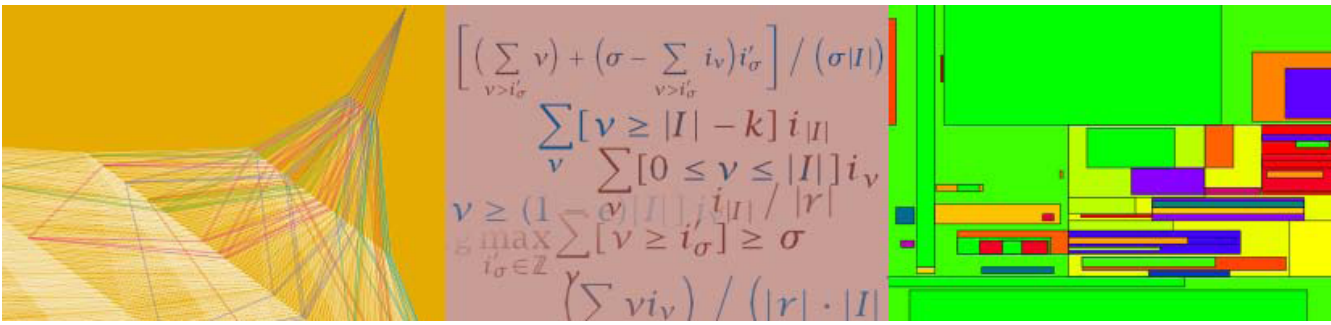
Helsinki University of Technology
Dissertations in Computer and Information Science
Espoo 2006 Report D12

USING AND EXTENDING ITEMSETS IN DATA MINING: QUERY APPROXIMATION, DENSE ITEMSETS, AND TILES

Jouni K. Seppänen



TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

Helsinki University of Technology
Dissertations in Computer and Information Science
Espoo 2006 Report D12

USING AND EXTENDING ITEMSETS IN DATA MINING: QUERY APPROXIMATION, DENSE ITEMSETS, AND TILES

Jouni K. Seppänen

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Computer Science and Engineering for public examination and debate in Auditorium T2 at Helsinki University of Technology (Espoo, Finland) on the 31st of May, 2006, at 12 o'clock noon.

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory of Computer and Information Science

Distribution:
Helsinki University of Technology
Laboratory of Computer and Information Science
P.O. Box 5400
FI-02015 TKK
FINLAND
Tel. +358-9-451 3272
Fax +358-9-451 3277
http://www.cis.hut.fi

Available in PDF format at http://lib.tkk.fi/Diss/2006/isbn951228202X/

ISBN 951-22-8201-1 (printed version)
ISBN 951-22-8202-X (electronic version)
ISSN 1459-7020

Otamedia Oy
Espoo 2006

Seppänen, J. K. (2006): **Using and extending itemsets in data mining: query approximation, dense itemsets, and tiles.** Doctoral thesis, Helsinki University of Technology, Dissertations in Computer and Information Science, Report D12, Espoo, Finland.

Keywords: data mining, frequent itemset, query selectivity, Boolean formulas, Bonferroni inequality, error-tolerant itemset

Abstract

Frequent itemsets are one of the best known concepts in data mining, and there is active research in itemset mining algorithms. An itemset is frequent in a database if its items co-occur in sufficiently many records. This thesis addresses two questions related to frequent itemsets. The first question is raised by a method for approximating logical queries by an inclusion-exclusion sum truncated to the terms corresponding to the frequent itemsets: how good are the approximations thereby obtained? The answer is twofold: in theory, the worst-case bound for the algorithm is very large, and a construction is given that shows the bound to be tight; but in practice, the approximations tend to be much closer to the correct answer than in the worst case. While some other algorithms based on frequent itemsets yield even better approximations, they are not as widely applicable.

The second question concerns extending the definition of frequent itemsets to relax the requirement of perfect co-occurrence: highly correlated items may form an interesting set, even if they never co-occur in a single record. The problem is to formalize this idea in a way that still admits efficient mining algorithms. Two different approaches are used. First, dense itemsets are defined in a manner similar to the usual frequent itemsets and can be found using a modification of the original itemset mining algorithm. Second, tiles are defined in a different way so as to form a model for the whole data, unlike frequent and dense itemsets. A heuristic algorithm based on spectral properties of the data is given and some of its properties are explored.

Tiivistelmä

Yksi tiedon louhinnan tunnetuimmista käsitteistä ovat kattavat joukot, ja niiden etsintäalgoritmeja tutkitaan aktiivisesti. Joukko on tietokannassa kattava, jos sen alkiot esiintyvät yhdessä riittävän monessa tietueessa. Väitöskirjassa käsitellään kahta kattaviin joukkoihin liittyvää kysymystä. Ensimmäinen liittyy algoritmiin, jolla arvioidaan loogisten kyselyjen tuloksia laskemalla inkluusio-ekskluusio-summa pelkästään kattavilla joukoilla; kysymys on, kuinka hyviä arvioita näin saadaan. Väitöskirjassa annetaan kaksi vastausta: Teoriassa algoritmin pahimman tapauksen raja on hyvin suuri, ja vastaesimerkillä osoitetaan, että raja on tiukka. Käytännössä arviot ovat paljon lähempänä oikeaa tulosta kuin teoreettinen raja antaa ymmärtää. Arvioita vertaillaan eräisiin muihin algoritmeihin, joiden tulokset ovat vielä parempia mutta jotka eivät ole yhtä yleisesti sovellettavissa.

Toinen kysymys koskee kattavien joukkojen määritelmän yleistämistä siten, että täydellisen yhteisesiintymisen vaatimuksesta tingitään. Joukko korreloituneita alkioita voi olla kiinnostava, vaikka alkiot eivät koskaan esiintyisi kaikki samassa tietueessa. Ongelma on tämän ajatuksen muuttaminen sellaiseksi määritelmäksi, että tehokkaita louhinta-algoritmeja voidaan käyttää. Väitöskirjassa esitetään kaksi lähestymistapaa. Ensinnäkin tiheät kattavat joukot määritellään samanlaiseen tapaan kuin tavalliset kattavat joukot, ja ne voidaan löytää samantyyppisellä algoritmilla. Toiseksi määritellään laatat, jotka muodostavat koko datalle mallin, toisin kuin kattavat ja tiheät kattavat joukot. Laattojen etsimistä varten kuvataan datan spektraalisiin ominaisuuksiin perustuva heuristiikka, jonka eräitä ominaisuuksia tutkitaan.

Contents

Preface

I have had the privilege of working toward this dissertation at

$$\text{CIS} \cap \text{FDK},$$

where CIS denotes the Laboratory of Computer and Information Science at the Helsinki University of Technology (TKK), and FDK is the From Data to Knowledge research unit at TKK and the University of Helsinki. The organizational chart is further complicated by my affiliation to

$$\text{BRU} \subset \text{HIIT},$$

the Basic Research Unit of the Helsinki Institute for Information Technology. I am grateful to all of these organizations for the use of their facilities. Financial support for this work has been provided by the ComBi graduate school and the Academy of Finland. Personal grants from the Foundation of Technology (TES) are also gratefully acknowledged.

Thanks are due to my advisor Heikki Mannila both for his expert guidance in the field of data mining and for his supportive mentorship. Jaakko Hollmén deserves thanks for the plenty of practical assistance rendered during my years at CIS, as well as for the scientific collaboration that has improved much of the material in this dissertation. The same is true of my other collaborators, including Aris Gionis, Ella Bingham, and Artur Bykowski, to all of whom I am grateful. A manuscript of this dissertation was reviewed by Gautam Das from the University of Texas at Arlington and Bart Goethals from the University of Antwerp, whom I thank for their constructive and encouraging comments. Any errors are, naturally, my own.

In addition to those already mentioned, I thank past and present members of our group at CIS ∩ FDK, including Hannes Heikinheimo, Heli Hiisilä, Johan Himberg, Mikko Korpela, Anne Patrikainen, Kai Puolamäki, Antti Rasinen, Salla Ruosaari, and Janne Toivola, for the wonderful working environment. The coworkers in CIS \ FDK and FDK \ CIS are too numerous to list here, but they too have contributed to the atmosphere.

Above all, I wish to express my gratitude to my wife Elina and my son Otso for their patience during the many long nights I have been working on this dissertation, and for the joy they have brought to my life.

Otaniemi, May 2006

Jouni K. Seppänen

Chapter 1

Introduction

The topic of this dissertation belongs to the research area of data mining. Data mining has been defined by Hand, Mannila and Smyth as

> *the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner.*
>
> [HMS01]

Some elaboration on the definition may be in order. First, the observational nature of the data differentiates data mining from most statistics: in data mining one does not typically have the luxury of setting up a careful experiment yielding data that allows rejecting or accepting a null hypothesis. Instead, the data is observed opportunistically, possibly because it is collected primarily for a purpose other than analysis. In contrast to hypothesis testing, data mining addresses tasks such as

1. exploratory data analysis,
2. descriptive modeling,
3. predictive modeling,
4. discovering patterns and rules, and
5. retrieval by content. [HMS01]

Of course, tools for these tasks are also provided by statistics, machine learning, and database research, and the borders between these areas are not rigid, or indeed clearly drawn at all. All such tools naturally aspire to the goals of understandability and usefulness, but the question whether the goals are attained is to a large degree subjective, and can in the end

only be evaluated in the context of some application. Another aspect of data mining alluded to in the definition is the focus on large data sets: when developing data mining methods, one must keep in mind the practical issues of implementability and scalability.

More specifically, the topic of the dissertation is "using and extending itemsets". The concept of itemsets has its roots in what is perhaps the most famous class of the "patterns and rules" mentioned above, association rules. The problem of association rule mining, which dates back to at least 1993, is to find all rules such as

a customer who buys milk and cookies also buys coffee

that concern a sufficiently large portion of the data and hold with a sufficient confidence [AIS93]. The way to solve this problem is to reduce it to finding frequent itemsets, i.e., sets such as

{ *milk*, *cookies*, *coffee* }

of items that co-occur in a sufficiently large part of the data set. The original algorithm to find frequent itemsets is an elegant application of breadth-first search called Apriori [AMS+96]. Testimony to the usefulness, real or perceived, of association rules is that research into more advanced algorithms continues [GZ03, BGZ04]; however, some have argued that the gains of the new algorithms are only evident in artificial settings and do not translate into practical advantages [ZKM01].

Association rules were the original motivation for frequent itemset mining, but of course a frequent itemset is a pattern in itself, and perhaps even better by virtue of not specifying a direction that could be misinterpreted as causality. The descriptive models and patterns in the preceding list of data mining tasks are distinguished by the local nature of patterns, which only pertain to some part of the data, while a descriptive model should pertain to the whole. A natural question, then, is whether a collection of local patterns can constitute a global description. The seminal paper *Multiple uses of frequent itemsets and condensed representations* by Mannila and Toivonen [MT96] approaches this question by viewing the itemsets as a surrogate for the data, which can be used to answer queries approximately.

This may seem a convoluted way of answering queries—why add the extra phase of itemset mining to obtain approximations instead of looking at the data itself for exact results? The obvious answer is that the frequent itemset collection may be much smaller than the original data, but even then, a sample of the data could be a much better surrogate than

itemsets. However, there may be some practical circumstances where using itemsets is warranted. Sometimes there may not be an explicit data set that can be queried. For instance, in mining event sequences [MTV97] or more complicated structures such as graphs [KK01], it may be practical to find the itemsets directly without constructing an intermediate data matrix. Another possibility arises in privacy-preserving data mining: the owner of the data may only allow access to frequent itemsets and not to the original data.

The cases mentioned can be seen as manifestations of the observational quality of the data, a recurring theme in data mining. A more abstract motivation is that of building a theory of frequent itemsets. Parts of this theory exist: there is a connection to hypergraph transversals, which implies some results on the complexity of itemset mining [BGKM02, GKM+03]. Another part of the theory are various condensed representations such as free-sets [BBR00, CG03], closed sets [PBTL99] and non-derivable sets [CG02, Cal04], which allow a data mining system to store only some of the frequent itemsets and infer the others when needed. The query approximation problem is one formulation of the question of what information the itemsets contain about the original data.

A related research area is that of inductive databases, which had its beginnings in the 1996 paper *A database perspective on knowledge discovery* by Imielinski and Mannila [IM96] and is under active research in e.g. the KDID workshops [Bou04]. The aim of inductive database research is to enable general-purpose systems to support data mining queries analogously to the way that current relational database management systems support SQL queries: ad-hoc queries can be entered by the user, or they can be part of an application, and it is the purpose of the database management system to optimize and answer the query. A comprehensive theory of itemsets should allow the system to choose an optimal way to answer itemset-related queries.

The truncated inclusion-exclusion algorithm of Mannila and Toivonen is analyzed in Chapter 3. The algorithm is shown to have bad worst-case behavior in that there is a large bound for the error that is actually achievable. However, experiments with both real and generated data sets show that the error bound does not reflect typical behavior of the algorithm. The algorithm is also generalized to arbitrary Boolean queries from the disjunctions originally presented by Mannila and Toivonen. In addition to the results of the analysis, the methods used could be of relevance in the theory of inductive databases.

Another question addressed in this dissertation is how to extend the definition of frequent itemsets to allow for incomplete occurrence of the items. Intuitively, the requirement that all items of a set must co-occur for the set to be frequent may exclude interesting sets whose items are strongly correlated but never co-occur perfectly. It is not obvious how to transform this intuition to a useful mathematical definition. The problems that arise are discussed in Chapter 4, and a definition is given for dense itemsets. Dense itemsets can be found by a simple generalization of the APRIORI algorithm, but they can be very different from frequent itemsets.

Chapter 5 presents a different solution to finding highly but non-perfectly correlated itemsets. This solution, termed tile mining, forms a model for the complete data from local patterns, unlike frequent or dense itemsets. Furthermore, it encompasses negative patterns, i.e., sets of items that are together absent from a record more often than other items. "Geometric" tiles are usable when the data has a natural order, but this is not usually the case. "Combinatorial" tiles are more general, but also much more difficult to find. A heuristic algorithm is presented that seeks an order for the data such that combinatorial tiles become geometric.

In summary, the dissertation addresses questions related to discovering local patterns, to descriptive modeling, and to combining these two tasks. While experiments are used to demonstrate the methods, the main thrust of this work is theoretical: Chapter 3 aims to advance the theory of frequent itemsets, and Chapters 4 and 5 seek to discover new data-mining concepts in addition to frequent itemsets. Possible future applications for the methods could be found in e.g. the analysis of biological data sets. For example, microarray experiments yield high volumes of gene expression data, which could be mined for patterns such as dense itemsets or tiles.

