y$
6:  end if
7:  $\Delta = \emptyset$
8:  $k = |\mathcal{I}| - |C_Y|$
9:  for $X \in CT_{LE}$ in decreasing order of $w(t, X)$ do
10:     $\Delta = X \cup \Delta$
11:     if $X \cap C_Y = \emptyset$ then
12:         if $k \cdot w(X) + \mathcal{L}(C_Y) > \mathcal{L}(C^*)$ then
13:             $C =$ Optimal($CT_{LE} \setminus \Delta$, $X \cup C_Y$, $C^*$)
14:             $C^* = \arg\max_{\{C^*, C\}}(\mathcal{L}(C^*), \mathcal{L}(C))$
15:         else
16:             return $C^*$
17:         end if
18:     end if
19: end for
20: return $C^*$
```

5.3.3 Code table search strategy

In the previous subsection we discussed data encoding assuming a fixed code table CT_{LE}. In this subsection we assume that CT_{LE} is unknown, and turn our attention to the problem of finding the best instance of it, given $\mathcal{P}_{LE}(\mathcal{D}, \epsilon)$. Our solution is to use the following simple greedy search strategy:

- Start with the (standard) code table consisting only of singleton attribute sets.
- Add each *candidate* $X \in \mathcal{P}_{LE}(\mathcal{D}, \epsilon)$ to the code table one by one. If the resulting codes, after adding X to the code table, lead to a better compression, keep X. Otherwise, discard it.

By its iterative nature, the success of this strategy largely depends on the order in which the patterns are considered.

Ordering the candidate sets

Using the above strategy, the optimal compression can be best approximated by trying all possible orders. However, as the number of possible orders of a set

Algorithm 5 The greedy row covering algorithm

Input: Row t for attributes $\mathcal{I}$ and code table CT_{LE}.

Output: A greedy approximation C for the optimal cover C^* as defined in Equation (5.3).

```
1: cover $C = \emptyset$
2: for $X \in CT_{LE}$ in decreasing order of $w(t, X)$ do
3:   if $X \cap C = \emptyset$ then
4:     $C = X \cup C$
5:     if $|C| = |I|$ then
6:       return $C$
7:     end if
8:   end if
9: end for
```

Algorithm 6 The low-entropy set selection algorithm LESS

Input: attribute set $\mathcal{I}$, the candidate LE-sets $\mathcal{P}_{LE}(\mathcal{D}, \epsilon)$, and a dataset $\mathcal{D}$.

Output: Code table CT_{LE}

```
1: //start with the standard code table
2: $CT_{LE} = ST = \{(A, c) | A \in \mathcal{I}\}$
3: // try each candidate in the code table
4: for $X \in \mathcal{P}(\mathcal{D}, \epsilon)$ in increasing order of $H(X)/|X|$ do
5:   if $L(\mathcal{D}, CT_{LE} \cup (X, c)) < L(\mathcal{D}, CT_{LE})$ then
6:     $CT_{LE} = CT_{LE} \cup (X, c)$
7:   end if
8: end for
9: return $CT_{LE}$
```

of size n equals $n!$, this clearly is infeasible for all but the smallest of pattern collections. Hence, we heuristically reduce the search space of the problem by introducing an order on $\mathcal{P}_{LE}(\mathcal{D}, \epsilon)$. That is, we order the candidates such that the best sets have a good chance of being used – those with high likelihood instantiations – are at the top of the list. At the candidate set level, this means preferring sets that have a low entropy per attribute, or $H(X)/|X|$.

5.3.4 The low-entropy set selection algorithm

Now the main ingredients for a low-entropy set-based compression algorithm are in place, and we can assemble these into the Low-Entropy Set Selection (LESS for short) algorithm. Algorithm 6 presents the pseudo–code version of the method.

As input, the algorithm requires the attribute set $\mathcal{I}$, the candidate set of low-entropy sets $\mathcal{P}_{LE}(\mathcal{D}, \epsilon)$, and a dataset $\mathcal{D}$. As a first step the routine initializes the code table CT_{LE} with the set of singleton patterns. Then, iteratively low-entropy sets are added to the code table one by one. Each time, for each row $t \in \mathcal{D}$ Algorithm 4 (or Algorithm 5) is called, and a new pattern coverage and the total compressed size of the dataset are calculated. If this addition improves the compression attained, the set is kept, otherwise it is permanently discarded.

Code table pruning

In the course of this iterative process it is quite possible that adding a pattern to the code table will suddenly greatly diminish the usage of other patterns in the code table thereby increasing their code lengths and possibly hindering overall compression. We therefore introduce a pruning variant of our method. Once a candidate of $X \in \mathcal{P}_{LE}(\mathcal{D}, \epsilon)$ is accepted into the code table, we reconsider all other elements $Y \in CT_{LE} : X \neq Y$ iteratively by temporarily removing them and calculating the compressed size. Using the MDL-principle, we then go for the best compression, permanently removing those elements that no longer help the compression.

5.4 Experiments

To test the low-entropy selection algorithm in practice a C++ implementation was constructed. The results of the experiments are reported in this section.

5.4.1 Size of pattern set reduction

Table 5.1 presents the quantified results of running LESS using the greedy covering Algorithm 5. The table shows the large reduction in the number of low-entropy sets that the algorithm attains. Even for relatively low thresholds of ϵ, up to 5 orders of magnitude fewer sets are selected, while attaining substantial compression of the databases.

We also see that enabling pruning has a strong effect on the number of sets selected: roughly an order of magnitude. Inspection of the code tables (see the example code table in Section 5.4.4) shows that the two strategies provide slightly different views on the data. Without pruning, the likelihood maximization process selects more specific sets. Consequently, from the selected sets typically only a few (one or two) very characteristic instantiations find major use. With pruning enabled the process is forced to select more general patterns. This effect is clearly illustrated by the much smaller number of sets returned, of which now multiple instantiations are often used. The better compression scores show that pruning results in better data descriptions.

5.4.2 MDL with LE-sets versus frequent sets

Table 5.1 includes a comparison of the reduction results between the LESS method and the frequent itemsets-based MDL approach KRIMP [83]. Note that

Table 5.1: Amount of pattern reduction obtained by LESS, using greedy covering Algorithm 5 and a variety of datasets, with a comparison with the frequent itemset-based method KRIMP. %compr.=compression achieved with the MDL model

Dataset			Pruning	LESS				KRIMP		
Name	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{D}\rvert$	*Enabled*	ϵ	$\lvert\mathcal{P}_{LE}(\mathcal{D},\epsilon)\rvert$	$\lvert CT_{LE}\rvert$	*% compr.*	$\sigma\lvert\mathcal{D}\rvert$	$\lvert\mathcal{P}(\mathcal{D},\sigma)\rvert$	$\lvert CT\rvert$
MAMMALS	40	2183	No	3.8	250628	**488**	65.0	200	11603531	536
			Yes			**30**	57.0			254
MOVIELENS	49	943	No	3.5	348902	**919**	65.9	40	355259	297
			Yes			**341**	59.3			182
COURSE C.	83	2405	No	2.8	445709	**918**	70.6	100	10896464	551
			Yes			**28**	61.7			285
ADULT	97	48842	No	2.9	143766	**153**	79.3	1	58461763	1941
			Yes			**10**	77.9			1303
HEART	50	303	No	3.3	414589	**115**	87.2	1	1922983	108
			Yes			**49**	85.8			79
LETTER R.	102	20000	No	3.3	368889	**838**	76.1	50	5219327	3395
			Yes			**21**	70.2			1259
MUSHROOM	119	8124	No	2.8	437239	**241**	73.5	1	5574930437	689
			Yes			**10**	69.3			424
PEN DIGITS	86	10992	No	2.5	71994	**160**	60.5	50	2414945	2667
			Yes			**28**	55.9			1091

in most cases for KRIMP we use a significantly larger candidate set $\mathcal{P}(\mathcal{D}, \sigma)$, compared with the candidate set $\mathcal{P}_{LE}(\mathcal{D}, \epsilon)$ used with LESS. This is an advantage to KRIMP, since there is a larger candidate set to choose from, for the final result. On the other hand, low-entropy sets have multiple instantiation to use in the compression, which is not the case for frequent sets.

We see that using LE-sets instead of frequent sets pays off, however, in the overall number of patterns that are used to describe the data. With pruning enabled, in 7 out of 8 cases the number of patterns selected is one or two orders of magnitude smaller with the LE-set-based LESS than with the frequent itemset-based KRIMP. For LESS, pruning disabled, the LE-sets selected have typically only one or two very characteristic instantiations finding major use. This results in a more frequent itemset type of behavior. Still, with pruning disabled, in 5 cases out of 8 the result set is smaller with LESS than with KRIMP.

5.4.3 Optimal and greedy covering

To compare the performance between the optimal and the greedy covering strategies (Algorithms 4 and 5), we use a reduced version of the Mammals dataset, Mammals$_{20}$ containing only the 20 most varying attributes of the full Mammals dataset. With this data we mined low-entropy sets using a maximum entropy threshold of 3.33 bits, resulting in 2321 low-entropy sets.

For each row we computed a cover using both the greedy covering Algorithm 5 and the optimal covering Algorithm 4, as well as a baseline cover computed only with singleton sets. The full collection of 2321 LE-sets was used as the fixed code table for the greedy and the optimal approaches.

The results for each single row are presented in Figure 5.1. First of all, we see that using sets pays off for both the optimal and the greedy methods with an increase in loglikelihood for each row. Moreover, we see that the optimal row cover does indeed result in the highest loglikelihood scores. However, the much faster greedy approach finds covers that approximate the optimal score fairly closely.