Contributions of the thesis. While the algorithm for query estimation is by Mannila and Toivonen [MT96], the theoretical analysis and the experiments are a new contribution. The main results of the theoretical analysis are Theorems 3.16 and 3.18, which constitute a complete answer to the question of how large the worst-case approximation error can be in terms of the negative border. Furthermore, the paper [MT96] discussed only the case where the queries are disjunctions of attributes; Section 3.6 generalizes it to arbitrary Boolean queries. This generalization is based on the inner product representation of Section 3.2, which allows writing the frequency of an arbitrary Boolean formula as a sum of itemset frequencies with integer coefficients. Such a representation was mentioned in the

paper as "of course well known", but no way to compute it was given. This author has not found as complete a description as in Section 3.2 of it in the literature, although the representation is (at least implicitly) used in various papers that use the theory of linear programming to attack similar problems; a description and references are given in Section 3.9.

The definition of dense itemsets along with the discussion and experiments on them are a novel contribution, although similar ideas have been published in the past. These ideas are discussed along with the exposition of dense itemsets in Chapter 4, and connections to other work are made in Section 4.5. The first algorithm for finding dense itemsets is an obvious generalization of APRIORI, but a slightly more complicated algorithm that avoids the need to specify a parameter is new. Experiments show that the algorithms are practical and that the results are interesting itemsets that could not have been found with traditional frequent itemset mining.

The tile models are also novel. The algorithm for finding an order for the data matrix is a fairly straightforward application of spectral graph theory, but the analysis in the case of tiles is new. See Section 5.4 for related work.

Relationship to other publications of the author. Chapter 3 has some overlap with the author's Master's thesis, which addressed similar questions. A weaker form of Theorem 3.18 was shown, and the inner product representation of Section 3.2 was developed. A paper corresponding to the theoretical parts of Chapter 3 has been published in a book [SM05]. There is some thematic overlap with a paper published originally in a workshop on inductive databases [BSH02] and then, revised, in a book [BSH04]; that paper uses linear programming, which Chapter 3 only mentions. A paper on combining mixture models and frequent itemsets [HSM03] is somewhat related thematically but not overlapping in content: frequent itemsets were mined in clusters discovered by a mixture modeling technique and combined into a new model of the data.

The dense itemsets of Chapter 4 were originally introduced in a conference poster with a related publication [SM04]. Chapter 4 is much expanded from the publication. The tile models of Chapter 5 were published in another conference [GMS04]. Again, Chapter 5 is a more extensive treatment. Tangentially related to these chapters are two papers on topic models [BMS02, SBM03], which discuss a probabilistic approach to defining sets of correlated variables, and give some heuristic algorithms. The maximum-entropy model, used as a reference in Chapter 3, also had a minor role in the first of the two papers.

Chapter 2

Preliminaries

This chapter presents a more formal view of the concepts mentioned in the previous chapter and introduces some notation. In particular, we review the basics of frequent itemset mining in preparation for Chapters 3 and 4. These chapters also share their example data sets, which are described here. While Chapter 5 discusses a problem similar to that of Chapter 4, the context and notation are different and thus presented within the chapter itself.

Binary data. In this dissertation, only binary data is considered. For example, the data set could record which customers bought which items in a supermarket, or which documents used which words of a dictionary, or which genes were expressed in which conditions, or which users visited which web pages. What is not recorded is numerical information such as the number of milk cartons bought by a customer, or the number of times a word is used in a document. Also ignored is any external information such as the prices of items or the language of words; such information could naturally be useful in some data mining tasks, but it is not used in frequent itemset mining.

The restriction to binary data allows us to adopt a simplified definition of a relational database. We will denote a set of *attributes* by U (for "Universe"); these are the items sold by the supermarket or the words in the dictionary. Each customer or document is represented by a subset T of U, called a *tuple*. The tuples form a *binary relation* r over U; thus a relation is a multiset[1] of tuples.

[1] We must allow different customers to buy the same set of products; therefore the relation is a multiset instead of a set. Another way to allow this would be to say that each tuple carries with it a unique identifier.

Milk	Coffee	Cookies	Diapers
1	1	0	0
0	1	1	0
0	1	1	0
0	0	0	1
1	0	0	1

Table 2.1. Example data matrix. Columns correspond to items available in a grocery store, rows to customers. An entry of 1 means that the customer bought the item, an entry of 0 that they did not.

Binary data, as described, can be seen as a matrix of zeros and ones as pictured in Table 2.1. This view may be misleading in at least two ways. First, it suggests that the rows and columns are arranged in some meaningful order, which may be true for some data, but we make no such assumption. Second, the numbers 0 and 1 could be seen as symmetric: subtracting each element of the matrix from 1 creates another matrix that seems in a way equivalent to the original one. However, the meanings of 0 and 1 are not symmetric: if 1 means that an item was included in a basket or that a word was used in a document, inverting the values would give for each customer the list of items she did not buy, or for each document the list of dictionary words it did not use. Usually, the latter lists would be much longer than the former ones. Note that in Chapter 5 we do adopt the view of a data matrix, with symmetric interpretations of 0 and 1.

Itemsets. Arbitrary subsets of U are called *itemsets* (even if their contents are words or genes). We denote items by letters $A, B, C, \ldots$ from the beginning of the alphabet, and itemsets by the letters X, Y and Z. In Chapter 4 we occasionally use also I and J for itemsets. An itemset is *frequent* if it appears as a subset of sufficiently many tuples in r. Put more precisely, the frequency of $X \subset U$ is

$$f(X) = \frac{\left|\{\, T \in r \mid X \subset T \,\}\right|}{|r|},$$

where the cardinality notation $|\cdot|$ refers to multiset cardinality, and an itemset X is frequent if $f(X) \geq \sigma$, where σ is a predefined *frequency threshold*.

The collection of σ-frequent itemsets is denoted $\mathcal{F}_\sigma$. As is well known, this collection is *downward closed*: given itemsets $X \subset Y \in \mathcal{F}_\sigma$, we have $X \in \mathcal{F}_\sigma$. We will in general denote by $\mathcal{F}$ an arbitrary downward closed collection of itemsets, and by $\mathcal{G}$ its complement.

APRIORI(U, r, σ)
Input: Attribute set U, relation r, threshold σ
Output: Family $\mathcal{F}$ of σ-frequent itemsets

$\mathcal{F} \leftarrow \{ \emptyset \}$
$C \leftarrow \{ \{ A \} \mid A \in U \}$
while $C \neq \emptyset$:
    $F \leftarrow$ COMPUTE-FREQUENCIES(C, r)
    $\mathcal{D} \leftarrow \{ X \in C \mid F[X] \geq \sigma \}$
    $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{D}$
    $C \leftarrow$ GENERATE-CANDIDATES$(\mathcal{D})$
return $\mathcal{F}$

Algorithm 2.1. Mine frequent itemsets (Apriori).

COMPUTE-FREQUENCIES(C, r)
Input: Candidate family C, relation r
Output: Array F containing frequency of each candidate set in C

$F[X] \leftarrow 0 \quad \forall X \in C$
for each $T \in r$:
    for each $X \in C$:
        if $T \subset X$:
            $F[X] \leftarrow F[X] + 1/|r|$
return F

Algorithm 2.2. Compute frequencies of all candidates (subroutine of Apriori).

It will also be useful to talk about the number of times that an itemset appears in the relation without any other items. We will call this quantity the *exact frequency* of X and denote it by $g(X)$; thus,

$$g(X) = \frac{\left| \{ T \in r \mid X = T \} \right|}{|r|},$$

and

$$f(X) = \sum_{Y \supset X} g(X).$$

GENERATE-CANDIDATES($\mathcal{D}$)
Input: Family $\mathcal{D}$ of itemsets of equal size k
Output: Family C of size $(k+1)$ itemsets whose all size k subsets are in $\mathcal{D}$

```
C ← ∅
for each X, Y ∈ D:
    Z ← X ∪ Y
    ok ← True
    for each A ∈ Z:
        if Z \ { A } ∉ D:
            ok ← False
            break
    if ok:
        C ← C ∪ { Z }
return C
```

Algorithm 2.3. Generate next level of candidates (subroutine of Apriori).

Apriori algorithm. The traditional way to find frequent itemsets is by the APRIORI algorithm, described as pseudocode in Algorithm 2.1. The algorithm performs a level-by-level search starting from the single-item sets and proceeding to larger sets. The variable C holds the family of candidate itemsets, whose frequencies are counted in one database pass by Algorithm 2.2. The frequent candidates are selected into $\mathcal{D}$, which is the input to Algorithm 2.3. This algorithm computes the next level of candidates: if the sets in $\mathcal{D}$ have size k, the new candidates are those sets of size $k+1$ whose all size-k subsets are in $\mathcal{D}$. The version shown is easy to understand, but it wastes some time in trying all pairs of sets. It is possible to reduce the number of pairings significantly [AMS+96].

Note that the APRIORI algorithm examines not only all frequent itemsets but also those non-frequent itemsets whose all subsets are frequent. We will define the family of these sets the *negative border* of the frequent itemset collection, and denote it by Bd^-.

Data used in experiments. The experiments presented in Chapters 3 and 4 have been performed on a number of data sets, which are summarized in Table 2.2. The *Course* data consists of course registration records at the Department of Computer Science, University of Helsinki. The attributes correspond to courses and the tuples to students; the presence of an attribute means that the student registered for the course.

Data set name	Attributes	Tuples	Density
Course	98	3506	0.0468
Abstracts	25335	128804	0.0032
T10I4D100K	870	100000	0.0116
T40I10D100K	942	100000	0.0420
Chess	75	3196	0.4933
Connect	129	67557	0.3333
Kosarak	41269	99002	0.0020
Mushroom	119	8124	0.1933
Pumsb	2113	49046	0.0350
*Pumsb**	2088	49046	0.0242

Table 2.2. Statistics on the data sets. The first column gives the name of each data set, the second and third its dimensions, i.e., how many attributes and how many tuples it has, and the final column gives the average proportion of attributes contained in a tuple, i.e., the mean of all values in the 0–1 matrix representation of the data.

The primary reason for using this data set here is that the author, having studied at the Department, has a subjective understanding of the meanings of many of the attributes.

The *Abstracts* data from the UCI KDD repository [HBM98] is a bag-of-words representation of abstracts of National Science Foundation awards for basic research. The attributes correspond to words indexed in the abstracts.

The remaining data sets were downloaded from the web site of FIMI, the Frequent Itemsets Mining Implementations workshop.[2] They have been used as benchmarks in the Frequent Itemset Mining Implementations workshops [GZ03, BGZ04]. The preprocessed data sets do not contain information on the semantics of the data. According to the FIMI web site, the *T10I4D100K* and *T40I10D100K* data sets have been generated with the IBM Almaden generator; *Chess*, *Connect*, and *Mushroom* originate in the UCI KDD repository [HBM98]; and *Kosarak* is anonymized clickstream data. The *Pumsb* and *Pumsb** data sets appear to be derived from the Public-use Microdata Sample provided by the Bureau of the Census of the United States government.

Frequent itemsets were mined from each data set. Different values of the threshold σ had to be used to obtain a reasonable number of itemsets. This is depicted in Figure 2.1, which shows the number of frequent itemsets in the data sets at various thresholds.

[2] `http://fimi.cs.helsinki.fi/data/`

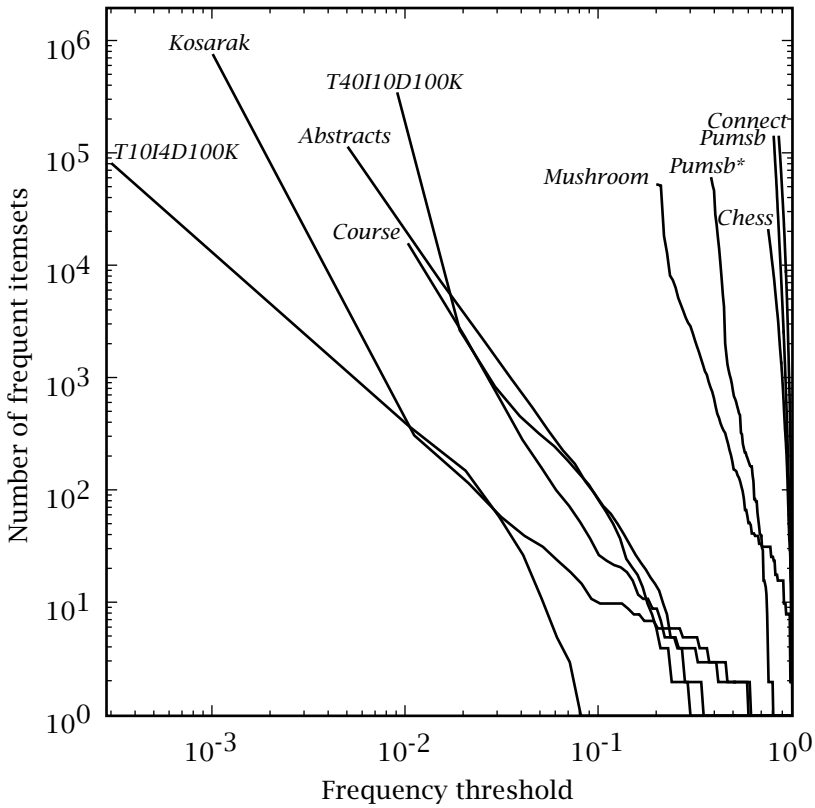


Figure 2.1. Number of frequent itemsets as a function of the frequency threshold in each data set. Both axes are logarithmically scaled.

Notation. To close this chapter, we list some deviations from standard mathematical notation that are used later. First, when discussing itemsets such as $\{A, B, C\}$, the braces and commas are usually omitted: thus ABC means the same itemset. Second, the *shadow* ∂X of an itemset X is defined as the family of size $|X| - 1$ subsets of X. Third, we occasionally need to refer to the set $\{1, 2, \ldots, n\}$, which we denote by $[n]$. Fourth, we also use brackets for the "Iverson notation"

$$[\text{predicate}] = \begin{cases} 1, & \text{predicate is true} \\ 0, & \text{predicate is false,} \end{cases}$$

which helps avoid complicated subscripts in sums. It has been advocated for this purpose by Knuth [Knu92]. As an example of the Iverson notation, consider the sum

$$\sum_{\substack{i=0 \\ i \notin X}}^{n} i^2 = \sum_{i=0}^{n} [i \notin X]\, i^2.$$

In the right-hand form, the second subscript has been moved to the baseline in order to improve readability. We will not take the notation to the extreme of

$$\sum_{i} [i \in \mathbb{Z} \setminus X]\, [0 \le i \le n]\, i^2,$$

but will use it when it helps avoiding excessive or complicated subscripts, mainly in sums taken over sets: for example, in

$$\sum_{X} [|X| \le 5] \cdots$$

the sum is taken over all sets X of size at most 5 that are subsets of some base set, which should be clear from the context.

Chapter 3

Approximating Boolean queries using frequent itemsets

In this chapter we consider the problem of how one can estimate the support of Boolean queries given a collection of frequent itemsets. We describe an algorithm that truncates the inclusion-exclusion sum to include only the frequencies of known itemsets, and give bounds for its performance in two cases: disjunctions of attributes, and general queries. In the disjunction case we show that our bound is in fact achievable.

3.1 The support estimation problem

The task considered in this chapter is that of estimating the result of a *Boolean query* ϕ. We define the syntax of Boolean queries (or formulas, or polynomials) as it is defined in logic: a Boolean query ϕ is one of the following:

- an item: $\phi = A$ for some $A \in U$,
- the negation of a Boolean query: $\phi = (\neg\psi)$ for some ψ,
- the conjunction of two Boolean queries: $\phi = (\psi \wedge \theta)$ for some ψ and θ, or
- the disjunction of two Boolean queries: $\phi = (\psi \vee \theta)$ for some ψ and θ.

We use standard shorthand notation; for example, for a set X, we denote by $\bigvee X$ the disjunction over all the elements of X.

The semantics of a Boolean query are defined as fractions of the size of the relation. We interpret tuples as truth assignments, such that a tuple T *supports* ϕ, denoted $T \vDash \phi$, as follows:

- $T \vDash A$ if and only if $A \in T$,
- $T \vDash (\neg\psi)$ if and only if $T \nvDash \psi$,
- $T \vDash (\psi \wedge \theta)$ if and only if $T \vDash \psi$ and $T \vDash \theta$, and
- $T \vDash (\psi \vee \theta)$ if and only if $T \vDash \psi$ or $T \vDash \theta$.

Now the frequency $f(\phi)$ of a query ϕ is the fraction of tuples that support the query:

$$f(\phi) = \frac{\sum_T [T \in r]\,[T \vDash \phi]}{|r|}.$$

An immediate consequence of the definitions is that the frequency of an itemset X coincides with the frequency of the conjunction of all items in X. This justifies using the same notation $f(\cdot)$ for the two concepts. In contrast, there is no simple generalization of exact frequency $g(X)$ for general formulas.

The problem, then, is to find approximations to $f(\phi)$ for some class of queries ϕ, given the collection $\mathcal{F}_\sigma$ of σ-frequent itemsets. We formalize the problem as follows:

Definition 3.1. *Given the collection $\mathcal{F}_\sigma$ of σ-frequent itemsets, their frequencies $f(X)$ for all $X \in \mathcal{F}_\sigma$, and a Boolean query ϕ, the problem* Approximate Query$(\mathcal{F}_\sigma, f(\cdot), \phi)$ *is to find an estimate $\hat{f}(\phi)$ that should be close to $f(\phi)$. The* error *of such an estimate is defined as $e(\phi) = f(\phi) - \hat{f}(\phi)$.*

The error as defined above depends both on the particular data set and on the input: in particular, as σ decreases, the family $\mathcal{F}_\sigma$ will contain more sets, and one expects the absolute error to decrease. We are interested in bounding the absolute value of the error over a wide variety of data sets and inputs.

An obvious algorithm for query approximation without using frequent itemsets at all is to evaluate the query on the relation r, or if r is too large, on a random sample thereof. However, the motivation behind the Approximate Query problem is to make the best use of frequent itemsets, in the spirit of data mining as secondary data-analysis: by mining frequent itemsets, one hopes to gain some information about the data. In some circumstances, the original data may not be available because of limitations in storage capacity, or a privacy policy may call for deleting the raw data and retaining only aggregate information such as frequent itemsets. Furthermore, the question is relevant to the development of a theory of data mining. Since the inception of association rule mining

in 1993 [AIS93], techniques for computing frequent sets have formed a significant part of the published data mining research, and continue to be a popular research subject [GZ03, BGZ04]. Much of the theory encompassing frequent itemsets has concentrated on speeding up the algorithms, but a theory of frequent itemsets should also consider the uses that the itemsets can be put to.

The algorithm that we analyze in this chapter is combinatorial in nature, and reasonably fast (relative to the size of the input). This is in contrast to the model-based algorithms widely used in machine learning. For the APPROXIMATE QUERY problem, probably the most fashionable model is the Markov random field, or maximum entropy [PMS00, PS01]. The scope of this chapter is limited to our combinatorial algorithm, but we give a comparison to maximum entropy in Sections 3.5 and 3.7.

In the next section we develop some mathematical machinery to handle the problem, and in Section 3.3 we present an algorithm, which we proceed to analyze in the subsequent sections.

3.2 Inner product representation

The frequency of a query ϕ in a relation r can be represented as an inner product of two vectors, one depending only on the query, the other only on the relation. To see this, start from the disjunctive normal form (DNF): write the query as a disjunction of conjunctions,

$$\phi = \bigvee_i \bigwedge_j L_{ij},$$

where each literal L_{ij} is either an item or a negated item. Then rewrite the DNF query by the following rule: if a conjunction $\bigwedge_j L_{ij}$ does not include an item A in any literal, replace it by the two conjunctions $(\bigwedge_j L_{ij}) \wedge A$ and $(\bigwedge_j L_{ij}) \wedge \neg A$. The rewriting does not change the semantics of the query. When no more rewriting is possible, each conjunction includes every item, either positively or negatively: for a fixed value of i,

$$\bigwedge_j L_{ij} = \bigwedge_{A \in P_i} A \wedge \bigwedge_{A \in N_i} \neg A$$

with $P_i \cup N_i = U$. But this shows that $f(\bigwedge_j L_{ij})$ counts the number of tuples T that are equal to P_i, i.e., $f(\bigwedge_j L_{ij}) = g(P_i)$. Delete any repetitions of the same conjunction; then

$$f(\phi) = \sum_i g(P_i),$$

because no tuple can support more than one of the conjunctions. For the ith conjunction to be included in the sum, it is necessary and sufficient that $P_i \vDash \phi$. Therefore, we can rewrite the sum as a sum over all possible itemsets:

$$f(\phi) = \sum_{X \subset U} [X \vDash \phi]\, g(X). \tag{3.1}$$

Interpreting this sum as a dot product of two vectors, we make the following definition.

Definition 3.2. *Impose some canonical order on the subsets of U and define vectors $\vec{g} = (\, g(X) \mid X \subset U \,)$ and $\vec{\zeta_\phi} = (\, [X \vDash \phi] \mid X \subset U \,)$ whose components are ordered by the canonical order of $X \subset U$; we call the components of $\vec{\zeta_\phi}$, denoted $\zeta_\phi(X)$, the* exact-frequency coefficients *of ϕ.*

The term "exact-frequency coefficient" is meant to convey that the numbers ζ_ϕ function as coefficients of the exact frequencies; while they form a vector of the same length as $\vec{g}$, they are not frequencies themselves. Specifically, the coefficients are always either 0 or 1. We can now rewrite (3.1) as follows.

Proposition 3.3. $f(\phi) = \vec{\zeta_\phi} \cdot \vec{g}$.

As pointed out above, the coefficients are either 0 or 1, Also the converse holds: any binary vector of length $2^{|U|}$ is a possible vector $\vec{\zeta_\phi}$ for some Boolean formula ϕ. The exact frequencies $g(X)$ must obviously sum up to 1, but they are otherwise independent. This independence prevents us from proving inequalities for cases where only some of the exact frequencies are known. To solve this problem, we use linear algebra: the subsets X of U are identified with the basis vectors of a $2^{|U|}$-dimensional vector space. Boolean formulas correspond bijectively via their DNF representations to vertices $\vec{\zeta}$ of the unit hypercube, and binary relations correspond to (rational) points $\vec{g}$ in the simplex spanned by the basis vectors. We next change the basis of the space to introduce dependencies between the coordinates of vectors.

Definition 3.4. *The* frequency coefficients *of a query ϕ, denoted $\xi_\phi(X)$, are defined via the following alternating sum of exact-frequency coefficients:*

$$\xi_\phi(X) = \sum_{Y \subset X} (-1)^{|X \setminus Y|} \zeta_\phi(Y). \tag{3.2}$$

We denote by $\vec{\xi_\phi}$ the vector of frequency coefficients in the canonical order of subsets of U.

For example, consider the simple query $\phi = A \wedge B$. The exact-frequency coefficients of this query are obviously $\zeta_\phi(Y) = 1$ if Y includes A and B, and 0 otherwise. For the set AB, the alternating sum (3.2) yields 1, and for any other set, it yields 0; and of course the frequency of ϕ is exactly $f(AB)$. The following proposition shows that the new coefficients can be similarly used for any query.

Proposition 3.5. *The frequency of ϕ can be represented as the inner product $f(\phi) = \vec{\xi_\phi} \cdot \vec{f}$, where $\vec{\xi_\phi}$ is the vector of frequency coefficients as defined above and $\vec{f}$ is the vector of frequencies.*

Proof. By the inclusion-exclusion principle, we have

$$g(X) = \sum_{Y \supset X} (-1)^{|Y \setminus X|} f(Y), \tag{3.3}$$

or in matrix notation, $\vec{g} = B\vec{f}$, where

$$B = (\, b_{X,Y} \mid X, Y \subset U \,), \quad b_{X,Y} = [X \subset Y]\,(-1)^{|Y \setminus X|}.$$

Substituting this into Proposition 3.3 and applying the associativity of matrix multiplication yields

$$f(\phi) = \vec{\zeta_\phi} \cdot \vec{g} = \vec{\zeta_\phi}^{\,T}(B\vec{f}) = (B^T\vec{\zeta_\phi})^T f = \vec{\xi_\phi} \cdot \vec{f}. \qquad \square$$

From Eq. (3.2), one easily sees that $\xi_\phi(X)$ is largest for parity-type formulas ϕ. For example, let $Y \vDash \phi$ if and only if $|X \setminus Y|$ is even; then $\xi_\phi(X) = 2^{|X|-1}$. Clearly, this is the largest absolute value possible, because every coefficient $\zeta_\phi(X)$ must be either 0 or 1. This answers Problem 9 in [Man02].

While the exact-frequency coefficients ζ_ϕ can be either 0 or 1 independently of each other, the frequency coefficients ξ_ϕ are more constrained, and it may not be immediately obvious what value combinations are possible for the coefficients. We can answer this question by attempting to reconstruct the exact-frequency coefficients, as follows.

Proposition 3.6. *Given a vector $\vec{\xi}$ of length $2^{|U|}$, there exists a Boolean formula ϕ such that $\vec{\xi} = \vec{\xi_\phi}$ if and only if the sum*

$$\sum_Y [Y \subset X]\, \xi(Y)$$

is either 0 or 1 for every $X \subset U$.

Proof. Necessity follows by representing $\vec{\zeta_\phi}$ in terms of $\vec{\xi_\phi}$. We start from the representation of frequencies as sums of exact frequencies,

$$f(X) = \sum_{Y \supset X} g(Y),$$

which is also a matrix product: $\vec{f} = C\vec{g}$ with $C = (\, c_{X,Y} \,)$, $c_{X,Y} = [X \subset Y]$. In the proof of Proposition 3.5 a matrix B was introduced such that $\vec{g} = B\vec{f}$ for any vector $\vec{g}$ of exact frequencies; because $\vec{g} = BC\vec{g}$, C must be the inverse of B. Since $\vec{\xi_\phi} = B^T\vec{\zeta_\phi}$, we have $\vec{\zeta_\phi} = C^T\vec{\xi_\phi}$, which in sum form is

$$\zeta_\phi(X) = \sum_Y [Y \subset X]\, \xi_\phi(Y). \tag{3.4}$$

Since $\zeta_\phi(X)$ is either 0 or 1, we have shown that the condition is necessary. To show sufficiency, we construct a Boolean formula ϕ from the vector $\vec{\xi}$. For any tuple $T \subset U$, let $T \vDash \phi$ if and only if $\sum_Y [Y \subset T]\, \xi(T) = 1$. This determines the exact-frequency coefficients of ϕ, and applying Proposition 3.5 shows that the frequency coefficients ξ_ϕ coincide with ξ. □

As a byproduct of the proof, we obtained Eq. (3.4). The following special case of this equation is useful later.

Corollary 3.7. *For any Boolean formula ϕ, the sum of all its frequency coefficients is $[U \vDash \phi]$, i.e.,*

$$\sum_{X \subset U} \xi_\phi(X) = \zeta_\phi(U) = [U \vDash \phi].$$

We next discuss a question related to using the inner product representation: if we are given a formula such as $(A \vee B) \wedge \neg(B \vee C)$, we can obviously obtain its frequency coefficients by first computing the exact-frequency coefficients and then summing, but is there a more direct way? Our definition of the frequency coefficients is semantic in that it depends only on the truth values of ϕ on various itemsets. It is also possible to describe the coefficients syntactically, recursing on the structure of the formula.

Proposition 3.8. *The frequency coefficients of a Boolean query ϕ can be expressed as follows.*

1. *If $\phi = A$ for $A \in U$, then $\xi_\phi(X) = [X = \{A\}]$.*
2. *If $\phi = \neg\psi$ for a Boolean query ψ, then*

$$\xi_\phi(X) = \begin{cases} 1 - \xi_\psi(\varnothing), & X = \varnothing, \\ -\xi_\psi(X), & X \neq \varnothing. \end{cases}$$

3. If $\phi = \psi \wedge \theta$, then

$$\xi_\phi(X) = \sum_{W,Z} [W \cup Z = X]\, \xi_\psi(W)\xi_\theta(Z). \tag{3.5}$$

Proof. 1. This case follows directly from Definition 3.4.

2. If ϕ is the negation of ψ, we trivially have $\zeta_\phi(X) = 1 - \zeta_\psi(X)$, and by substituting into Definition 3.4 obtain the claim.