Next, we test the covering strategies with the LESS algorithm (Algorithm 6) using the LE-sets as candidates. We first ran the test without pruning. In two hours, the optimal variant selected 25 sets to compress the data into 184572 bits. In one minute, the greedy approach found a description of only 163908 bits, using 148 sets. By using a larger number of sets, this approach attains a higher loglikelihood over the data (-22091 and -19767, respectively). Analyzing the resulting code tables, it is evident that the optimal method is more deliberate: if a set is allowed into the code table, it will stay in use and will not be fully traded in, unlike with the greedy method. However, the resulting code tables are very similar to those of the greedy algorithm when pruning is *enabled*. The greedy approach seems to concatenate the “optimal” sets together into 12 sets to achieve a likelihood of -22330, while using only 145940 bits. Overall, the greedy cover algorithm allows MDL to condense the data better. When considered together,

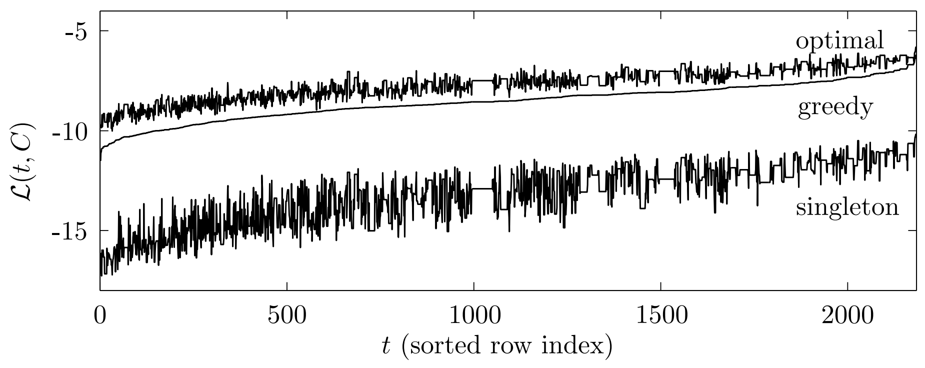


Figure 5.1: The loglikelihood scores for each row in the MAMMALS$_{20}$ dataset using the optimal cover Algorithm 4, the greedy cover Algorithm 5 and singleton (only singleton itemsets) covering. The rows have been sorted according to the loglikelihood score of greedy cover Algorithm 5. The function $\mathcal{L}(t, C)$ denotes the loglikelihood of the row t as defined in Equation (5.2).

this tells us that greedy Algorithm 5 can be used as a fast and high quality alternative to optimal covering Algorithm 4.

5.4.4 Mammals data: an example code table

Based on the experiments, the code tables produced by LESS Algorithm 6 are mostly small enough to fit on a single sheet of paper. Table 5.2 provides an example using MAMMALS. The figure presents a code table computed from a collection of 67767 LE-sets with an entropy of less than 3.3 bits. The resulting code table comprises 29 LE-sets, and is obtained using the greedy covering Algorithm 5 and pruning. Each row in the table corresponds to one low-entropy set, with bullets indicating the attributes of the set.

A full analysis of the code table is beyond the scope of this thesis. However, we notice some familiar pattern elements from the previous chapters. The set with id 27 is similar to the LE-tree in Figure 4.7(a) including Brown Bear, Wood Mouse, Blue Hare and Elk, with Eurasian Lynx replaced with the Greater White-toothed Shrew. The set (id 23) with the species Eurasian Water Shrew, Red Squirrel and Bank Vole can be found similarly in Figure 4.7(b), only with Beech Marten replacing the couple Field Vole and Eurasian Pygmy Shrew. The couple is, however, found together in the set with id 20: Ermine, Field Vole, Eurasian Pygmy Shrew and Grey Wolf.

Taking a more in-depth look at Table 5.3 reveals the instantiation used in the data description for the set with id 4. The pattern includes the two vole species Common Vole and European Pine Vole together with the predators Wildcat and

Table 5.2: A code table (in three subtables) computed from the data set Mammals and a set of 67767 low-entropy sets (with $\epsilon \leq 3.3$). The code table was obtained using the greedy cover Algorithm 5 and pruning. Each row in the table corresponds to one low-entropy set, with bullets indicating the attributes of the set.

Set id	Wildcat	Common Vole	European Pine Vole	European Polecat	Red Deer	European Otter	Edible Dormouse	Beech Marten	Harvest Mouse	Brown Hare	Common Mole	Fallow Deer	European Water Vole	Yellow-necked Mouse	Common Shrew	European Beaver	West European Hedgehog
1	●	●						●	●								
2	●	●					●	●									
3	●	●	●				●										
4	●	●	●	●													
5		●	●			●			●								
6		●	●				●							●			
7				●	●	●											
8					●	●	●										
9				●	●			●									
10					●	●									●		
11						●	●			●	●						
12				●	●					●	●						
13				●	●								●				
14								●		●	●		●				
15									●	●	●	●					
16									●	●	●						●
17							●			●					●	●	
18													●	●			●

Set id	Beech Marten	Common Shrew	European beaver	Roe Deer	European Pine Marten	West European Hedgehog	Eurasian Water Shrew	Red Squirrel	Bank Vole	Raccoon Dog	Ermine	Field Vole	Eurasian Pygmy Shrew	Grey Wolf	Eurasian Lynx	Mediterranean Water Shrew	Lesser White-toothed Shrew	Bicolored Shrew
19		●	●	●	●													
20											●	●	●	●				
21						●					●	●	●					
22		●				●	●	●										
23	●						●	●	●									
24							●	●	●	●								
25						●									●	●	●	
26															●	●	●	●

Set id	Brown Bear	Wood Mouse	Blue Hare	Greater White-toothed Shrew	Elk	Wild Boar	Striped Field Mouse	Garden Dormouse	Southern White-breasted Hedgehog	European Water Vole	Field Vole	Roe Deer	European Pine Marten
27	●	●	●	●	●								
28						●	●	●	●				
29										●	●	●	●

Table 5.3: Detailed view of how LESS Algorithm 6 uses the low-entropy set with set id 4 of Figure 5.2 to encode the data. The set depicts the most relevant presence interactions of the four mammal species according to the low-entropy set selection method.

Wildcat	Common Vole	European Pine Vole	European Polecat	counts
0	0	0	0	44
0	0	0	1	10
0	1	0	0	18
0	1	0	1	199
0	1	1	1	248
1	0	0	0	5
1	0	0	1	7
total # usage				531

European Polecat. The table shows that the major instantiations involve several occurrence combinations, as well as absence of the species. For the 531 rows in the data that this pattern is covers, Common Vole and European Polecat co-exist in about 84 % of cases, and from about 55 % of these cases we see European Pine Vole in the same area. For these cases Wildcat is never present and is found to occur only in 12 cases out of the 531 for this pattern.

Table 5.3 is also a good example of why far fewer patterns are needed with the LE-sets versus frequent itemsets, as was shown in Table 5.1. As the table suggests, relevant interactions involve several occurrence combinations, as well as absence of the species. Hence, if the same interactions had to be described using regular itemsets, many separate sets would be required, instead of the single low-entropy set.

The phenomenon can be seen as a two-level abstraction. The pattern level provides dependencies between attributes, while the instantiation level specifies in detail the attribute value combinations occurring in the data.

5.4.5 Run times

The run times of the experiments ranged from one minute up to ten hours. Analysis shows that the run time is mainly dependent on the number of rows and particularly the size of the code tables. Hence, the timed required for the experiments, where pruning keeps the code tables small, are og the order of few a minutes up to one hour – typically 45 minutes. The experiments with pruning disabled (where the code tables are allowed to grow to hundreds of elements)

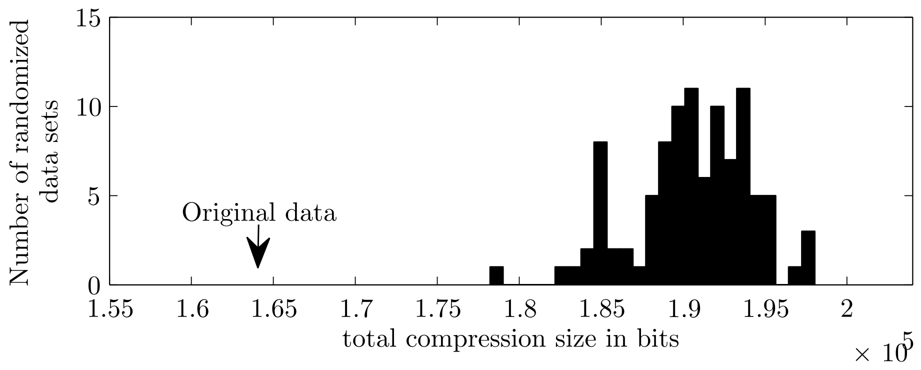


Figure 5.2: Distribution of the compressed sizes of the swap randomized MAMMALS$_{20}$ datasets.

typically took up to three hours, with an exception for MUSHROOM of ten hours.

5.4.6 Model validation via swap randomization

Finally, swap randomization was performed for MAMMALS$_{20}$ to evaluate the extent of relevant structure in the data modeled by LESS using the greedy covering algorithm. The compression score of the model was used as a test statistic.

Figure 5.2 shows the histogram of the compressed sizes of the randomized datasets. We can see that the original data can be compressed significantly better than that of the randomized datasets (empirical p-value of 0.0005). Further, analyzing the contents of the code tables, we also note a significant difference in LE-set cardinality. For the real data, the average set was 2.35 attributes long, while for the randomized data we see sets with an average number of 1.89 attributes.

5.5 Related work

Lately, the pattern explosion problem has attracted a lot of research. For frequent pattern mining, lossless methods such as non-derivable [11] and closed [74] itemsets were proposed to remove the redundancy within the pattern set. Methods that provide a lossy representation of the complete pattern set include maximal itemsets [7].

More recent works on the topic of finding small subsets of informative patterns include [65, 93, 83, 55, 8, 86, 87]. Mielikäinen and Mannila consider in [65] the problem of ordering a collection of patterns so that each prefix of the ordering gives as good an estimation of the attribute frequencies in the data as possible. Yan *et al.* propose in [93] a method that selects k representative patterns that together summarize the pattern set well. Pattern Teams [55] are groups of k

non-redundant patterns that have been exhaustively (k<10) optimized according to criteria such as joint entropy. Bringmann *et al.* [8] propose a greedy variant that can consider larger (in the 100's) pattern sets. All these methods are lossy by nature, in the sense that they find pattern sets that cover only part of the data. Alternatively, pattern sets can be selected on the data description basis, which falls naturally in the compression approach to data mining [22]. Siebes et al. [83] introduced the MDL-based itemset selection algorithm KRIMP. More recent work on itemset selection includes [86] and [87].

Although the selection approach followed here is similar to that of Siebes *et al.* [83], the generality and applicability of the methods are rather different. By considering data as 0/1–symmetric, all major interactions are captured between attributes, not just co-occurrences. Partly thanks to this generalization, the experiments show that in most cases LESS yields tens of patterns, as opposed to hundreds to thousands for KRIMP [83]. Through these much smaller numbers, manual inspection of the pattern set is now possible. Also, these pattern sets have a different meaning, as they view the data in terms of strongly interacting variables, not just present items. Further, the technical solutions proposed here are more general. Instead of using ad-hoc order heuristics to determine which patterns describe what part of the data, a more principled way of finding locally optimal covers of the data through the maximum likelihood principle is introduced.

5.6 Conclusions

In this chapter we discussed a method for selecting small collections of descriptive low-entropy sets through compression. By founding the cover strategies on the maximum likelihood principle, we have a natural approach to using instantiations to describe data only where this makes sense.