3. If ϕ is the conjunction of ψ and θ, we have for the exact-frequency coefficients that $\zeta_\phi(X) = \zeta_\psi(X)\zeta_\theta(X)$. Substituting this into Definition 3.4, we obtain

$$\xi_\phi(X) = \sum_{Y \subset X} (-1)^{|X \setminus Y|} \zeta_\psi(Y)\zeta_\theta(Y). \tag{3.6}$$

From Equation (3.4), we obtain

$$\zeta_\psi(Y)\zeta_\theta(Y) = \Big(\sum_{W \subset Y} \xi_\psi(W)\Big)\Big(\sum_{Z \subset Y} \xi_\theta(Z)\Big) = \sum_{W,Z \subset Y} \xi_\psi(W)\xi_\theta(Z). \tag{3.7}$$

Combining (3.6) and (3.7) and changing the order of summation now yields

$$\xi_\phi(X) = \sum_{W,Z \subset X} \Big(\sum_{Y} [W \cup Z \subset Y \subset X]\, (-1)^{|X \setminus Y|}\Big)\, \xi_\psi(W)\xi_\theta(Z).$$

The alternating sum in parentheses cancels out whenever $W \cup Z \neq X$. If $W \cup Z = X$, the sum is equal to 1. This proves Equation (3.5). □

We illustrate the syntactic characterization of frequency coefficients by two examples.

Example 3.9. We construct the frequency representation of $(A \vee B) \wedge \neg(B \vee C)$. We start by transforming the disjunctions by de Morgan's laws: $A \vee B \equiv \neg(\neg A \wedge \neg B)$ and $B \vee C \equiv \neg(\neg B \wedge \neg C)$. From the base case and the rule for negation, we obtain

$$f(\neg A) = f(\varnothing) - f(A),$$

and similarly for the other attributes.

For the sake of convenience, we express the frequencies of the queries as polynomials in the frequencies of itemsets: in the notation of Proposition 3.8, the formula above would be written as

$$\xi_{\neg A}(X) = \begin{cases} 1, & X = \varnothing, \\ -1, & X = \{A\}, \\ 0, & \text{otherwise.} \end{cases}$$

In the polynomial notation, Equation (3.5) can be written as a special kind of multiplication: define the product $\otimes$ for single terms as

$$\big(\xi_\psi(W)f(W)\big) \otimes \big(\xi_\theta(Z)f(Z)\big) = \big(\xi_\psi(W)\xi_\theta(Z)\big)f(W \cup Z)$$

and extend it for sums of such terms so that the distributive law holds. Then Equation (3.5) has the form

$$f(\psi \wedge \theta) = \sum_X \xi_{\psi\wedge\theta}(X)f(X) = \Big(\sum_Y \xi_\psi(Y)f(Y)\Big) \otimes \Big(\sum_Z \xi_\theta(Z)f(Z)\Big).$$

Returning to the example, we use the $\otimes$ product to obtain

$$\begin{aligned} f(\neg A \wedge \neg B) &= \big(f(\varnothing) - f(A)\big) \otimes \big(f(\varnothing) - f(B)\big) \\ &= f(\varnothing) - f(A) - f(B) + f(AB). \end{aligned}$$

The rule for negation yields

$$f(A \vee B) = f(\neg(\neg A \wedge \neg B)) = f(A) + f(B) - f(AB).$$

Similarly we obtain

$$f(\neg(B \vee C)) = f(\varnothing) - f(B) - f(C) + f(BC).$$

Putting these results together using the $\otimes$ product yields, after simplification, the goal:

$$f((A \vee B) \wedge \neg(B \vee C)) = f(A) - f(AB) - f(AC) + f(ABC).$$

Example 3.10. If the query is a disjunction of attributes, $\phi = \bigvee D$, we can transform it by de Morgan's laws to

$$\bigvee D \equiv \neg\big(\bigwedge_{A\in D} \neg A\big)$$

and apply Eq. (3.5) to obtain

$$f\big(\bigvee D\big) = f(\varnothing) - \bigotimes_{A\in D}\big(f(\varnothing) - f(A)\big) = \sum_X [\varnothing \neq X \subset D]\,(-1)^{|X|+1} f(X).$$

This is, of course, the inclusion-exclusion principle.

In this section we have shown how one can exactly evaluate arbitrary Boolean queries using the frequencies of itemsets: by Proposition 3.5, the frequency of the query is representable as an inner product between a vector depending only on the query and another vector that contains the frequencies of all itemsets. This is of course only useful if all the itemset frequencies are known. Otherwise, we need to find approximate answers, which is the subject of the next section.

3.3 Truncated inclusion-exclusion

We now return to the support estimation problem, and present the approach of truncating the sum $\vec{\xi_\phi} \cdot \vec{f}$ to the known terms. In this approach, which we name TRUNCATE SUM, we compute

$$\hat{f}(\phi) = \vec{\xi_\phi} \cdot \vec{f_0}, \tag{3.8}$$

where $\vec{f_0}$ is a vector where the unknown frequencies are replaced by zeros,

$$f_0(X) = \begin{cases} f(X), & X \in \mathcal{F}_\sigma, \\ 0, & \text{otherwise.} \end{cases}$$

This approach was originally suggested by Mannila and Toivonen [MT96]. Compared to their work, we give a more thorough analysis: we first consider the case of disjunctions and derive a bound which we show to be tight, and then generalize the bound to arbitrary queries.

First, we give a simple example. Assume that $\phi = \bigwedge C$ for some itemset $C \subset U$, i.e., $f(\phi) = f(C)$. Then we have $\zeta_\phi(Y) = [C \subset Y]$ and

$$\xi_\phi(X) = \sum_{Y \subset X} (-1)^{|X \setminus Y|} [Y \supset C] = \begin{cases} 1, & X = C, \\ 0, & X \neq C, \end{cases}$$

and therefore $\hat{f}(\phi) = f(\phi)$ if $C \in \mathcal{F}_\sigma$ and $\hat{f}(\phi) = 0$ otherwise. In the former case, the error is zero, and in the latter case, it is bounded by σ. We thus have the following result.

Proposition 3.11. *For a conjunction of attributes $\phi = \bigwedge C$,* TRUNCATE SUM *yields results to* APPROXIMATE QUERY *that have maximal absolute error σ.*

In Section 3.4, we analyze the TRUNCATE SUM algorithm for disjunctions of attributes ($\phi = \bigvee D$ for $D \subset U$), and show experimental results in Section 3.5. In Section 3.6 we address the problem for arbitrary queries, also with experimental results in Section 3.7. We conclude this section by looking briefly into how the TRUNCATE SUM algorithm can be implemented.

To compute the truncated sum (3.8), we need of course the frequencies of all frequent itemsets and the coefficients $\vec{\xi_\phi}$ of the query ϕ. Given these inputs, the computation is a simple inner product, which can be performed in time that is linear in the size of the inputs. If the query belongs to a suitably restricted class (such as the disjunctions of attributes considered in the following section), the coefficients $\vec{\xi_\phi}$ can be computed

on the fly. In the general case, we must compute these coefficients from the query.

We assume that the query is represented by listing the exact-frequency coefficients $\zeta_\phi(X)$ for every frequent itemset X. This is the most general way to represent Boolean queries, but may not be the simplest. If the query is given as a (syntactic) formula, we can either obtain the coefficients in a manner similar to (but simpler than) Proposition 3.8, or simply evaluate the query for each frequent itemset; the number of evaluations needed is the size of $\mathcal{F}_\sigma$, and thus linear in the size of the input.

Given the exact-frequency coefficients $\zeta_\phi(X)$ for frequent itemsets X, we can apply Eq. (3.2) to obtain the frequency coefficients needed for the truncated sum (3.8). The simplest possible way to evaluate the formula is inefficient: to compute $\xi_\phi(X)$, we need to sum $2^{|X|}$ numbers. However, we can trade a little space to reduce the computation time. Define for each set X and integer j, $0 \le j \le |X|$, the number $S_j(X)$ as the sum

$$S_j(X) = \sum_Y [Y \subset X]\,[|Y| = j]\,\zeta_\phi(Y).$$

If we can compute these numbers, we can easily obtain the coefficients $\xi_\phi(X) = \sum_j (-1)^{|X|-j} S_j(X)$. In the case $j = |X|$, the definition reduces to $S_j(X) = \zeta_\phi(X)$, and for $j < |X|$, we have

$$S_j(X) = \frac{1}{|X| - j} \sum_Y [Y \in \partial X]\, S_j(Y).$$

To see this, note that a j-element set $Z \subset X$ is included in every set $Y \in \partial X$ except those that lack an element of Z.

Thus we can compute the numbers S_j from the bottom up: first, $S_0(\emptyset) = \zeta_\phi(\emptyset)$; second, for singletons X, $S_0(X) = S_0(\emptyset)$ and $S_1(X) = \zeta_\phi(X)$; etc. For each set X, we thus need to sum up $O(|X|^2)$ numbers, much fewer than the $2^{|X|}$ required by the naive approach.

3.4 Disjunctions of attributes

We first attack the case where the query is a disjunction of attributes. We show a tight bound for the worst-case error of Truncate Sum; the bound depends exponentially on the size of the negative border.

We denote the disjunctive query by $\phi = \bigvee D$, where D is a set of attributes. The frequency of ϕ is the fraction of tuples that support ϕ; a tuple $T \in r$ supports ϕ if $D \cap T \neq \emptyset$. We can derive the coefficients ξ_ϕ from Eq. (3.2), as in Example 3.10:

$$f(\phi) = \sum_{X \subset D} [X \neq \emptyset]\,(-1)^{|X|+1} f(X). \tag{3.9}$$

The TRUNCATE SUM algorithm computes the shorter sum

$$\hat{f}(\phi) = \sum_{X \subset D} [\varnothing \neq X \in \mathcal{F}_\sigma] \, (-1)^{|X|+1} f(X). \tag{3.10}$$

This, essentially, is the query approximation approach proposed by Mannila and Toivonen [MT96]. In their Theorem 5, they give an error bound of $C \sum_{X \in \mathrm{Bd}^-} f(X)$ with $C = 2^{b-2} / b$, where b denotes the size of the negative border, $b = |\mathrm{Bd}^-|$. We improve the factor C to

$$\binom{b}{\lfloor b/2 \rfloor} / b$$

in our Theorem 3.16, and in Theorem 3.18 show that our bound is tight.

The disjunctive query $\phi = \bigvee D$ ignores attributes outside D. It is therefore useful to have a notation for the exact frequency of an itemset when attributes outside D are ignored.

Definition 3.12. *The* exact frequency of X relative to D, *denoted by* $g_D(X)$, *is defined for* $X \subset D$ *as the fraction of tuples in* r *whose intersection with* D *is* X.

Observe that we can write g_D as the frequency of a Boolean query:

$$g_D(X) = f\Big(\bigwedge_{A \in X} A \;\wedge\; \bigwedge_{A \in D \setminus X} \neg A \Big).$$

The error made in the approximation (3.10) is

$$e(\phi) = f(\phi) - \hat{f}(\phi) = \sum_{X \subset D} [X \in \mathcal{G}] \, (-1)^{|X|+1} f(X), \tag{3.11}$$

where by $\mathcal{G}$ we denote the family of non-frequent sets, i.e., the complement of $\mathcal{F}_\sigma$. We seek bounds that relate the error to the negative border Bd^-; the intuition behind such bounds is provided by the well-known Bonferroni inequalities, which state that if our collection $\mathcal{F}_\sigma$ happens to contain exactly the itemsets of size at most k, then the error is bounded by the sum of frequencies of itemsets of size $k+1$ [GS96]. We start from a simple lemma that we can use directly to obtain a rather weak inequality (Corollary 3.14) but that finds more use in the proof of Theorem 3.16.

Lemma 3.13. *Let* $\mathcal{G} \subset \mathcal{P}(U)$ *be an arbitrary upward-closed family of itemsets not including the empty set. Then*

$$\Big| \sum_{X \in \mathcal{G}} (-1)^{|X|} \Big| \leq \binom{|U|}{\lceil |U|/2 \rceil}.$$

Proof. One way of proving the Bonferroni inequalities is based on pairing up most of the tuples in $\mathcal{G}$ so that each pair cancels out; we proceed similarly. We first introduce some notation: let $t = |U|$ and $t' = \lceil t/2 \rceil$. It is well known that the power set $\mathcal{P}(U)$ can be written as a union of $\binom{t}{t'}$ disjoint chains, where a *chain* means a collection C of sets where for any two sets $X, Y \in C$, either $X \subset Y$ or $Y \subset X$ [Bol88, Theorem 1 of Section 4]. The construction of Bollobás yields chains that are symmetric and consist of consecutive sets: if we write $C = \{X_1, X_2, \ldots, X_k\}$ with $X_1 \subset X_2 \subset \ldots X_k$, then $|X_1| + |X_k| = t$ and $|X_{j+1}| = |X_j| + 1$ for all $1 \leq j < k$. Thus, if t is *odd*, each chain C is of even length, and the alternating sum $\sum_{X \in C} (-1)^{|X|+1}$ is zero. If t is *even*, the chains from the construction are of odd length. However, in that case we can remove one attribute A from U, perform the construction on $U \setminus \{A\}$ to obtain a collection of $\binom{t-1}{t'-1}$ chains, and then add to the collection a duplicate of each chain with A added to every set: the result is a partition of $\mathcal{P}(U)$ into $2\binom{t-1}{t'-1} = \binom{t}{t'}$ chains, each of which consists of an even number of consecutive sets.

We can thus assume that there is a partition $U = C_1 \cup C_2 \cup \cdots \cup C_m$ of U into $m = \binom{t}{t'}$ disjoint chains such that $\sum_{X \in C_j} (-1)^{|X|+1} = 0$ for each chain C_j. Now

$$\sum_{X \subset U} [X \in \mathcal{G}] \, (-1)^{|X|} = \sum_{j=1}^{m} \sum_{X \in C_j} [X \in \mathcal{G}] \, (-1)^{|X|}.$$

Every chain C_j that is wholly contained in either $\mathcal{G}$ or its complement contributes 0 to this sum. Every other chain contributes either 0 or ± 1. Therefore,

$$\left| \sum_{X \subset U} [X \in \mathcal{G}] \, (-1)^{|X|+1} \right| \leq m. \quad \square$$

Corollary 3.14. *For a disjunction of attributes $\phi = \bigvee D$,*

$$|e(\phi)| \leq \sum_{X \in \mathcal{G}} \binom{|X|}{\lceil |X|/2 \rceil} g_D(X).$$

Proof. We write the frequency as a sum over tuples:

$$f(X) = |r|^{-1} \sum_{T \in r} [T \supset X],$$

and therefore the error (3.11) is

$$e(\phi) = |r|^{-1} \sum_{T \in r} \sum_{X \subset T \cap D} [X \in \mathcal{G}] \, (-1)^{|X|+1}.$$

Applying Lemma 3.13, we obtain

$$|e(\phi)| \leq |r|^{-1} \sum_{T \in r} \binom{|T \cap D|}{\lceil |T \cap D|/2 \rceil}.$$

The claim follows by observing that $g_D(X) = |r|^{-1} \sum_{T \in r} [T \cap D = X]$. □

Recall that the Bonferroni inequalities, which apply to the case where the family $\mathcal{F}_\sigma$ consists of all itemsets of size at most k, give an error bound related to the itemsets of size $k+1$. A natural analogue of the size $k+1$ itemsets is the negative border Bd^- [MT96]. Recall that it is defined as the family of minimal non-frequent sets:

$$\mathrm{Bd}^- = \{ X \in \mathcal{G} \mid Y \in \mathcal{F}_\sigma \ \forall Y \subsetneq X \}.$$

If $\mathcal{F}_\sigma$ consists of sets of size at most k, then Bd^- is obviously the family of sets of size $k+1$. Thus, the negative border is a generalization of the Bonferroni bound.

We prove a lemma connecting the negative border to the error $e(\phi)$. First, we need to introduce some more notation: $\mathcal{G}_D$ is the set of non-frequent subsets of D,

$$\mathcal{G}_D = \{ X \mid X \in \mathcal{G}, X \subset D \},$$

and the *negative border relative to* D, denoted Bd_D^-, consists of the minimal sets in $\mathcal{G}_D$. Note that $\mathrm{Bd}_D^- \subset \mathrm{Bd}^-$, since if X is minimal in $\mathcal{G}_D$, all subsets of X are in $\mathcal{F}_\sigma$, and therefore X is minimal also in $\mathcal{G}$.

Lemma 3.15. *Consider the query* $\phi = \bigvee D$. *If* $\mathrm{Bd}_D^- \neq \{ \emptyset \}$, *the algorithm* TRUNCATE SUM *has an error of*

$$e(\phi) = \sum_{\emptyset \neq \mathcal{E} \subset \mathrm{Bd}_D^-} (-1)^{|\mathcal{E}|+|\bigcup \mathcal{E}|} g_D\Big(\bigcup \mathcal{E}\Big). \tag{3.12}$$

Before proving the lemma, we illustrate it by considering some simple cases. If Bd_D^- consists of a single set $B \neq \emptyset$, the error is an inclusion-exclusion sum

$$e(\phi) = \sum_X [B \subset X \subset D] \, (-1)^{|X|+1} f(X),$$

which is exactly the inclusion-exclusion expression for $(-1)^{|B|+1} g_D(B)$.

Likewise, if Bd^- is the two-set family $\{ B_1, B_2 \}$ with $B_j \cap D \neq \emptyset$ for $j = 1, 2$, we obtain

$$e(\phi) = \sum_X [B_1 \subset X \subset D \text{ or } B_2 \subset X \subset D] \, (-1)^{|X|+1} f(X).$$

We can use inclusion-exclusion to decompose the condition on X:

$$[B_1 \subset X \subset D \text{ or } B_2 \subset X \subset D] \\ = [B_1 \subset X \subset D] + [B_2 \subset X \subset D] - [B_1 \cup B_2 \subset X \subset D]$$

Thus we can break the formula for $e(\phi)$ into three components, which sum up to $g_D(B_1)$, $g_D(B_2)$ and $-g_D(B_1 \cup B_2)$. The proof of the lemma is a straightforward extension of this idea.

Proof of Lemma 3.15. Recall that the error is given by

$$e(\phi) = \sum_X [X \in \mathcal{G}_D]\,(-1)^{|X|+1} f(X). \tag{3.13}$$

We can rewrite the condition $X \in \mathcal{G}_D$ in terms of the minimal sets in $\mathcal{G}_D$ as follows. We have $X \in \mathcal{G}_D$ if and only if $X \supset B$ for some $B \in \mathrm{Bd}^-_D$. We apply inclusion-exclusion on the Iverson function:

$$\begin{aligned}[X \in \mathcal{G}_D] &= \sum_{B \in \mathrm{Bd}^-_D} [B \subset X \subset D] - \sum_{B_1, B_2 \in \mathrm{Bd}^-_D} [B_1 \cup B_2 \subset X \subset D] + \cdots \\ &= \sum_{\emptyset \neq \mathcal{I} \subset \mathrm{Bd}^-_D} (-1)^{|\mathcal{I}|+1} \left[\bigcup \mathcal{I} \subset X \subset D\right].\end{aligned}$$

Substituting this into the error sum (3.13) and changing the order of summation, we obtain

$$e(\phi) = \sum_{\emptyset \neq \mathcal{I} \subset \mathrm{Bd}^-_D} (-1)^{|\mathcal{I}|+1} \sum_X \left[\bigcup \mathcal{I} \subset X \subset D\right] (-1)^{|X|+1} f(X).$$

It now suffices to show that for $Y \subset D$,

$$(-1)^{|Y|} g_D(Y) = \sum_X [Y \subset X \subset D]\,(-1)^{|X|} f(X), \tag{3.14}$$

for then letting $Y = \bigcup \mathcal{I}$ yields (3.12). To establish (3.14), we consider a tuple $T \in r$ and write $T = R \cup S$ with $R \subset D$, $S \subset U \setminus D$. The tuple contributes $(-1)^{|R|}$ to all terms on the right-hand side where $Y \subset X \subset R$. In the case $R = Y$, there is only one such X and the contribution is exactly $(-1)^{|Y|}$. In all other cases, the contributions cancel out. □

Based on the lemma, we can prove an analogue to the Bonferroni inequalities that gives, however, rather large bounds compared to the Bonferroni case.

Theorem 3.16. *For a disjunction of attributes $\phi = \bigvee D$, the absolute error $|e(\phi)|$ of* Truncate Sum *is bounded by*

$$\binom{|\mathrm{Bd}^-_D|}{\lceil |\mathrm{Bd}^-_D|/2 \rceil} |\mathrm{Bd}^-_D|^{-1} \sum_{X \in \mathrm{Bd}^-_D} f(X).$$

Proof. Arrange the sum (3.12) in the form

$$e(\phi) = \sum_{X \in \mathcal{G}_D} \nu(X) g_D(X).$$

For the coefficients $\nu(X)$ we have

$$\nu(X) = (-1)^{|X|} \sum_{\mathcal{E} \subset \mathrm{Bd}_X^-} \left[\bigcup \mathcal{E} = X\right] (-1)^{|\mathcal{E}|}.$$

In this sum, the condition $\left[\bigcup \mathcal{E} = X\right]$ defines an upwards-closed subfamily of the powerset of Bd_X^-. We know from Lemma 3.13 that the absolute value of this alternating sum is bounded by $\binom{m}{m'}$ with $m = |\mathrm{Bd}_X^-|$ and $m' = \lceil m/2 \rceil$.

Arrange also the sum $\sum_{X \in \mathrm{Bd}_D^-} f(X)$ in the form

$$\sum_{X \in \mathcal{G}_D} \mu(X) g_D(X).$$

We have for the coefficients $\mu(X)$

$$\mu(X) = \sum_{Y \in \mathrm{Bd}_D^-} [Y \subset X] = |\mathrm{Bd}_X^-| > 0$$

for all $X \in \mathcal{G}_D$. The ratio $|\nu(X)|/\mu(X)$ is bounded by $\binom{m}{m'}/m$, and this bound is largest for $X = D$. Thus

$$\begin{aligned} |e(\phi)| &\le \sum_{X \in \mathcal{G}_D} |\nu(X)| g_D(X) \le \sum_{X \in \mathcal{G}_D} \Big(\max \frac{|\nu(X)|}{\mu(X)}\Big) \mu(X) g_D(X) \\ &\le \binom{|\mathrm{Bd}_D^-|}{\lceil |\mathrm{Bd}_D^-|/2 \rceil} |\mathrm{Bd}_D^-|^{-1} \sum_{X \in \mathrm{Bd}_D^-} f(X). \end{aligned}$$

□

Using the inequality $f(X) < \sigma$ for $X \in \mathrm{Bd}^-$, we can obtain a form of the bound that is independent of the actual frequencies of sets in the border.

Corollary 3.17. *For a disjunction of attributes $\phi = \bigvee D$,*

$$e(\phi) \le \binom{|\mathrm{Bd}^-|}{\lceil |\mathrm{Bd}^-|/2 \rceil} \sigma.$$

Thus, the bound depends superpolynomially on the size of the border. A natural question is whether the bound can be improved. The answer is negative: the bound is in the worst case tight. To prove this, we construct an example with a small negative border and a large approximation error.

The key part in the construction is arranging for equality to hold when Lemma 3.13 is used in the proof of Theorem 3.16. This is the case when the minimal families $\mathcal{E} \subset \mathrm{Bd}_X^-$ that satisfy the condition $\bigcup \mathcal{E} = X$ are exactly of size $\lceil |\mathrm{Bd}_X^-|/2 \rceil$.

Theorem 3.18. *Given any positive integer m, there exists a set U, a relation r over U, and a downward-closed collection of itemsets $\mathcal{F}$ such that $|\mathrm{Bd}^-| \geq m$, and for the disjunctive query $\phi = \bigvee U$ the absolute error of* Truncate Sum *is*

$$|e(\phi)| = \binom{|\mathrm{Bd}^-|}{\lceil |\mathrm{Bd}^-|/2 \rceil} |\mathrm{Bd}^-|^{-1} \sum_{X \in \mathrm{Bd}^-} f(X).$$

Proof. Choose integer parameters $p > k > \lceil m/2 \rceil$; p will be the number of sets in the negative border, and we will see later that choosing $p = 2k + 1$ suits our purposes well. We need $n = \binom{p}{k}$ attributes: let $U = [n] = \{1, \ldots, n\}$. We set up Bd^- so that for all families $\mathcal{E} \subset \mathrm{Bd}^-$, $|\mathcal{E}| \leq k$ implies $\bigcup \mathcal{E} \neq U$, and $|\mathcal{E}| > k$ implies $\bigcup \mathcal{E} = U$. To achieve this, we first enumerate all the k-element subsets of $[p]$; there are n of them, and we name them $K_1, K_2, \ldots, K_n$ in some arbitrary order. Then for all $q \in [p]$, we define W_q as the set of those i such that $q \notin K_i$. Let $\mathrm{Bd}^- = \{ W_q \mid q \in [p] \}$. Note that Bd^- is an antichain, since all sets W_q have the same number of elements; thus we can define $\mathcal{F}$ as the downward-closed collection of sets that are not supersets of any sets in Bd^-, and Bd^- will automatically be the negative border corresponding to $\mathcal{F}$.

We must now prove the assertion that for $\mathcal{E} \subset \mathrm{Bd}^-$, $\bigcup \mathcal{E} = U$ if and only if $|\mathcal{E}| > k$. Given any collection $\mathcal{E}$ of border sets, we can write $\mathcal{E} = \{ W_q \mid q \in Q \}$ for some index set $Q \subset [p]$. If $|\mathcal{E}| = |Q| \leq k$, some set K_i must be a superset of the index set Q, since we have enumerated all k-element subsets of $[p]$. But then we have that $i \notin \bigcup \mathcal{E}$, and thus $\bigcup \mathcal{E} \neq U$. Conversely, if $\bigcup \mathcal{E} \neq U$, there must be some $i \notin \bigcup \mathcal{E}$, and therefore for all $q \in Q$ we must have $q \in K_i$, because $i \notin W_q$. But this means that $Q \subset K_i$, and therefore $|\mathcal{E}| = |Q| \leq |K_i| = k$. We have thus shown that $\bigcup \mathcal{E} = U$ if and only if $|\mathcal{E}| > k$.

For the query ϕ, we select the disjunction of all attributes, $\phi = \bigvee U$. The terms $g_D(X)$ reduce to the usual exact frequencies $g(X)$, and the family Bd_D^- to the usual negative border Bd^-. We make all sets in the complement of $\mathcal{F}$ equally frequent by letting $g(U) = c$ and $g(X) = 0$ for all $X \notin \mathcal{F}$, $X \neq U$. To ensure that $\mathcal{F} = \mathcal{F}_\sigma$ for some value of σ and that $\sum_X g(X) = 1$, we let $g(X) = (1 - c)/|\mathrm{Bd}^+|$ for all $X \in \mathrm{Bd}^+$, where by Bd^+ we denote the sets in $\mathcal{F}$ that are maximal with respect to subset containment (the "positive border").

Now we are in a position to apply Lemma 3.15. The sum over $\mathcal{E} \subset \mathrm{Bd}^-$ becomes a sum over those $\mathcal{E}$ for which $\bigcup \mathcal{E} = U$, since $g(\bigcup \mathcal{E}) = 0$ otherwise. By the construction, these are exactly those $\mathcal{E}$ such that $|\mathcal{E}| > k$. Thus

$$e(\phi) = \sum_{\mathcal{E} \subset \mathrm{Bd}^-} (-1)^{|\mathcal{E}|+|r|} [|\mathcal{E}| > k] c = (-1)^{|r|} \sum_{j=k+1}^{n} (-1)^j \binom{p}{j} c.$$

It is an easy proof by induction that

$$\sum_{j=k+1}^{n} (-1)^j \binom{p}{j} = (-1)^{k+1} \binom{p-1}{k}.$$

If we now let $p = 2k + 1$, we have

$$|e(\phi)| = \binom{2k}{k} = \binom{|\mathrm{Bd}^-|}{|\mathrm{Bd}^-|/2} c.$$

Since we have $f(X) = c$ for all $X \in \mathrm{Bd}^-$, the frequency sum of sets in the border is $\sum_X [X \in \mathrm{Bd}^-] f(X) = |\mathrm{Bd}^-| c = |\mathrm{Bd}_D^-| c$. □

While the construction creates a small number of sets in the border, there are of course many sets that are "almost" in the border, which is not true in the usual Bonferroni situation. Thus, we can define another kind of border, which in fact gives rise to a Bonferroni-like inequality.