The method combines the advantages of both the lossless and lossy approaches to data description; the number of patterns returned is comparable to the latter, while at the same time our pattern sets do provide a lossless description of the data. Based on the experiments, the code tables produced by LESS are mostly small enough to fit on a single sheet of paper. This is a very big advantage when working with domain experts, compared to the hundreds, thousands or tens of thousands of patterns produced by many frequent pattern mining methods.

The small size of the pattern sets is due to a two-level abstraction made possible by low-entropy sets. The pattern level provides dependencies between attributes, while the instantiation level specifies in detail the attribute value combinations occurring in the data. Furthermore, for low-entropy sets the average number of bits needed to encode the values of the attributes is naturally low. Thus, the overall data description length can be efficiently minimized.

Chapter 6

Crossmining binary and numerical data

6.1 Introduction

Many real-world datasets include other types of attribute beside the binary type that we have focused on thus far. For instance, in the MovieLens dataset, the binary part of the data corresponds to users' movie preferences; however, an additional *numerical* part is included, with user age, location (zip code), gender and occupation. Correspondingly, the dataset Mammals includes geographical location (longitude and latitude), as well as environmental observations of mean temperature, mean annual precipitation, mean annual temperature range and average elevation for each grid cell, in addition to the binary presence/absence records of species.

For the MovieLens dataset, it might be of interest to cluster the users according to their demographics: age, location, etc., and to find a small set of characteristic movies preferred by the users in each of these clusters. Similarly, an ecologist might be interested in finding environmentally homogeneous regions in the Mammals data, and in associating with them a small discrete set of characteristic species for each of these regions.

However, neither an itemset mining method, nor a clustering method can alone achieve such a goal very elegantly. Using itemset mining we would need to force the continuous numerical attributes into discrete buckets. Using clustering, the resulting model will characterize the binary attributes of the data with a continuous vector for which the values may be difficult to interpret as either presence or absence.

In this chapter we discuss a new kind of approach to address this problem: *crossmining*. The idea is to find frequent itemsets that provide a link to structure in the numerical attributes of the data. This structure may be defined as a cluster

Dataset							Crossmining patterns
Binary					Numerical		
Elk	Wolf	Rat	Lynx	Bear	Precip-itation	Temper-ature	
1	1	0	0	0	601	14.9	
1	1	1	0	1	583	13.1	
1	1	1	0	1	544	9.5	{Elk, Wolf},(560, 12.3)
1	1	0	1	1	513	11.6	
0	1	1	1	1	426	0.3	
0	0	1	1	0	397	-1.7	{Rat, Lynx},(394, -1.4)
0	0	1	1	1	358	-2.8	

Figure 6.1: An example of a small dataset with a vertical split into binary and real-valued numerical attributes. The itemsets {Elk, Wolf} and {Rat, Lynx} partition the data into two clusters.

or some other type of model[1] suitable for numerical data. The overall task is then to find a partitioning of the numerical part of the data using the structure defined by the frequent itemsets. In this manner both the binary and the numerical parts of the data are modeled in a natural way.

To illustrate this idea, take the example in Figure 6.1. The itemset {Elk, Wolf} occurs in the first four data rows. Considering a k-means type of model, these rows define the mean vector $(560, 12.3)$ for the numerical part of the data. Similarly, for each frequent itemset in the data, a mean vector can be defined based on the numerical attributes and the rows that the itemset occurs in. As in clustering, we can then select a set of k itemsets that gives a small overall modeling error for the numerical part of the data. The figure provides an answer obtained for the two-cluster case using the itemsets {Elk, Wolf} and {Rat, Lynx}.

From the viewpoint of clustering, the idea can be thought of as a way of constraining the search for possible clusters using frequent itemsets. The advantage compared to conventional clustering, such as k-means, is that the resulting model will be more natural in terms of the binary part of the data. Furthermore, it will also be more compact. This is an important observation as the number of binary attributes in a real-life dataset can be very large. For Mammals, it is 194. For MovieLens it is 1682. Moreover, many market basket datasets may have up to tens of thousands of items. A frequent itemset, which typically includes less than twenty items, is a much shorter cluster representative compared with the full vector of, say, 194 species frequencies that the k-means would use for a dataset such as Mammals.

[1] An alternative could be to approximate the numerical attributes with a low-dimensional subspace, or with points having low fractal dimension [21]

Finally, from the viewpoint of itemset mining the idea can be viewed as itemset selection. However here, as opposed to the approach in Chapter 5, the numerical attributes are used to measure the interestingness of the itemsets.

In the following we consider the problem presented above. We discuss a formal definition, study it theoretically, and come up with a simple constant-factor approximation algorithm. Experiments show that the algorithm finds high quality itemsets that convey structure, in both the numerical and binary parts of the data, in a compact and intuitive manner.

6.2 Problem definition

Consider a dataset $\mathcal{D}$ containing two types of attributes, binary-valued attributes and real-valued numerical attributes. We refer to this vertical split on $\mathcal{D}$ by denoting each row $t \in \mathcal{D}$ as $t = [t_B, t_R]$, where t_B is a binary vector and t_R is a real-valued vector. Here we use the convention of referring to the binary part of each row as a subset of the universe of items, i.e., $t_B \subseteq \mathcal{I}$. We will refer to the binary and numerical parts of the data $\mathcal{D}$ as $\mathcal{D}_B$ and $\mathcal{D}_R$, respectively.

Definition 6.1 *Consider data $\mathcal{D}$, a collection of frequent itemsets $\mathcal{P}(\mathcal{D}_B, \sigma)$ (for some σ), and a class of models $\mathcal{M}$. A* candidate model *c is a pair*

$$c = (X, M),$$

where $X \in \mathcal{P}(\mathcal{D}_B, \sigma)$ and $M \in \mathcal{M}$, such that M is the best model that can be assigned to the real-valued attributes of the rows covered by X.

To simplify the discussion we assume in practice that the class $\mathcal{M}$ concerns clustering around a mean, so that

$$M = \text{mean}(\{t_R \mid X \subseteq t_B, t = [t_B, t_R] \in \mathcal{D}\}).$$

However, more complex models are also possible, like the local dimension of the data, or an entropy measure.

For a candidate model $c = (X, M)$, the error of one row $t = [t_B, t_R] \in \mathcal{D}$ with respect to c is defined as follows,

$$e(t \mid c) = \begin{cases} \|t_R - M\|^2 & \text{if } X \subseteq t_B, \\ \infty & \text{otherwise.} \end{cases} \tag{6.1}$$

The idea is to assign a candidate model c for each row $t \in \mathcal{D}$ such that the error defined by Equation (6.1) is as small as possible. However, depending on the pattern collection $\mathcal{P}$, it could be that for some $t = [t_B, t_R] \in \mathcal{D}$ there is no $c = (X, M)$, such that $X \subseteq t_B$. In this case the error will run off to infinity. To ensure that this does not happen, we will always assume that there exists a default model that covers all the rows in the data $\mathcal{D}$. This will be the model defined by the empty set, default $= (\emptyset, M_0)$ with M_0 being the mean of the real-valued attributes of the entire data.

Since the default model can always be assigned to any row t, it only makes sense to assign a candidate model c to t when c yields a smaller error than the default model. Indeed, given several candidate models $S = \{c_1, \ldots, c_k\}$ we want to assign the model $c_i \in S$ that yields the largest reduction in error, that is,

$$g(t \mid S) = \max_{c_i \in S}\{e(t \mid \text{default}) - e(t \mid c_i), 0\}. \tag{6.2}$$

Given a fixed set of candidate models in S, we usually say that a row $t \in \mathcal{D}$ is assigned (or modeled) by the best candidate model in S. Respectively, this model will correspond to the argument of the maximum of Equation (6.2). If we have $g(t \mid S) = 0$, the default model is assigned.

The problem we study here is the following:

Problem 6.1 *Given a dataset $\mathcal{D}$, a collection*[2] *of frequent itemsets $\mathcal{P}(\mathcal{D}_B, \sigma)$, for some σ, a class of models $\mathcal{M}$ and a set of candidate models*

$$C = \{(X, M) \mid X \in \mathcal{P}(\mathcal{D}_B, \sigma),\ M \in \mathcal{M}\},$$

find a k-size subset $S \subseteq C$ of candidate models such that the sum

$$G(\mathcal{D}, S) = \sum_{t \in \mathcal{D}} g(t \mid S) \tag{6.3}$$

is maximized.

Note that Problem 6.1 could have been defined with respect to minimizing

$$e(t \mid S) = \min_{c_i \in S}\{e(t \mid \text{default}), \{e(t \mid s_i)\}, \tag{6.4}$$

where $e(t \mid S)$ is the absolute error obtained by the candidate model assignment for each row t. Our analysis of the problem will, however, benefit more from formulating the problem as the maximization variant. Both Equation (6.2) and Equation (6.4) optimize exactly the same thing.

6.3 Problem properties

Next we consider the basic properties of Problem 6.1. We start with complexity considerations.

[2]In practice, the collection $\mathcal{P}$ may correspond to frequent sets, frequent closed sets or maximal frequent sets (recall Section 2.2.4) mined on the binary attributes of the data. Because closed sets are a lossless representation of frequent sets it will be better in practice to directly use frequent closed sets instead of the regular frequent itemsets.

6.3.1 Complexity

Theorem 6.1 *Problem 6.1 is NP-hard.*

Proof Consider an instance of the NP-complete set cover problem, defined by a collection of m subsets $S_1, \dots, S_m$ such that each $S_i \subseteq U$ where U is the universe of n objects $U = \{u_1, \dots, u_n\}$. We want to know whether there exists k sets from $\{S_i\}$ whose union is equal to U.

Given an arbitrary instance of the set cover problem, we reduce it to an instance of Problem 6.1 as follows. Let $\mathcal{I} = \{A_i \mid i \in [1 \dots m]\}$ and $\mathcal{D} = \{t_j \mid j \in [1 \dots n]\}$. We then define each row as $t_j = [t_B^j, t_R^j]$ with $t_B^j = \{A_i \mid u_j \in S_i,\ i \in [1 \dots m]\}$ and $t_R^j = 1$. That is, for the binary part, we have $t_B^j(A_i) = 1$ if and only if $u_j \in S_i$. In other words, the row occurrence of the binary attribute A_i represents the elements of the set S_i. The numerical part consists of rows that all have a single real-valued attribute with the value 1. Finally, we define as many candidate models as there are binary attributes, that is $c_i = (\{A_i\}, 1)$ for $i \in [1 \dots m]$, and a default model default $= (\emptyset, r_0)$ for some $r_0 > 1$.