Definition 3.19. *The* thick negative border Bd_*^- *is the family of itemsets X that are not frequent but that have at least one attribute A such that $X \setminus \{A\}$ is frequent.*

Theorem 3.20. *For a disjunction of attributes $\phi = \bigvee D$,*

$$|e(\phi)| \leq \sum_{X \in \mathrm{Bd}_*^-} f(X).$$

Proof. We write the error $e(\phi)$ as a sum over all tuples $T \in r$, $e(\phi) = |r|^{-1} \sum_T [T \in r] \kappa(T)$, and show that the contribution $\kappa(T)$ made by T towards $e(\phi)$ is bounded by the number of subsets of T in Bd_*^-, which implies the claimed upper bound. The contribution is

$$\kappa(T) = \sum_{X \subset T \cap D} [X \in \mathcal{G}] (-1)^{|X|+1}, \tag{3.15}$$

where, again, $\mathcal{G}$ is the complement of $\mathcal{F}_\sigma$. Select any attribute $A \in T \cap D$, and delete from the sum (3.15) all pairs $X, Y \in \mathcal{G}$ such that $Y = X \cup \{A\}$. The sum becomes

$$\kappa(T) = \sum_{X \subset T \cap D} [X \in \mathcal{G}] [X \setminus \{A\} \in \mathcal{F}_\sigma] (-1)^{|X|+1}.$$

All sets fulfilling both conditions of the sum are in $\mathrm{Bd}^-_* \cap \mathcal{P}(T \cap D)$, and therefore

$$|\kappa(T)| \leq \sum_X [X \subset T \cap D]\,[X \in \mathrm{Bd}^-_*] \leq \sum_{X \subset T} [X \in \mathrm{Bd}^-_*].$$

For the error we obtain by changing the order of summation

$$|r| \cdot |e(\phi)| \leq \sum_X [X \in \mathrm{Bd}^-_*] \sum_{T \in r} [X \subset T] = |r| \sum_X [X \in \mathrm{Bd}^-_*]\, f(X). \quad \square$$

We have proved two theorems for upper-bounding the absolute error: Theorems 3.16 and 3.20. Both theorems are problematic in practice: the bound of Theorem 3.16 grows exponentially, and the thick border of Theorem 3.20 can be very large. It would be useful to find a bound for TRUNCATE SUM in-between these two theorems. Note that the construction of Theorem 3.18 creates a large number of maximal frequent sets. By analogy with the negative border, one can define the *positive border* Bd^+ as the collection of these sets. For the construction, Bd^+ is large and Bd^- is small; in many practical cases, Bd^+ is smaller and Bd^- larger. The set $\mathrm{Bd}^+ \cup \mathrm{Bd}^-$ seems worth investigating, and we conjecture (as in [Man02, SM05]) that

$$e(\phi) \leq \sum_X [X \in \mathrm{Bd}^- \cup \mathrm{Bd}^+]\, f(X).$$

3.5 Experimental evaluation

To see how well TRUNCATE SUM works in practice, it was evaluated on the data sets described in Chapter 2. The overall test procedure was as follows. For each data set, a number of random disjunctive queries were prepared; frequent itemsets were mined at various thresholds σ; the algorithm was run to find its estimates for the queries; and these estimates were compared to the correct values and to the estimates of two other algorithms, described later. Details are provided in the following.

Test data. For each data set, a threshold σ was selected. As can be observed from Figure 2.1 on page 12, it would not be reasonable to use the same threshold for every data set: at $\sigma = 0.1$, no set in *T10I4D100K* is frequent, but at $\sigma = 0.9$, thousands of sets in *Pumsb* are frequent. The values of σ used are listed in Table 3.1, along with the sizes of the frequent set collections and the negative borders.

Data set	Threshold	Frequent sets	Border sets
T10I4D100K	0.01	386	70741
Kosarak	0.01	384	42580
Course	0.02	2349	1516
Abstracts	0.03	3019	202430
T40I10D100K	0.08	138	10179
Mushroom	0.30	2736	415
*Pumsb**	0.60	168	7166
Chess	0.90	623	125
Connect	0.90	27128	395
Pumsb	0.95	173	7183

Table 3.1. Thresholds σ with corresponding sizes $|\mathcal{F}_\sigma|$ and $|\mathrm{Bd}^-|$ of frequent itemset collections and negative borders.

Queries. Two sets of test queries were prepared, *Disjunctions 1* and *Disjunctions 2*. Both sets contain only disjunctions of attributes, i.e., formulas of the form

$$\phi = \bigvee D$$

with D a set of attributes. In both sets, the number of attributes in a disjunction varies from 2 to 20, and there are 100 queries of each size. The difference between the sets is that in *Disjunctions 1* the attributes are selected completely randomly, and in *Disjunctions 2*, the selection probability of each attribute A is weighted by $f(A)$; that is, attributes that occur often in the data also occur often in the queries. Both ways of selecting the queries could be argued to simulate real life: in some cases, users could be equally interested in all attributes, while in other cases, the most frequently occurring attributes could also be the most interesting.

Algorithms. The TRUNCATE SUM algorithm described in the previous sections was the main focus of the test. Since it sometimes gives answers outside the range $[0, 1]$, its results were clipped into this range. For comparison, two model-based algorithms were implemented: INDEPENDENCE and ITERATIVE PROPORTIONAL FITTING. The INDEPENDENCE algorithm is based on the assumption that the attributes are independent, regardless of the information in the frequent itemsets: for the query $\phi = \bigvee D$, the algorithm returns

$$\hat{f}(\phi) = 1 - \prod_{A \in D} (1 - f(A)).$$

The algorithm does not really implement a solution of the APPROXIMATE QUERY problem, as it needs the frequencies of all attributes, even the infrequent ones. However, all these frequencies are discovered by APRIORI, as every itemset of size 1 must be either frequent or in the border.

The second algorithm is a generalization of the first one. One argument for the INDEPENDENCE algorithm is that if only the single-attribute frequencies are known, the independence assumption provides the maximum-entropy distribution fitting the frequencies. If, however, more frequencies are known, the task of finding the maximum-entropy distribution fitting the input is equally well-defined, and somewhat surprisingly, there is an elegant algorithm that discovers the distribution. Once the distribution is known, it is easy to find the estimate $\hat{f}(\phi)$ from the distribution.

To formalize the task, we interpret the vector $\vec{g}$ as a probability distribution over U, and define the Shannon entropy

$$H(\vec{g}) = -\sum_X g(X) \log g(X)$$

for each possible distribution. We seek the distribution that maximizes the entropy subject to the constraints arising from the frequent itemset collection: every set in $\mathcal{F}_\sigma$ must have the correct frequency. This is solved by the ITERATIVE PROPORTIONAL FITTING algorithm, which starts from the uniform distribution and iterates correcting the distribution to comply with each constraint. Each correction may break earlier constraints, but the algorithm has been proved to converge to the maximum-entropy distribution [DS40, IK68]. This algorithm has been used in the frequent itemset context by Pavlov et al. [PMS00, PS01].

The ITERATIVE PROPORTIONAL FITTING algorithm is elegant and easy to implement, but there are some disadvantages. The most serious problem is scalability. Since the algorithm keeps in memory the full distribution $\vec{g}$, consisting of $2^{|U|}$ numbers, the method as described above is clearly only applicable for very few attributes. The natural way to cope with this is to project the data to the subset of attributes appearing in the query. For more than about 13 attributes, the algorithm was impractically slow. As the slowdown is exponential, it is unlikely that any simple optimizations of the algorithm would allow processing significantly larger queries; thus the algorithm was only run for queries having at most 13 attributes.

Another problem is that if only the frequent itemsets are used as input, the algorithm often gives results that are very far off the mark. This problem is an analogue of the need in INDEPENDENCE to include all attributes in the input, not only frequent ones. If an infrequent attribute is omitted, the maximum-entropy assumption is that its frequency is 0.5. Similar

situations can also happen at higher levels of the itemset lattice: if we are estimating the frequency of, say $A \vee B$, and A and B are frequent but AB is not, the maximum-entropy assumption is that $f(AB) = f(A)f(B)$, no matter how low the frequency threshold σ is. To alleviate this kind of problems, the negative border was added to the input. It is obvious from Table 3.1 that in many cases the input becomes considerably larger, sometimes by two orders of magnitude. Even the number of sets of size 1 (see Table 2.2 on page 11) is sometimes much larger than the number of frequent itemsets. In order to make a fair comparison, results for TRUNCATE SUM were also computed with the border sets included in its input.

Another implementation detail is the convergence criterion: the algorithm was deemed to have converged when the L_1 distance between successive distributions falls below 0.001, or after 100 iterations.

Results. Figures 3.1 and 3.2 display the errors for the three algorithms for the two sets of queries in the various data sets. The errors are visualized using a "box and whiskers" plot, which shows the median value as a horizontal line and the interquartile range as a box around the median; that is, the lower end of the box shows the 25th percentile and the upper end the 75th percentile. The whiskers (dashed lines ending in a short horizontal bar) show the range of the data outside the box, but the distance between the whiskers is restricted to be at most 1.5 times the interquartile range; points outside this range are plotted individually. For example, the *Course* panel of Figure 3.1 shows that in the *Course* data the TRUNCATE SUM algorithm without border sets (leftmost plot) makes a median error of about 0.02, most of the errors are below 0.05, but the maximal errors are about 0.18; and when the border sets are included (second plot), the majority of the errors are smaller, but the maximal errors reach over 0.25.

Overall, these results show that the INDEPENDENCE algorithm usually outperforms TRUNCATE SUM on the data sets and queries tested, with the thresholds used. The exceptions occur in the *Course* and *Kosarak* data sets: in *Course*, TRUNCATE SUM is clearly better than INDEPENDENCE on both query sets even without the border sets, and similarly in *Kosarak* in the *Disjunctions 2* queries, where the most frequently occurring attributes are weighted. Interestingly, there are some cases where including the border sets *increases* the errors made by TRUNCATE SUM; this occurs clearly in *Disjunctions 2* in *Course*, *Kosarak*, and *Mushroom*, and to some extent in *Disjunctions 1* in *Mushroom*; there, the median and maximum errors increase, although the third quartile decreases slightly. We will return to this issue in Section 3.8.

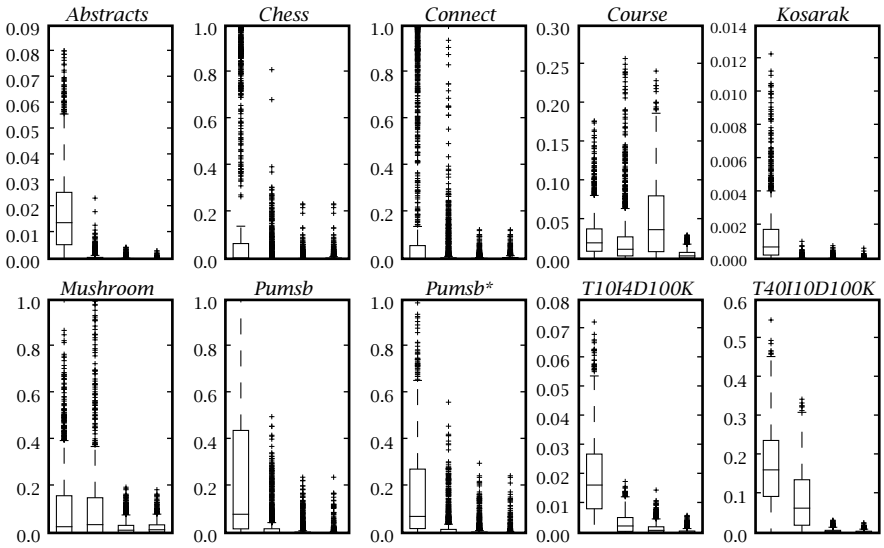


Figure 3.1. Results on *Disjunctions 1* in all data sets. In each subfigure, the leftmost plot shows the absolute error of Truncate Sum without the border sets, the second plot shows the corresponding results with the border sets included, the third plot shows the absolute error of Independence, and the rightmost plot the absolute error of Iterative Proportional Fitting. Note that Iterative Proportional Fitting could only be run for 1200 of the 1900 queries (those with at most 13 attributes), and that the scaling differs between the subfigures.

The ITERATIVE PROPORTIONAL FITTING algorithm was overall the best of the three. This result is not surprising, since INDEPENDENCE coincides with the the $\sigma = 1$ special case of ITERATIVE PROPORTIONAL FITTING: in this case the input consists of exactly the sets of size 1 (except if some sets other than $\emptyset$ have frequency 1, which does not occur in the test data sets), in which case the maximum entropy principle reduces to the independence assumption. It must be remembered, however, that it was possible to run ITERATIVE PROPORTIONAL FITTING only for the 1200 queries with at most 13 attributes, and the results are thus not directly comparable. Also, as was pointed out in the description of the algorithms, it needs more input than TRUNCATE SUM: if the sets in the border are omitted, many of the results (not shown) are much worse. When the sets in the border are included, the results of TRUNCATE SUM are in many cases close to those of ITERATIVE PROPORTIONAL FITTING.

Figures 3.3 and 3.4 illustrate the dependence of the results on the number of attributes in the queries in two of the data sets. Figure 3.3 depicts errors in *Course* data, Figure 3.4 in *T10I4D100K*. In each figure,

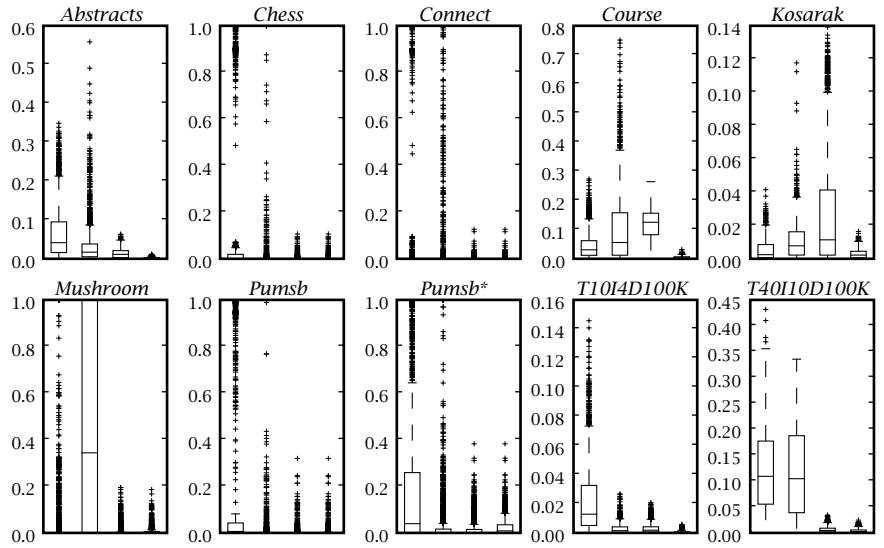


Figure 3.2. Results on *Disjunctions 2* in all data sets. As in Figure 3.1, the plots correspond from left to right to Truncate Sum without and with the border sets, Independence, and Iterative Proportional Fitting.

the errors on the *Disjunctions 1* queries are displayed as a function of the number of attributes in the queries. Figure 3.3 shows that the improvement observed in TRUNCATE SUM when adding the border sets is mainly due to the queries with few attributes. In the longer queries, both the large outliers and the majority of the errors are in fact larger than without the border sets. The results shown in Figure 3.4 are dramatically different: adding the border sets improves the results across all queries.