The set cover problem is equivalent to deciding if there exists a set of k candidate models that yield the sum $n \cdot (1 - r_0)^2$ for Equation (6.3). □

While mean models were used in the proof, it is fairly easy to see that most nontrivial model classes admit a similar proof. The hardness of the result relies on the exponential number of decisions for covering with candidate models, not on the details of the model class $\mathcal{M}$ itself. If the number of itemsets defining the candidate models is constant, the number of possible k size solutions is also bounded by a constant. In such cases the problem could be solved in time proportional to $|\mathcal{D}|s$, where s is the possible number of all solutions, and $|\mathcal{D}|$ the number of rows in $\mathcal{D}$. Of course, the number of possible itemsets may be exponentially large if a very low frequency threshold σ has been used.

6.3.2 Approximability

Despite NP-hardness, it turns out that by using submodular functions it is possible to show approximation guarantees for Problem 6.1. A function is said to be submodular if it satisfies the following property: the marginal gain from adding an element c to the solution set S decreases if the element c is added to a superset of S: $f(S \cup \{c\}) - f(S) \geq f(S' \cup \{c\}) - f(S')$ for $S \subseteq S'$.

Submodular functions have been extensively studied for optimization problems [25]. Particularly interesting for our problem is the following result shown by Nemhauser, Wolsey, and Fisher [69].

Theorem 6.2 ([69]) *For a non-negative, monotone submodular function f, let S be a solution set of size k obtained by selecting an element $c \notin S$ one at a time, each time choosing the element that provides the largest marginal gain $f(S \cup \{c\}) - f(S)$. Let S^* be a set that maximizes the value of f, then $f(S) \geq (1 - 1/e) \cdot f(S^*)$.*

The use of this submodularity result for Problem 6.1 resembles maximizing the spread of influence through a social network in [50]. Because the result of Theorem 6.2 is very general, it has been applied to other optimization problems from the literature as well (see Section 6.6 for related work).

Our strategy is to show that the function we wish to maximize in Problem 6.1 satisfies the properties defined in Theorem 6.2: it is always non-negative, monotone and submodular. By ensuring these three properties the approximation guarantee of Theorem 6.2 will immediately follow. Recall that the optimization function of Problem 6.1 is the following.

$$G(\mathcal{D}, S) = \sum_{t \in \mathcal{D}} g(t \mid S) \tag{6.5}$$

By the definition of $g(t \mid S)$, it is immediately clear that the function is always non-negative. Also, we could assume without loss of generality that $G(\mathcal{D}, \emptyset) = 0$. The other two properties are proved next. We make use of the following notation.

We denote with $\mathcal{D}_c^S$ the subset of rows from $\mathcal{D}$ that are assigned to the candidate model c, when c is added to the current set of solution models S. Formally, $\mathcal{D}_c^S = \{t \in \mathcal{D} \mid c = \text{argmax}_{S \cup \{c\}}\ g(t \mid S \cup \{c\})\}$.

Observe that for all $t \in \mathcal{D}_c^S$, we always have $e(t \mid c) \leq e(t \mid c')$ for each $c' \in S$. Another way of writing this is $\mathcal{D}_c^S = \{t \in \mathcal{D} \mid e(t \mid c) \leq e(t \mid c'), \forall c' \in S\}$. This observation is important for the monotonicity property.

Proposition 6.1 *(Monotonicity)* $G(\mathcal{D}, S \cup c) \geq G(\mathcal{D}, S)$.

Proof For the candidate models in S and a single candidate model c, the rows in $\mathcal{D}$ can be expressed as $\mathcal{D} = \mathcal{D}_c^S \cup \mathcal{D} \setminus \mathcal{D}_c^S$. We can now decompose $G(\mathcal{D}, S \cup \{c\})$ into the score measured for $\mathcal{D}_c^S$ plus the score for the rest of the rows in $\mathcal{D}$.

$$G(\mathcal{D}, S \cup \{c\}) = \sum_{t \in \mathcal{D} \setminus \mathcal{D}_c^S} g(t \mid S) + \sum_{t \in \mathcal{D}_c^S} g(t \mid \{c\})$$

The same decomposition can be applied for $G(\mathcal{D}, S)$, but this time all rows in $\mathcal{D}$ can only be assigned to models in S. That is,

$$G(\mathcal{D}, S) = \sum_{t \in \mathcal{D} \setminus \mathcal{D}_c^S} g(t \mid S) + \sum_{t \in \mathcal{D}_c^S} g(t \mid S)$$

Subtracting now the second equation from the first,

$$G(\mathcal{D}, S \cup \{s\}) - G(\mathcal{D}, S) = \sum_{t \in \mathcal{D}_c^S} \left[g(t \mid \{c\}) - g(t|S) \right] \tag{6.6}$$

By construction for all $t \in \mathcal{D}_c^S$ we have $e(t \mid c) \leq e(t \mid c')$ for every $c' \in S$. From this it follows that

$$e(t \mid \text{default}) - e(t \mid c) \geq \max_{c' \in S} \{e(t \mid \text{default}) - e(t \mid c'), 0\}.$$

Hence, for all $t \in \mathcal{D}_c^S$, we have $g(t \mid \{s\}) - g(t \mid S) \geq 0$, and therefore, $G(\mathcal{D}, S)$ is monotone. $\square$

Next we show the submodularity property of the function $G(\mathcal{D}, S)$.

Proposition 6.2 *(Submodularity)* $G(\mathcal{D}, S \cup \{c\}) - G(\mathcal{D}, S) \geq G(\mathcal{D}, S' \cup \{c\}) - G(\mathcal{D}, S')$*, with* $S \subseteq S'$.

Proof For the sake of simplicity let $\Delta(S) = G(\mathcal{D}, S \cup \{c\}) - G(\mathcal{D}, S)$ and $\Delta(S') = G(\mathcal{D}, S' \cup \{c\}) - G(\mathcal{D}, S')$. From Equation (6.6) in the previous proof we can write the value of $\Delta(S)$ as the increment of having rows reassigned from candidate models S to candidate models $S \cup \{c\}$. As is shown, only rows in $\mathcal{D}_c^S$ contribute to this value. The same applies to $\Delta(S')$ and the set of rows $\mathcal{D}_c^{S'}$. That is,

$$\begin{aligned}
\Delta(S) &= \sum_{t \in \mathcal{D}_c^S} \big[g(t \mid \{c\}) - g(t \mid S)\big] \\
\Delta(S') &= \sum_{t \in \mathcal{D}_c^{S'}} \big[g(t \mid \{c\}) - g(t \mid S')\big]
\end{aligned}$$

Because $S \subseteq S'$ we know that $\mathcal{D}_c^{S'} \subseteq \mathcal{D}_c^S$. Therefore, rows $\mathcal{D}_c^S$ can be divided into $\mathcal{D}_c^S = \mathcal{D}_c^{S'} \cup (\mathcal{D}_c^S \setminus \mathcal{D}_c^{S'})$. This allows the following simple rewriting of $\Delta(S)$.

$$\begin{aligned}
\Delta(S) &= \sum_{t \in \mathcal{D}_c^{S'}} \big[g(t \mid \{c\}) - g(t \mid S)\big] \\
&\quad + \sum_{t \in \mathcal{D}_c^S \setminus \mathcal{D}_c^{S'}} \big[g(t \mid \{c\}) - g(t \mid S)\big]
\end{aligned}$$

From the monotonicity property we always have that $g(t \mid S') \geq g(t \mid S)$. It immediately follows that

$$\begin{aligned}
\Delta(S) &\geq \sum_{t \in \mathcal{D}_c^{S'}} \big[g(t \mid \{c\}) - g(t \mid S')\big] \\
&\quad + \sum_{t \in \mathcal{D}_c^S \setminus \mathcal{D}_c^{S'}} \big[g(t \mid \{c\}) - g(t \mid S)\big] \\
&= \Delta(S') + \sum_{t \in \mathcal{D}_c^S \setminus \mathcal{D}_c^{S'}} \big[g(t \mid \{c\}) - g(t \mid S)\big] \\
&\geq \Delta(S')
\end{aligned}$$

Thus, we have $G(\mathcal{D}, S \cup \{c\}) - G(\mathcal{D}, S) \geq G(\mathcal{D}, S' \cup \{c\}) - G(\mathcal{D}, S')$ and the function is submodular. $\square$

6.3.3 A note on maximization versus minimization

As mentioned in Section 6.2, the score $g(t \mid S)$ of Equation (6.2) could also have been defined as a minimization function of the form

$$e(t \mid S) = \min_{c_i \in S}\{e(t \mid \text{default}), \{e(t \mid s_i)\}.$$

This way Problem 6.1 would have been transformed into minimizing the sum

$$E(\mathcal{D}, S) = \sum_{t \in \mathcal{D}} e(t \mid S). \tag{6.7}$$

It is easy to see that the same constant factor approximation result of Theorem 6.2 does not apply directly to this quantity. Note, however, that

$$G(\mathcal{D}, S) = \sum_{t \in \mathcal{D}} e(t \mid \text{default}) - E(\mathcal{D}, S).$$

By expressing the bound obtained from Theorem 6.2 in terms of $E(\mathcal{D}, S)$, we have the following bound for the minimization variant of the problem

$$E(\mathcal{D}, S) \leq (1 - 1/e) \cdot E(\mathcal{D}, S^*) + 1/e \sum_{t \in \mathcal{D}} e(t \mid \text{default}).$$

6.4 Algorithms

From Theorem 6.2, we have the following greedy algorithm: Given a collection of candidate models C, at all times maintain a candidate solution set S. Choose the next element $c \in C : c \notin S$ having the largest marginal gain $G(\mathcal{D}, S \cup \{c\}) - G(\mathcal{D}, S)$ until $|S| = k$. As discussed above, this greedy approach will yield a solution of candidate models S such that $G(\mathcal{D}, S) \geq (1 - 1/e) \cdot G(T, S^*)$, where S^* is the optimal solution to Problem 6.1.

After adding a candidate model c to the solution set S, the greedy routine will keep c in S until the end. However, it may be that at some point of the greedy routine, a large portion of the rows modeled by c will be reassigned to some other segment c' added after c, rendering s largely inutile in S. Furthermore, the few remaining rows $\mathcal{D}_c^S \setminus \mathcal{D}_{c'}^S$ might be better assigned to a more specific candidate model $\hat{c}$ in $C \setminus S$. Hence, after running the greedy algorithm we further optimize the solution by performing local swaps between the sets in S and $C \setminus S$, that is, if $G(\mathcal{D}, S) \leq G(\mathcal{D}, (S \setminus c) \cup \hat{c})$, for some $c \in S$ and $\hat{c} \in C \setminus S$, we swap c with $\hat{c}$ in the results set S.