In both cases, the absolute error of TRUNCATE SUM increases with the length of the query. There are at least two possible explanations. First, when the set D of attributes in the query $\phi = \bigvee D$ increases, the number $|\mathcal{G}_D|$ of non-frequent subsets of D increases, and so does the size of the border relative to D, Bd^-_D. Indeed, the limit of Theorem 3.16 depends on the size of the border. The second possible explanation is that the error could depend on the result $f(\phi)$, which also increases with $|D|$. For example, an algorithm that always returns the estimate $\hat{f}(\phi) = 0$ would behave like this. The first explanation is suggested by Figure 3.5, where the errors $e(\phi)$ are plotted against the correct results $f(\phi)$ for different-length queries in *Disjunctions 1* and *2* in *Course* data. The figure exhibits a clear dependence between the query length and the correct result, and this dependence seems to explain the increase in the error.

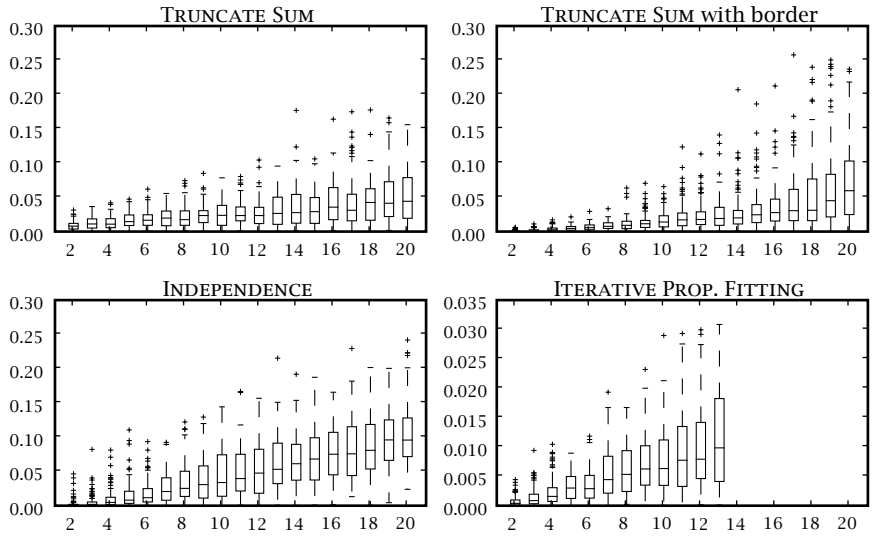


Figure 3.3. Errors conditioned on the number of attributes in a query in *Course* data, *Disjunctions 1* queries, threshold $\sigma = 0.02$. Note the different scale in the last panel.

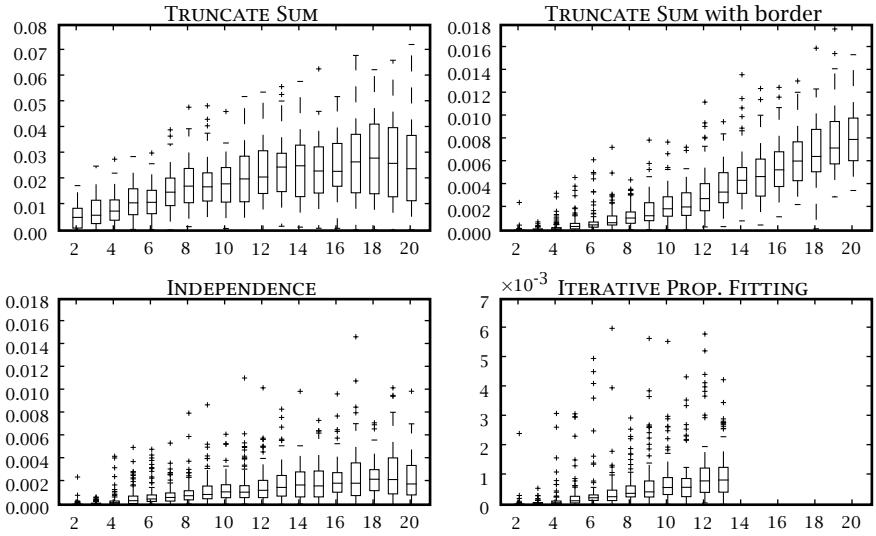


Figure 3.4. Errors conditioned on the number of attributes in a query in *T10I4D100K* data, *Disjunctions 1* queries, threshold $\sigma = 0.01$. Note the different scale in the first and last panels.

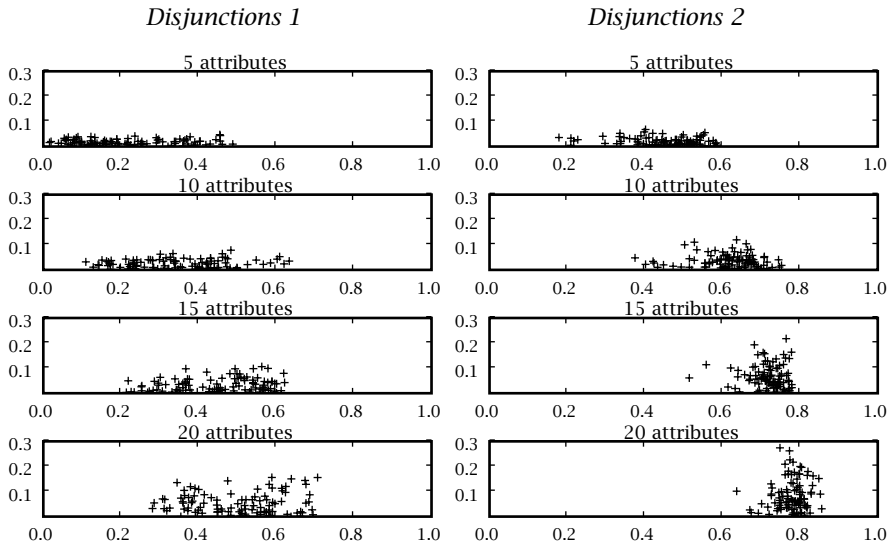


Figure 3.5. Dependence of errors on real frequencies in *Disjunctions 1* and *2*, *Course* data, $\sigma = 0.02$. Correct frequency $f(\phi)$ on x-axis, error $e(\phi)$ of Truncate Sum on y-axis.

In order to further investigate this phenomenon and to test how close the upper bound of Theorem 3.16 is to reality, Figures 3.6 and 3.7 display the errors relative to the border. The quantity depicted in the box-and-whiskers plots is $|e(\phi)| / \sum_X [X \in \mathrm{Bd}_D^-]\, f(X)$, the error relative to the sum of the frequencies of the sets in the negative border. To prevent division by zero, cases where $e(\phi) = 0$ were plotted as 0, regardless of the value of the border sum. When computing the error $e(\phi)$, the estimate $\hat{f}(\phi)$ was *not* clipped to the interval $[0, 1]$. Thus, Theorem 3.18 implies that this quantity could be as high as $\binom{m}{\lceil m/2 \rceil} / m$ with $m = |\mathrm{Bd}_D^-|$. Remarkably, the highest values are approximately 1.0, showing that the construction of Theorem 3.18 is quite artificial.

To check against implementation errors, an end-to-end test was run, using as input the construction of Theorem 3.18 with parameters $p = 7$, $k = 2$, $c = 1/2$, and computing the error relative to the sum of frequencies in the border, as in Figure 3.6. The result was $2.14 \approx \binom{6}{2}/7$, as predicted by the theorem. With these parameters, there were 2096730 frequent itemsets, close to the 2^{21} possible. Testing the case $k = 3$ proved computationally difficult, as the number of frequent itemsets would have been of the order of 2^{35}.

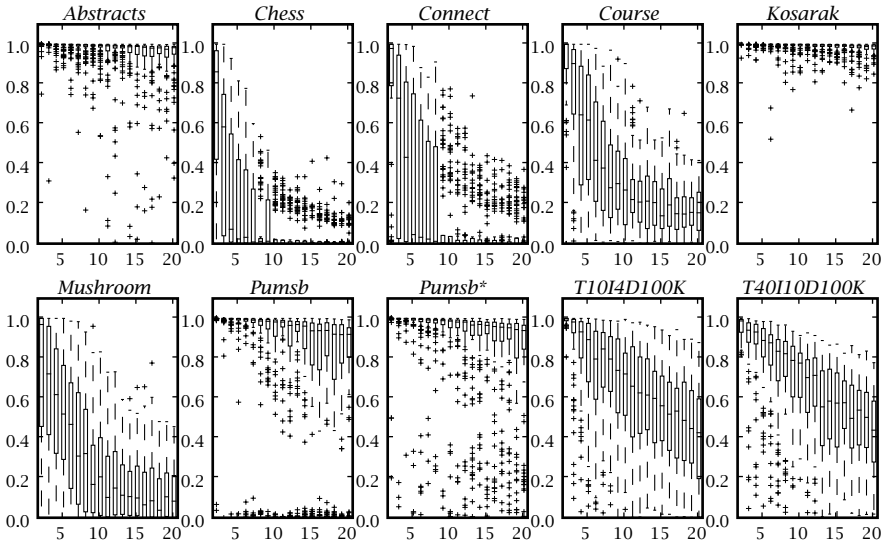


Figure 3.6. Error of Truncate Sum relative to the sum of frequencies of sets in the border, $|e(\phi)| / \sum_X [X \in \mathrm{Bd}_D^-] f(X)$ for *Disjunctions 1*. In computing the error $e(\phi)$, the result $\hat{f}(\phi)$ of Truncate Sum is not clipped into $[0,1]$. Length of query on x-axis.

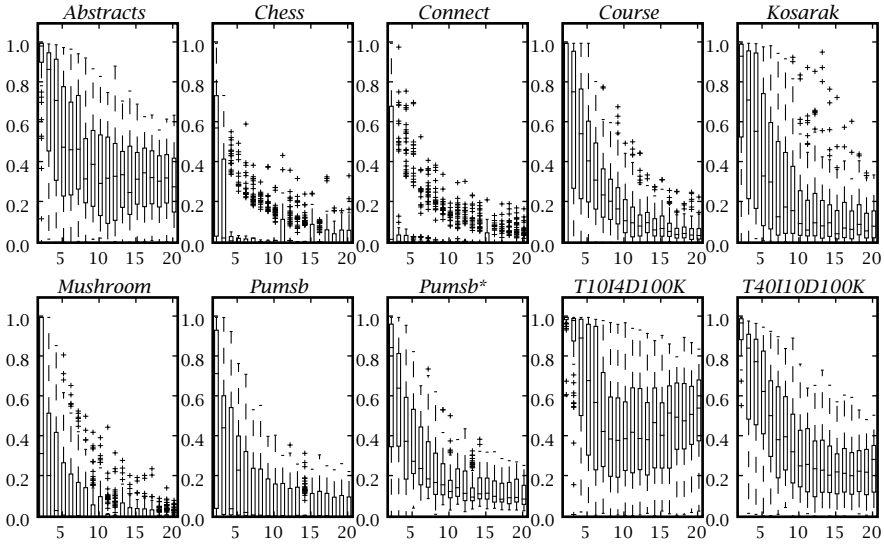


Figure 3.7. Error of Truncate Sum relative to the sum of frequencies of sets in the border, $|e(\phi)| / \sum_X [X \in \mathrm{Bd}_D^-] f(X)$ for *Disjunctions 2*. In computing the error $e(\phi)$, the result $\hat{f}(\phi)$ of Truncate Sum is not clipped into $[0,1]$. Length of query on x-axis.

A fact partially obscured by the box-and-whiskers plots in Figures 3.6 and 3.7 is that the distribution of errors is bimodal, especially in the dense data sets such as *Chess* and *Connect*: one mode is close to 0, another is higher. To investigate the effect of the threshold σ on the results, the errors for a small number of queries were plotted in Figures 3.8, 3.9 and 3.10, for the data sets *Abstracts*, *Mushroom*, and *T40I10D100K*, respectively. The two rows show results for the query sets *Disjunctions 1* and *2*, and the three columns correspond to three different queries selected from each of the query sets. The queries were selected by the absolute difference in the errors of INDEPENDENCE and TRUNCATE SUM: the first column shows the query where INDEPENDENCE wins by the largest margin, the second the median query, and the third the query where TRUNCATE SUM wins by the largest margin. The number of attributes in each selected query is indicated in the subfigures. In each subfigure, the x-axis corresponds to σ and the y-axis to absolute frequencies. The solid line displays the real frequency $f(\phi)$ and the dotted line the result of INDEPENDENCE (which always uses all sets of cardinality 1 and thus does not depend on σ). The circles display the estimate of TRUNCATE SUM, again not clipped to $[0, 1]$, and without using the border sets. The crosses display the estimate if the border sets are included.

Figure 3.8 shows that the query in *Disjunctions 1* where INDEPENDENCE most beats TRUNCATE SUM is one where the individual attributes have low frequencies that have a relatively large sum. The algorithms that use the individual frequencies as input do very well in this case. The median case is similar, only with a lower frequency of the query. The third case, where TRUNCATE SUM wins INDEPENDENCE by the largest margin, clearly shows the result dropping by about 0.04 when σ increases from 0.03 to 0.05. The compensation by the border sets (the crosses) helps in this case, as in the preceding ones. In *Disjunctions 2*, where attributes are chosen with a weight proportional to their frequency, the worst case for TRUNCATE SUM is fairly bad, and adding the border is an improvement as often as not. In the best case, however, adding the border is definitely an improvement with all values of σ used except for the smallest (and even then the degradation is slight).