Algorithm 7 gives a pseudo-code description of the greedy algorithm for Problem 6.1.

Algorithm 7 Greedy crossmining algorithm.

```
Input: A dataset $\mathcal{D}$, a set of candidate models $C$, and $k$ the size of the solution set $S$.
Output: A subset $S \subseteq C$ of size $k$.
 1: // First the greedy marginal gain optimization.
 2: $S = \emptyset$
 3: while $|S| < k$ do
 4:     $s = \text{argmax}_{c \in C \setminus S} G(\mathcal{D}, S \cup \{c\}) - G(\mathcal{D}, S)$
 5:     $S = S \cup s$
 6: end while
 7: // Finishing with the swaps.
 8: for all $(s, c)$, such that $s \in S, c \in C \setminus S$ do
 9:     if $G(\mathcal{D}, S) \leq G(\mathcal{D}, (S \setminus s) \cup c)$ then
10:         $S = (S \setminus s) \cup c$
11:     end if
12: end for
13: return $S$
```

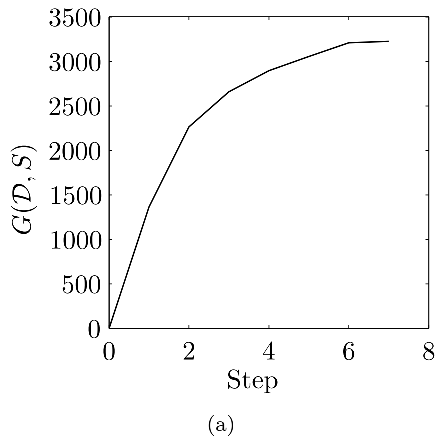
6.5 Experiments

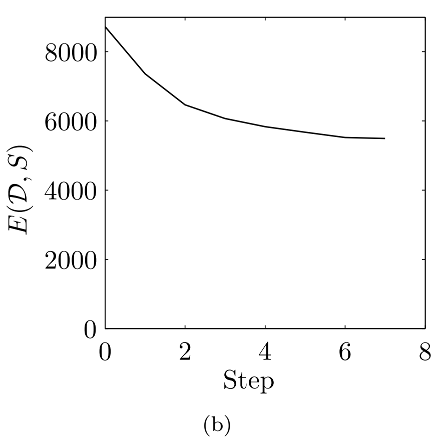
To verify the applicability of the crossmining method an implementation of Algorithm 7 using Matlab was constructed. Its behavior was tested on the two datasets that both have binary and numerical real-valued attributes: Mammals and MovieLens. To ease the intuition of the results obtained, only mean models were considered in the experiments. All numerical attributes were normalized to zero mean and unit variance.

6.5.1 Mammals data

A collection of closed itemsets mined with a minimum support threshold of 25% was used as the basis of the candidate model set for the crossmining of the dataset Mammals. To reduce redundancy in the candidate model set, only items with a frequency of less than 80% were considered. This resulted in a collection of 55 130 closed itemsets. As numerical data mean annual temperature, mean annual precipitation, mean annual temperature range and average elevation of each grid cell were used.

Figure 6.2 shows the development of the score $G(\mathcal{D}, S)$ and error $E(\mathcal{D}, S)$ as a function of the steps taken by Algorithm 7, while selecting a set of six candidate models. The figure shows that the first candidate models, corresponding to itemsets {Elk} and {Hedgehog, Wood mouse}, reduce the error heavily, while the gains in error reduction for the last steps, when adding {Racoon Dog, Brown Hare} and {Blue Hare}, get smaller. As the last step the algorithm swaps the itemset {Elk} with a refined set of {Elk, Weasel, Squirrel}, further introducing a small reduction in the error function. No other swaps in the process of the





Step	Action		Score gain
1	Select	{Elk}	1364
2	Select	{West European Hedgehog, Wood Mouse}	899
3	Select	{Edible Dormouse, Bank Vole, Roe Deer}	394
4	Select	{Wildcat}	237
5	Select	{Racoon dog, Brown hare}	160
6	Select	{Blue hare}	155
7	Swap	{Elk} with {Elk, Ermine, Red Squirrel}	26

Figure 6.2: Score/error function convergence during selection of 6 itemsets from the dataset Mammals using Algorithm 7. The functions $G(\mathcal{D}, S)$ and $E(\mathcal{D}, S)$ are defined as in Equations (6.2) and (6.7), respectively.

algorithm are made, giving support to the quality of the initial result obtained by the pure greedy algorithm.

Figure 6.3 shows the geographical mapping of the clusters defined by the final candidate models, while Table 6.1 gives a closer look at their numerical descriptions. Scandinavian countries are modeled with the itemsets {Elk, Weasel, Red Squirrel} and {Blue Hare}. More specifically, the former describes areas across Finland and Sweden having low average temperature and low average elevation, while the latter defines an area within Norway of higher average elevation. Coastal regions, mostly within western Europe and the British Isles, are characterized by the itemset {West European Hedgehog, Wood Mouse}. The grid cells covered by this itemset experience on average high precipitation. The itemset {Dormouse, Vole, Roe Deer} covers the mountain systems across Alps and Pyrenees, while the itemset {Wildcat} characterizes a cluster of southern European countries, with mainly high average temperatures. The candidate model related to the itemset {Racoon dog, Brown hare} is assigned to the areas covering the Baltic states and Poland. Notice that the final coverage of the candidate model

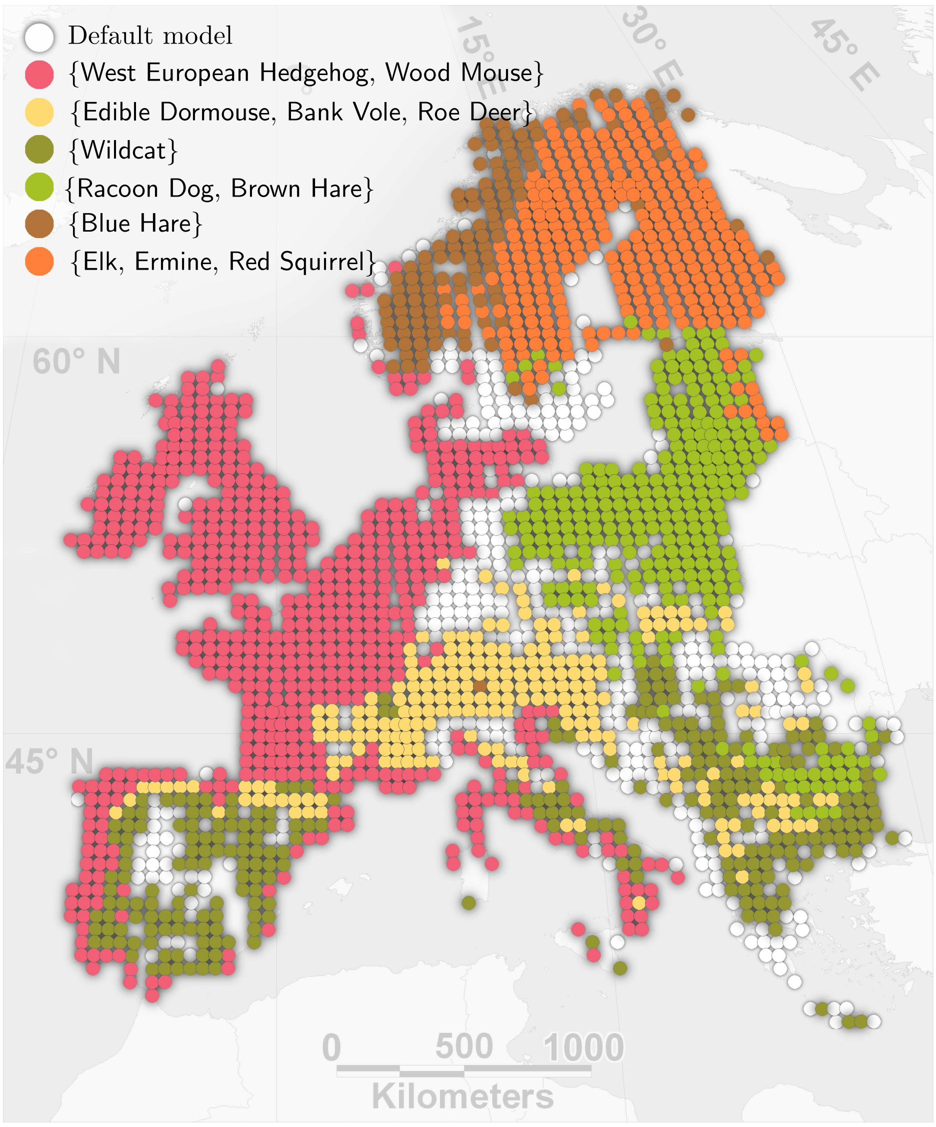

Default model
{West European Hedgehog, Wood Mouse}
{Edible Dormouse, Bank Vole, Roe Deer}
{Wildcat}
{Racoon Dog, Brown Hare}
{Blue Hare}
{Elk, Ermine, Red Squirrel}
15° E
30° E
45° E
60° N
45° N
0
500
1000
Kilometers

Figure 6.3: Spatial mapping of the clusters defined by the 6 final candidate models for Mammals.

Table 6.1: Numerical description of the 6 best candidate models for Mammals. The columns Elev., Precip., Temp. and Temp. range correspond to the mean of the attributes mean average elevation, mean annual precipitation, mean annual temperature and mean annual temperature range of those rows covered by the candidate model itemset. Frequency corresponds to the number of rows that the itemset occurs in. Final cover is the number of rows that are eventually assigned to the actual candidate model.

Itemset	Mean model				Frequency	Final
	Elev.	Prepic.	Temp.	Temp. range		cover
	(m)	(mm/year)	(C°)	(C°)		
Default model	339.2	758.2	8.2	27.0	2183	394
{Elk, Ermine, Red Squirrel}	248.9	667.2	3.4	31.0	592	297
{West European Hedgehog, Wood Mouse}	317.8	811.3	9.9	23.9	1103	603
{Edible Dormouse, Bank Vole, Roe Deer}	540.2	827.1	8.7	26.9	557	224
{Wildcat}	460.2	748.8	10.9	27.4	624	264
{Racoon Dog, Brown Hare}	204.7	633.5	6.8	30.1	552	284
{Blue Hare}	354.1	808.7	3.7	28.7	721	117

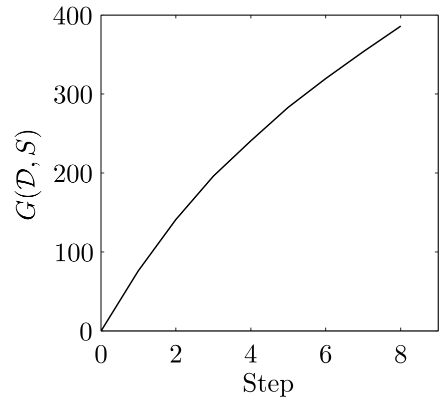
is not necessarily equal to the frequency of the corresponding itemset. For instance, while the itemset {Elk, Weasel, Red Squirrel} covers 592 grid cells in the data, a subset of 297 grid cells are best described by its candidate model in the final result. This is due to the choice of overlapping candidate models: rows covered by multiple itemsets are always assigned to the best fitting candidate model.

6.5.2 MovieLens data

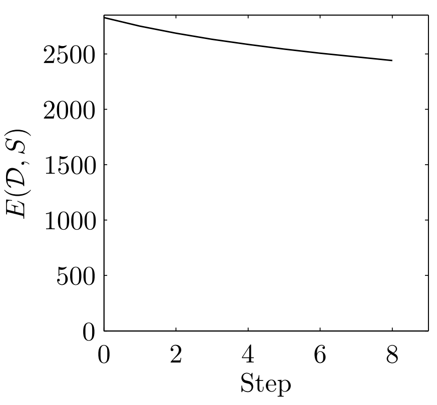
A set of maximal frequent itemsets mined with a minimum frequency threshold of 6% were used as the basis for the candidate models of the MovieLens dataset. This resulted in a collection 63 432 candidate models.

Figure 6.4 shows the development of the score $G(\mathcal{D}, S)$ and error $E(\mathcal{D}, S)$ as a function of the steps taken by Algorithm 7, while selecting a set of eight candidate models. We notice that, in comparison to Mammals, the relative reduction in error is much smaller here. Still, no local swaps are made at the end of the process, indicating that the result of the greedy algorithm is optimal with respect to swapping.

Figure 6.5 visualizes the resulting models chosen for the final crossmining partition with respect to the attributes age and gender. Recall that the default model reflects the average of the attributes over the entire data. The default model is mapped into the middle of the plot. The movie combinations {Das Boot (1981), The Full Monty (1997)} and {The Treasure of the Sierra Madre (1948), The Bridge on the River Kwai (1957)} can be considered to characterize the preferences of groups of older male users, while the taste of younger male users may be characterize by the movie combinations {Trainspotting (1996), Scream (1996)} and



(a)



(b)

Step	Action		Score gain
1	Select	{The Treasure of the Sierra Madre (1948), The Bridge on the River Kwai (1957)}	76
2	Select	{Little Women (1994)}	65
3	Select	{Liar Liar (1997), Scream (1996)}	55
4	Select	{Rosewood (1997)}	45
5	Select	{Trainspotting (1996), Scream (1996)}	42
6	Select	{Secrets & Lies (1996), Fargo (1996), The English Patient (1996)}	36
7	Select	{The Truth About Cats & Dogs (1996)	34
8	Select	{Das Boot (1981), The Full Monty (1997)}	32

Figure 6.4: Score/error function convergence during selection of 8 itemsets from the dataset MovieLens using Algorithm 7. The functions $G(\mathcal{D}, S)$ and $E(\mathcal{D}, S)$ are defined as in Equations (6.2) and (6.7), respectively.

{Liar Liar (1997), Scream (1996)}, according to the models. The itemset {Little Women (1994)} is chosen as a characterization of female preferences. Respectively, the itemsets {Rosewood (1997)} and {Secrets & Lies (1996), The English Patient (1996), Fargo (1996)} characterize a group of slightly older women, while the combination {The Truth About Cats & Dogs (1996), Jerry Maguire (1996)} characterizes slightly younger women. Table 6.2 gives a further numeric description of the 8 final candidate models for MovieLens.

6.5.3 Comparison with k-means clustering

The outcome of the crossmining method was experimentally compared with the outcome of the standard k-means algorithm. The dataset Mammals and the result of Section 6.5.1 were used as the bases for comparison. Furthermore, three k-means clusterings were performed. One clustering was optimized using the

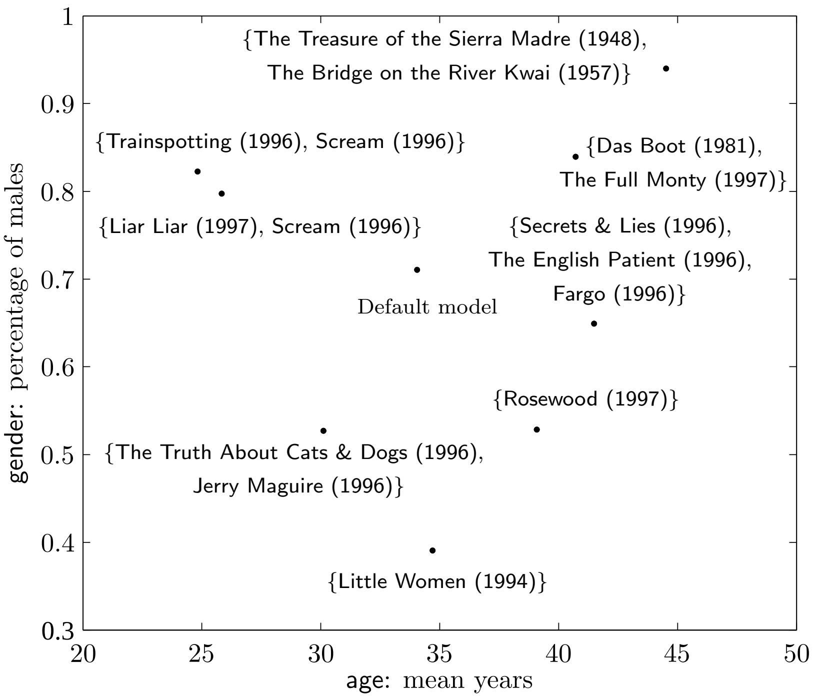


Figure 6.5: Visualization of the resulting 8 movie candidate models with respect to the attributes age and gender for the crossmining results for the dataset MovieLens.

numerical part of the data. This clustering is denoted by k-means$_{\mathbb{R}}$. A second clustering was optimized using the binary part of the data. This clustering is denoted by k-means$_{\mathbb{B}}$. Finally, a third clustering, denoted here by k-means$_{\mathbb{R}\cup\mathbb{B}}$, was optimized using both the binary and numerical part of the data together.

The within-cluster sum-of-squares (as defined in Equation (2.2) of Chapter 2) was used as the error metric. Note that each of the methods, including Algorithm 7, will give an assignment of the data points into k distinct clusters (partitions). For each partition the corresponding mean vectors were computed in both the binary and numerical parts of the data. These means were then used to compute the sum-of-squares errors in each case.

Notice here that in case of crossmining, the means of the clusters are not necessarily the same as the mean models defined by the candidate models. The candidate model is computed over all the rows covered by the itemset. However, one point may be covered by several itemsets but assigned to only one candidate model in the final crossmining results. Here the error was computed with respect to the mean of the partitions, as opposed to using the candidate model means.

Note also that the numerical part of Mammals features four numerical at-

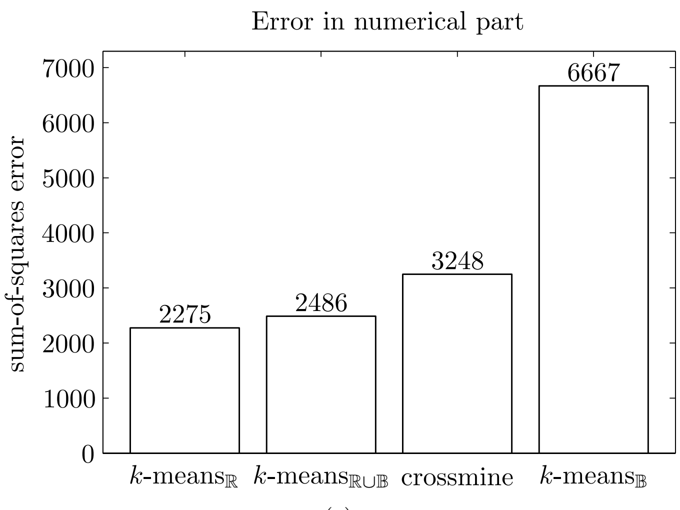
Table 6.2: Numerical description of the 8 final candidate models for MovieLens. Frequency corresponds to the number of rows that the itemset occurs in. Final cover is the number of rows that are eventually assigned to the actual candidate model.

Itemset	Mean model		Fre-	Final
	Age	Gender	quency	cover
	(Years)	Male ratio		
Default model	34.05	0.71	943	667
{The Treasure of the Sierra Madre (1948), The Bridge on the River Kwai (1957)}	44.52	0.94	50	33
{Little Women (1994)}	34.70	0.39	64	38
{Liar Liar (1997), Scream (1996)}	25.84	0.80	74	44
{Rosewood (1997)}	39.09	0.53	70	37
{Trainspotting (1996), Scream (1996)}	24.82	0.82	62	44
{Secrets & Lies (1996), Fargo (1996), The English Patient (1996)}	41.49	0.65	57	24
{The Truth About Cats & Dogs (1996), Jerry Maguire (1996)}	30.11	0.53	74	28
{Das Boot (1981), The Full Monty (1997)}	40.71	0.84	56	28

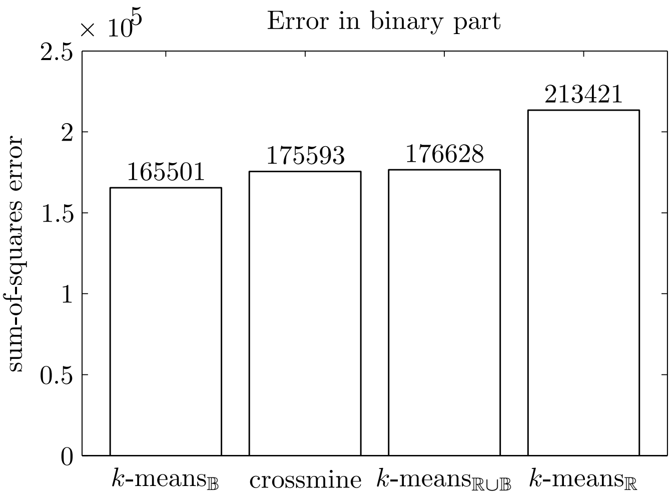
tributes while the number of binary attributes is 124. Hence, for clustering in the combined space of both binary and numerical valued attributes, the two parts of the data were weighted such that the sets of both binary and the numerical valued attributes had equal weight with respect to the overall score. In practice this was done by copying the set of the 4 numerical attributes 31 times to obtain a weight of 124 attributes matching the binary part of the data.

The crossmining result presented in Section 6.5.1 features k=6 final candidate models. To have an equal number of parameters for the methods, k=6 was also used with the k-means clusterings. Note however, that Algorithm 7 may assign some points to the default model. Hence, the rows assigned to the default model were excluded from the comparison. The three k-means clusterings (for $k = 6$) were computed for these remaining data rows.

The result in Figure 6.6 shows that in crossmining, constraining the search of possible clusters with frequent itemsets comes with a slight penalty in the numerical part of the data when compared with k-means$_{\mathbb{R}\cup\mathbb{B}}$. However, the difference is not very large. On the other hand, for the binary part of the data, crossmining performs equally well, or even slightly better. Hence, we see that the compact itemset based representation used by crossmining is in this case as effective as the mean vectors of k-means$_{\mathbb{R}\cup\mathbb{B}}$ in capturing structure in the binary part of the data. In other words, with a slight reduction in modeling error for the numerical part of the data, we get the shorter discrete set of models offered by the crossmining approach for the binary part of the data.



(a)



(b)

Figure 6.6: Comparison of the within-cluster sum-of-squares error between three k-means clusterings (with $k = 6$) and the result presented in Table 6.1 (excluding data rows assigned to the default model). The bars have been ordered according to the resulting errors.

6.6 Related work

The problem defined here exhibits an apparent similarity to the facility location problem [82, 47, 61], where the aim is to "open" some number of facilities to serve customers and typically there is a fixed cost for each facility opened plus a penalty for each customer-facility distance. The nature of this problem has produced mainly results for minimization of the proper score function. Of course, as mentioned earlier, Problem 6.1 could also be defined in the minimization form. Still, the details in the optimization setting of the facility location problem are typically different from the setting presented here: the most relevant approximation results, related to the k-median problem, assume that the customer-facility distances have the metric property. This does not hold here. A good summary of the approximation bounds for the facility location problems can be found in [47].

Crossmining is also closely related to the segmentation problems from [51, 52]. The bounds obtained here correspond to the bounds obtained for the catalog segmentation problem, which can be seen as a generalization of the problem presented here. However, the idea of using collections of itemsets to bound the space of decisions to good models is new in this context. Moreover, the analysis of Algorithm 7 does not come from any existing work, nor does the idea of providing a characterization of the final model with a set of relevant features corresponding to itemsets. Other works in the current literature that use submodular functions to prove approximation bounds include [50]

A similar application to crossmining is clustering [60, 46, 33, 88, 92]. The contribution here is two-fold. First, from the numerous clustering methods it usually becomes difficult to evaluate the partition, especially when the number of binary attributes is large. In our case, the partition is always properly represented by a unique itemset which defines a small set of relevant features describing the partition. Second, one could cluster numerical attributes independently from the binary attributes, and then mine itemsets from each partition. In this case it is not clear what are the characteristic itemsets for each partition. Furthermore, different partitions could end up being characterized by the same itemset. Using crossmining, however, these questions are addressed.

The crossmining problem can also be seen as a form of constrained clustering. In typical constrained clustering algorithms, the constraints are of the form of a *cannot-link* and *must-link* [90]. A *cannot-link* constraint says that rows t_1 and t_2 must belong to different clusters, while a *must-link* forces them to be in the same cluster. In crossmining a *cannot-link* can be encoded by assigning disjoint binary parts to the rows. A *must-link* constraint can be approximated by assigning the rows exactly to the same binary attributes. In the case of the *must-link*, this does not, however, guarantee that the rows end up in the same segment if they have very different numerical attributes. If *must-link* constraints are required in the application, they can of course be added to the crossmining problem.

In [43] Hollmén *et al.* study the collections of frequent sets in clusters produced by a probabilistic clustering using mixtures of Bernoulli models. The approach discussed here is the reverse: first a collection of frequent sets is generated, then based on which the partition is produced. Conceptually closer are Sese and Morishita [81] with the idea of itemset classified clustering. As here, they study data with both binary and numerical attributes. They search for itemsets that maximize interclass variance in the numerical attributes between the rows that an itemset covers and the respective complement. The difference from the crossmining method is that they try to find interesting groups of examples rather than a set of itemsets that cover all examples. In [98] Zenko *et al.* address the idea of learning predictive clustering rules. They combine decision trees learning and rule learning with concepts of clustering in a setting that is fairly close to that used in this thesis. However, their approach is based on the CN2 rule induction algorithm [15] and is more general in the sense that it allows continuous variables in both parts of the data.

Finally, the results contribute to itemset selection: here itemsets are selected based on their ability to describe structure in the numerical part of the data. Algorithm 7 can be seen as providing the best top-k itemsets in the data based on auxiliary numerical data. Return to Section 5.5 for more related work on pattern and itemset selection.

6.7 Conclusions

In this chapter we have studied the problem of mining binary data along with the corresponding numerical data. From the viewpoint of clustering, the approach can be thought of as a way of constraining the search for possible clusters using frequent itemsets. From the viewpoint of itemset mining, the approach can be seen as itemset selection.

We concluded that the problem can be solved with a simple greedy constant-factor approximation algorithm. Experiments on numerical data show that the algorithm finds high quality itemsets that also convey structur in the numerical part of the data. The advantage here, compared with clustering, such as k-means, is that the results yield more natural and more compact cluster representations for the binary part of the data.

Chapter 7

Conclusions and discussion

In this thesis we have studied new methods for enhancing the expressive power of frequent pattern mining. The goal has been to come up with methods that have the ability to express new kinds of relationships between attributes and to convey only the most relevant and important interactions of the attributes in a concise and non-redundant manner.

In Chapter 3 we discussed a new type of tree pattern class for expressing hierarchies of general and more specific attributes in unstructured binary data. The new introduced definition has advantageous properties: high quality patterns are unlikely to occur in random data, and the definition allows construction of a simple level-wise algorithm. The results suggest that trees can be used to discover relationships in data that cannot be expressed alone with the more traditional frequent itemset or association rule type of patterns.

Chapter 4 proposed a new type of score for frequent pattern mining, entropy. The score makes it possible to express more general types of occurrence structure than with frequent itemsets. Entropy can easily be applied to both set and tree types of pattern, and as a monotonic concept allows the use of the level-wise approach.

Chapter 5 presents the idea of using low-entropy based patterns and minimum description length (MDL) for compact data description. It was shown that each row in the data can be encoded using the maximum likelihood principle, which is in relation to minimizing data encoding length. Harnessing the expressive power of the low-entropy sets, the method gives small and easily interpretable collections of patterns that in most cases are an order of magnitude smaller than by using frequent itemsets.

Chapter 6 introduced the idea of relating itemsets to numerical variables in a database of mixed data types. This can be considered either as a pattern selection approach or a constraint clustering problem. Theoretical contributions included proofs of NP-hardness and a simple greedy constant-factor approximation algorithm. Experiments on data, including both binary and numerical attributes, showed that the algorithm finds high quality itemsets that convey structure in both the numerical and the binary part of the data in a compact and intuitive manner.

Many interesting issues related to the methods proposed here still remain unaddressed. A drawback of the tree mining algorithm used in Chapter 3 is that each tree is generated multiple times as an isomorphic copy of itself. Research on finding trees from relational tree- or graph-structured data [12] is already addressing many issues related to efficient tree enumeration. It would be interesting, however, to study the special properties of this problem from a more algorithmic point of view, in order to come up with a more efficient way of generating the defined tree pattern class.

The score for low-entropy sets proposed in Chapter 4 is unnormalized and unpenalized by the size of the pattern. The reason for this is that the monotonicity property breaks when the most obvious normalizations are brought in. However, it would be interesting to investigate whether there exists some other meaningful normalization that would still enable efficient discovery of the proposed entropy-based pattern classes.

Both pattern selection frameworks, discussed in Chapters 5 and 6, are based on two basic steps. In the first step, a complete set of candidate patterns, using traditional frequent pattern mining algorithms, is generated. In the second step, pattern selection is applied. To minimize the number of candidates generated, a relevant question is whether it is possible to omit the first step and mine the candidates patterns as we proceed.

For the low-entropy set selection problem the overall approach was based on heuristics. An obvious area for research would be to study whether there exists a more theoretically motivated approach for solving the overall problem. The crossmining problem in Chapter 6 is theoretically better understood. However, as the problem is close to clustering, instead of greedy selection, could the crossmining model be optimized using a more clustering type of approach, that is, more in the spirit of the well known EM-algorithm?

Bibliography

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data. *Data Mining and Knowledge Discovery*, 11(1):5–33, July 2005.

[2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, New York, NY, USA, 1993. ACM.

[3] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[5] T. Asai, H. Arimura, T. Uno, and S. ichi Nakano. Discovering frequent substructures in large unordered trees. In *The 6th International Conference on Discovery Science*, pages 47–61, 2003.

[6] J. Bascompte, P. Jordano, C. J. Melián, and J. M. Olesen. The nested assembly of plant-animal mutualistic networks. *Proceedings of the National Academy of Sciences*, 100(16):9383–9387, 2003.

[7] R. J. Bayardo, Jr. Efficiently mining long patterns from databases. *SIGMOD Record*, 27(2):85–93, 1998.

[8] B. Bringmann and A. Zimmermann. The chosen few: On identifying valuable patterns. In *IEEE International Conference on Data Mining. ICDM 2007*, pages 63–72, Oct. 2007.

[9] B. Bringmann, A. Zimmermann, L. De Raedt, and S. Nijssen. Don't be afraid of simpler patterns. In *Knowledge Discovery in Databases: PKDD 2006*, pages 55–66, 2006.

[10] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu. Mafia: a maximal frequent itemset algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1490–1504, Nov. 2005.

[11] T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In *Proceedings of the 6th European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 74–85, 2002.

[12] Y. Chi, R. R. Muntz, S. Nijssen, and J. N. Kok. Frequent subtree mining – an overview. *Fundamenta Informaticae*, 66(1-2):161–198, 2005.

[13] Y. Chi, Y. Yang, and R. R. Muntz. HybridTreeMiner: an efficient algorithm for mining frequent rooted trees and free trees using canonical forms. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 11–20, 2004.

[14] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[15] P. Clark and T. Niblett. The cn2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.

[16] G. W. Cobb and Y.-P. Chen. An application of markov chain monte carlo to community ecology. *The American Mathematical Monthly*, 110(4):265–288, 2003.

[17] F. Coenen. The LUCS-KDD discretised/normalised ARM and CARM data library, 2003.

[18] G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[19] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley Series in Telecommunications. Wiley Interscience, 1991.

[20] C. J. Donlan, J. Knowlton, D. F. Doak, and N. Biavaschi. Nested communities, invasive species and holocene extinctions: evaluating the power of a potential conservation tool. *Oecologia*, 145(3):475–485, 2005.

[21] C. Faloutsos and I. Kamel. Beyond uniformity and independence: analysis of r-trees using the concept of fractal dimension. In *PODS '94: Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 4–13, New York, NY, USA, 1994. ACM.

[22] C. Faloutsos and V. Megalooikonomou. On data mining, compression and kolmogorov complexity. *Data Mining and Knowledge Discovery*, 15:3–20, 2007.

[23] J. Felsenstein. *Inferring Phylogenies.* Sinauer Associates, Inc., Sunderland, MA, 2004.

[24] J. H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, 23:881–890, 1974.

[25] S. Fujishige. *Submodular Functions and Optimization*, volume 58 of *Annals of Discrete Mathematics.* Elsevier, 2nd edition, 2005.

[26] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[27] G. C. Garriga, H. Heikinheimo, and J. K. Seppänen. Cross-mining binary and numerical attributes. In *IEEE International Conference on Data Mining (ICDM)*, pages 481–486, 2007.

[28] A. Gionis, T. Kujala, and H. Mannila. Fragments of order. In *KDD '03: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 129–136, New York, NY, USA, 2003. ACM.

[29] 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):14, 2006.

[30] P. D. Grünwald, I. J. Myung, and M. A. Pitt, editors. *Advances in Minimum Description Length: Theory and Applications.* MIT Press, 2005.

[31] 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, August 2007.

[32] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 1–12, New York, NY, USA, 2000. ACM.

[33] D. J. Hand, H. Mannila, and P. Smyth. *Principles of data mining.* MIT Press, Cambridge, MA, USA, 2001.

[34] I. Hanski. Dynamics of regional distribution: the core and satellite species hypothesis. *Oikos*, 32(2):210–221, 1982.

[35] I. Hanski and M. Gyllenberg. Two general metapopulation models and the core-satellite species hypothesis. *The American Naturalist*, 142(1):17–41, 1993.

[36] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer, 2001.

[37] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

[38] H. Heikinheimo, M. Fortelius, J. Eronen, and H. Mannila. Biogeography of European land mammals shows environmentally distinct and spatially coherent clusters. *Journal of Biogeography*, 34(6):1053–1064, 2007.

[39] H. Heikinheimo, E. Hinkkanen, H. Mannila, T. Mielikäinen, and J. K. Seppänen. Finding low-entropy sets and trees from binary data. In *KDD '07: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 350–359, New York, NY, USA, 2007. ACM.

[40] H. Heikinheimo, H. Mannila, and J. K. Seppänen. Finding trees from unordered 0-1 data. In *10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 175–186, 2006.

[41] H. Heikinheimo, J. Vreeken, A. Siebes, and H. Mannila. Low-entropy set selection. In *SIAM International Conference on Data Mining*, pages 569–580, 2009.

[42] R. J. Hijmans, S. E. Cameron, J. L. Parra, P. G. Jones, and A. Jarvis. Very high resolution interpolated climate surfaces for global land areas. *International Journal of Climatology*, 25(15):1965–1978, 2005.

[43] J. Hollmén, J. K. Seppänen, and H. Mannila. Mixture models and frequent sets: combining global and local methods for 0-1 data. In *SIAM International Conference on Data Mining*, pages 289–293, 2003.

[44] P. J. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, June 1985.

[45] J. J. Ibánez, J. Caniego, and A. García-Álvarez. Nested subset analysis and taxa-range size distributions of pedological assemblages: implications for biodiversity studies. *Ecological Modelling*, 182(3–4):239–256, 2005.

[46] A. K. Jain and R. C. Dubes. *Algorithms for clustering data.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[47] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of ACM*, 48(2):274–296, 2001.

[48] S. Jaroszewicz and D. A. Simovici. Pruning redundant association rules using maximum entropy principle. In *PAKDD '02: Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 135–147, London, UK, 2002. Springer-Verlag.

[49] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–251, 1967.

[50] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD '03: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146, New York, NY, USA, 2003. ACM.

[51] J. Kleinberg, C. Papadimitriou, and P. Raghavan. A microeconomic view of data mining. *Data Mining and Knowledge Discovery*, 2(4):311–324, 1998.

[52] J. Kleinberg, C. Papadimitriou, and P. Raghavan. Segmentation problems. *Journal of ACM*, 51(2):263–280, 2004.

[53] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *CIKM '94: Proceedings of the third International Conference on Information and Knowledge Management*, pages 401–407, New York, NY, USA, 1994. ACM.

[54] A. J. Knobbe and E. K. Y. Ho. Maximally informative k-itemsets and their efficient discovery. In *KDD '06: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 237–244, New York, NY, USA, 2006. ACM.

[55] A. J. Knobbe and E. K. Y. Ho. Pattern teams. In *European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 577–584, 2006.

[56] M. Koivisto and K. Sood. Exact bayesian structure discovery in bayesian networks. *Journal of Machine Learning Research*, 5:549–573, 2004.

[57] D. L. Kreher and D. R. Stinson. *Combinatorial Algorithms: Generation, Enumeration and Search.* Discrete mathematics and its applications. CRC Press, 1999.

[58] B. Lent, A. Swami, and J. Widom. Clustering association rules. In *13th International Conference on Data Engineering*, pages 220–231, 1997.

[59] B. Liu, W. Hsu, and Y. Ma. Pruning and summarizing the discovered associations. In *KDD '99: Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 125–134, New York, NY, USA, 1999. ACM.

[60] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[61] M. Mahdian, Y. Ye, and J. Zhang. Approximation algorithms for metric facility location problems. *SIAM Journal of Computing*, 36(2):411–432, 2006.

[62] H. Mannila and E. Terzi. Nestedness and segmented nestedness. In *KDD '07: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 480–489, New York, NY, USA, 2007. ACM.

[63] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *AAAI Workshop on Knowledge Discovery in Databases*, pages 181–192, 1994.

[64] M. Meilă and M. I. Jordan. Learning mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2000.

[65] T. Mielikäinen and H. Mannila. The pattern ordering problem. In *European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 327–338, 2003.

[66] 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.* Poyser, 1999.

[67] S. Morishita and J. Sese. Transversing itemset lattices with statistical metric pruning. In *PODS '00: Proceedings of the 9th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 226–236, New York, NY, USA, 2000. ACM.

[68] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[69] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, 1978.

[70] S. Nijssen. Bayes optimal classification for decision trees. In *ICML '08: Proceedings of the 25th International Conference on Machine Learning*, pages 696–703, New York, NY, USA, 2008. ACM.

[71] S. Nijssen and E. Fromont. Mining optimal decision trees from itemset lattices. In *KDD '07: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 530–539, New York, NY, USA, 2007. ACM.

[72] S. Nijssen and J. N. Kok. Efficient discovery of frequent unordered trees. In *First International Workshop on Mining Graphs, Trees and Sequences (MGST)*, pages 55–64, 2003.

[73] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, 6(1):90–105, 2004.

[74] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. *Lecture Notes in Computer Science*, 1540:398–416, 1999.

[75] B. D. Patterson and W. Atmar. Nested subsets and the structure of insular mammalian faunas and archipelagos. *Biological Journal of the Linnean Society*, 28(1–2):65–82, 1986.

[76] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[77] J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2000.

[78] A. K. Poernomo and V. Gopalkrishnan. Efficient computation of partial-support for mining interesting itemsets. In *SIAM International Conference on Data Mining*, pages 1014–1025, 2009.

[79] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[80] J. K. Seppänen and H. Mannila. Dense itemsets. In *KDD '04: Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 683–688, New York, NY, USA, 2004. ACM.

[81] J. Sese and S. Morishita. Itemset classified clustering. In *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 398–409, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[82] D. B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems (extended abstract). In *STOC '97: Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 265–274, New York, NY, USA, 1997. ACM.

[83] A. Siebes, J. Vreeken, and M. van Leeuwen. Item sets that compress. In *SIAM International Conference on Data Mining*, pages 393–404, 2006.

[84] T. Silander and P. Myllymäki. A simple optimal approach for finding the globally optimal bayesian network structure. In *22nd Annual Conference on Uncertainty in Artificial Intelligence*, pages 445–452, 2006.

[85] N. J. A. Sloane. The on-line encyclopedia of integer sequences, 2006. `http://www.research.att.com/~njas/sequences/`.

[86] N. Tatti and H. Heikinheimo. Decomposable families of itemsets. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 472–487, 2008.

[87] N. Tatti and J. Vreeken. Finding good itemsets by packing data. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM'08)*, 2008.

[88] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Elsevier, second edition, 2003.

[89] A. Tuzhilin and G. Adomavicius. Handling very large numbers of association rules in the analysis of microarray data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 396–404, New York, NY, USA, 2002. ACM.

[90] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl. Constrained k-means clustering with background knowledge. In *ICML '01: Proceedings of the 18th International Conference on Machine Learning*, pages 577–584, 2001.

[91] J. Wang, J. Han, and J. Pei. Closet+: searching for the best strategies for mining frequent closed itemsets. In *KDD '03: Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 236–245, New York, NY, USA, 2003. ACM.

[92] R. Xu and I. Wunsch, D. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, May 2005.

[93] X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing itemset patterns: a profile-based approach. In *KDD '05: Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 314–323, New York, NY, USA, 2005. ACM.

[94] C. Yang, U. Fayyad, and P. S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *KDD '01: Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 194–203, New York, NY, USA, 2001. ACM.

[95] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.

[96] M. J. Zaki. Efficiently mining frequent trees in a forest: algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1021–1035, Aug. 2005.

[97] M. J. Zaki and C.-J. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *SIAM International Conference on Data Mining*, 2002.

[98] B. Zenko, S. Džeroski, and J. Struyf. Learning predictive clustering rules. In *Knowledge Discovery in Inductive Databases, 4th International Workshop, KDID'05, Revised, Selected and Invited Papers*, volume 3933 of *Lecture Notes in Computer Science*, pages 234–250. Springer, 2006.

[99] A. Zimmermann and L. De Raedt. CorClass: Correlated association rule mining for classification. In *International Conference on Discovery Science*, pages 60–72, 2004.