Figure 3.9, generated from the fairly dense *Mushroom* data set, shows that in both of the worst cases increasing σ improves the results. This seeming paradox may be caused by the selection criterion: these are the cases where the results of TRUNCATE SUM at $\sigma = 0.3$ are extremely bad, and reducing the number of terms in the sum is likely to improve the result. In the median cases the results are better, and the performance degrades with increasing σ, as would be expected. In most cases including the border in the sum causes severe degradation in the results.

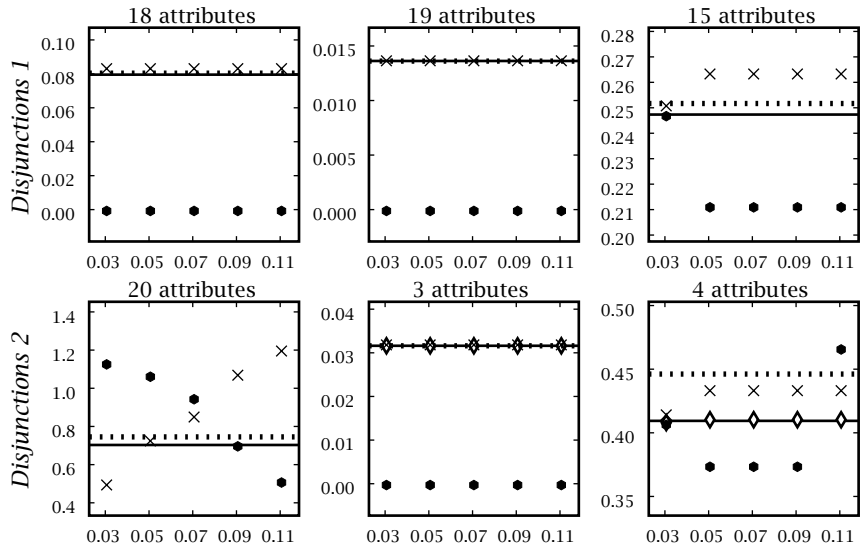


Figure 3.8. Estimates for single queries in *Abstracts* with varying thresholds σ. Solid line: real frequency; dotted line: Independence; circles: Truncate Sum; crosses: Truncate Sum with border sets; diamonds: Iterative Proportional Fitting (when possible). Note the different vertical scales in subfigures, and see the text on how the queries were selected.

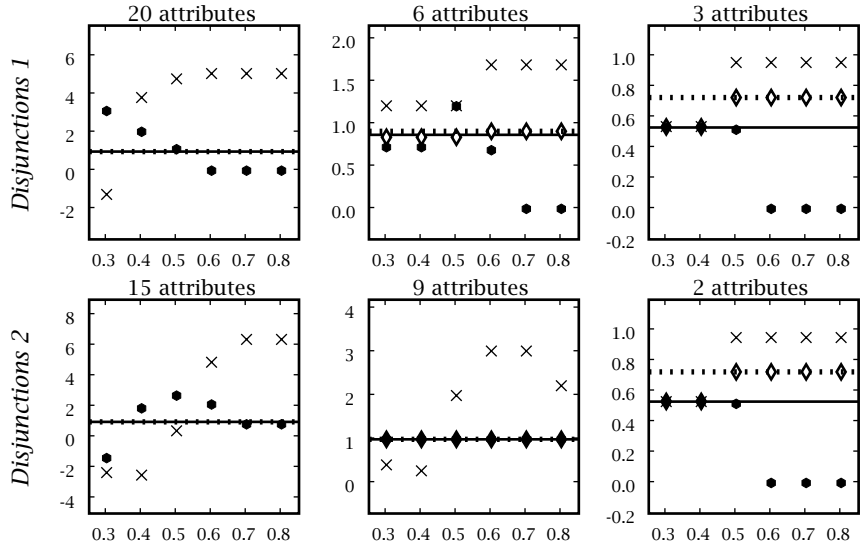


Figure 3.9. Estimates for single queries in *Mushroom* with varying thresholds σ. Symbols as in Figure 3.8. Note the different vertical scales in subfigures, and see the text on how the queries were selected.

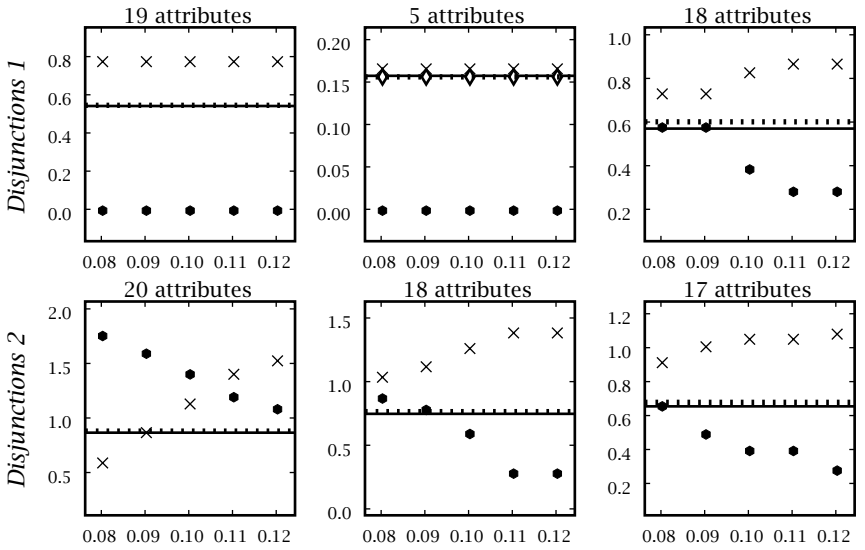


Figure 3.10. Estimates for single queries in *T40I10D100K* with varying thresholds σ. Symbols as in Figure 3.8. Note the different vertical scales in subfigures, and see the text on how the queries were selected.

Figure 3.10 is based on the *T40I10D100K* data, which has the lowest number of frequent itemsets at the threshold $\sigma = 0.08$ used. Thus it is not surprising that the worst case in *Disjunctions 1* has zero as the estimate of TRUNCATE SUM, and many attributes in the query. The worst case in *Disjunctions 2* is similar to the one in the *Mushroom* data. Again, including the border in the sum helps in the bad cases in *Disjunctions 1* but mostly makes the result worse in other cases.

Conclusions. With the query models used in this experiment, TRUNCATE SUM performs much better on the sparser data sets tested. Adding the border sets usually improves the performance of TRUNCATE SUM, but not consistently. The INDEPENDENCE algorithm often beats TRUNCATE SUM, with some exceptions, and ITERATIVE PROPORTIONAL FITTING is usually even better on those queries where it can be run in a reasonable time.

The errors increase with the length of the query, which is more likely attributable to the increase in the size of the border than to a dependence on the error on the frequency of the query. However, the error does not increase nearly as fast as the sum of the frequencies of the sets in the border for any of the data sets tested, although the theoretical upper bound of Theorem 3.16 suggests that the inverse could be true. As a function of σ, the performance of TRUNCATE SUM is fairly unstable.

3.6 Arbitrary queries

In this section we generalize the discussion of Section 3.4 by proving counterparts of Lemma 3.15 and Theorem 3.16. The bounds provided by these results can be even larger than the disjunction-specific bounds of Section 3.4.. We start by proving a generalization of Lemma 3.15, which represents the error using inclusion-exclusion on itemset families $\mathcal{E} \subset \mathrm{Bd}^-$.

Lemma 3.21. *When ϕ is a Boolean formula with exact-frequency coefficients $\zeta(X) = [X \vDash \phi]$ and the border Bd^- does not contain the empty set,*

$$e(\phi) = \sum_X \nu(X) g(X),$$

where

$$\nu(X) = (-1)^{|X|} \sum_{\emptyset \neq \mathcal{E} \subset \mathrm{Bd}^-_X} (-1)^{|\mathcal{E}|+1} \sum_Y [X \setminus \bigcup \mathcal{E} \subset Y \subset X] \, (-1)^{|Y|} \zeta(Y).$$

Proof. The error is

$$\begin{aligned} e(\phi) &= \sum_X [X \in \mathcal{G}] \, \xi(X) f(X) \\ &= \sum_{X,Y} [X \in \mathcal{G}] \, f(X) \, [Y \subset X] \, (-1)^{|X \setminus Y|} \zeta(Y). \end{aligned}$$

Again we apply inclusion-exclusion on the condition $X \in \mathcal{G}$:

$$[X \in \mathcal{G}] = \sum_{\emptyset \neq \mathcal{E} \subset \mathrm{Bd}^-} (-1)^{|\mathcal{E}|+1} \, [X \supset \bigcup \mathcal{E}],$$

obtaining

$$\begin{aligned} e(\phi) &= \sum_{\emptyset \neq \mathcal{E} \subset \mathrm{Bd}^-} (-1)^{|\mathcal{E}|+1} \sum_{X,Y} [X \supset \bigcup \mathcal{E}] \, f(X) \, [Y \subset X] \, (-1)^{|X \setminus Y|} \zeta(Y) \\ &= \sum_{\emptyset \neq \mathcal{E} \subset \mathrm{Bd}^-} (-1)^{|\mathcal{E}|+1} \sum_Y (-1)^{|Y|} \zeta(Y) \sum_X [X \supset \bigcup \mathcal{E} \cup Y] \, (-1)^{|X|} f(X) \\ &= \sum_{\emptyset \neq \mathcal{E} \subset \mathrm{Bd}^-} (-1)^{|\mathcal{E}|+1} \sum_Y (-1)^{|Y|+|\bigcup \mathcal{E} \cup Y|} \zeta(Y) g(\bigcup \mathcal{E} \cup Y). \end{aligned}$$

Regrouping the terms yields

$$e(\phi) = \sum_X g(X) (-1)^{|X|} \sum_{\emptyset \neq \mathcal{E} \subset \mathrm{Bd}^-_X} (-1)^{|\mathcal{E}|+1} \sum_Y [X \setminus \bigcup \mathcal{E} \subset Y \subset X] \, (-1)^{|Y|} \zeta(Y),$$

which is the claim. □

To see that this generalizes Lemma 3.15, let $\phi = \bigvee D$. Then $\zeta(X) = [X \cap D \neq \emptyset]$. Consider the sum over Y:

$$\sum_Y [X \setminus \bigcup \mathcal{E} \subset Y \subset X] \, (-1)^{|Y|} [Y \cap D \neq \emptyset]. \tag{3.16}$$

We may assume that $\bigcup \mathcal{E} \subset X$, since the outer sum is taken over $\mathcal{E} \subset \mathrm{Bd}_X^-$. Furthermore, if $\bigcup \mathcal{E}$ contains any attribute A that is not in D, we can pair up terms corresponding to $Y \ni A$ and $Y \setminus \{A\}$, and thus show that the sum (3.16) is 0. On the other hand, if X contains any attributes that are in D but not in $\bigcup \mathcal{E}$, the Iverson function $[Y \cap D \neq \emptyset]$ is always 1, and the sum (3.16) is seen to compute the difference in number of even and odd subsets of $\bigcup \mathcal{E}$, which is of course 0 since $\bigcup \mathcal{E} \neq \emptyset$.

Assume now that $X = \bigcup \mathcal{E} \cup Z$ with $\bigcup \mathcal{E} \subset D$ and $Z \cap D = \emptyset$. We thus have for $Z \subset Y \subset X$ that $[Y \cap D \neq \emptyset] = [Y \neq Z]$, and the sum (3.16) becomes $-(-1)^{|Z|} = (-1)^{|X \setminus \bigcup \mathcal{E}|+1} = (-1)^{|X|+|\bigcup \mathcal{E}|+1}$, since $\bigcup \mathcal{E} \subset X$.

We have shown for all X that

$$\sum_{\emptyset \neq \mathcal{E} \subset \mathrm{Bd}_X^-} (-1)^{|\mathcal{E}|+1} \sum_Y [X \setminus \bigcup \mathcal{E} \subset Y \subset X] \, (-1)^{|Y|} \zeta(Y)$$
$$= \sum_{\emptyset \neq \mathcal{E} \subset \mathrm{Bd}_X^-} [\bigcup \mathcal{E} \subset D] \, [(X \setminus \bigcup \mathcal{E}) \cap D = \emptyset] \, (-1)^{|\mathcal{E}|+|\bigcup \mathcal{E}|}.$$

The result of Lemma 3.15 follows by noting that for $X \subset D$

$$g_D(X) = \sum_Y [Y \cap D = \emptyset] \, g(X \cup Y)$$

and rearranging terms.

The coefficients $\nu(X)$ used in the statement of the lemma have already played a role in proving Theorem 3.16: the key part was showing that $|\nu(X)| \leq \binom{|\mathrm{Bd}_X^-|}{\lceil |\mathrm{Bd}_X^-|/2 \rceil}$ for disjunctions ϕ. A natural question then is, how large can $|\nu(X)|$ be for general queries? To answer this question, we rearrange the sum as

$$\nu(X) = (-1)^{|X|} \sum_Y [Y \subset X] \, (-1)^{|Y|} \zeta(Y) \sum_{\emptyset \neq \mathcal{E} \subset \mathrm{Bd}_X^-} (-1)^{|\mathcal{E}|+1} \, [X \setminus \bigcup \mathcal{E} \subset Y]. \tag{3.17}$$

Denote by S the innermost sum. We can rewrite it in the form

$$S = \sum_{\emptyset \neq \mathcal{E} \subset \mathrm{Bd}_X^-} [X \setminus Y \subset \bigcup \mathcal{E}] \, (-1)^{|\mathcal{E}|+1}, \tag{3.18}$$

which is seen to be an inclusion-exclusion sum over the upwards-closed subfamily

$$\left\{ \mathcal{E} \subset \mathrm{Bd}_X^- \;\middle|\; \bigcup \mathcal{E} \supset X \setminus Y \right\} \tag{3.19}$$

of the powerset of Bd^-_X. We have for $|S|$ an upper bound of $\binom{m}{\lceil m/2 \rceil}$, where $m = |\mathrm{Bd}^-_X|$, by applying Lemma 3.13 to this sum. Combining this with the fact that $\zeta(Y)$ is always 0 or 1, we obtain

$$|\nu(X)| \leq 2^{|X|-1} \binom{|\mathrm{Bd}^-_X|}{\lceil |\mathrm{Bd}^-_X|/2 \rceil}.$$

We thus have the following analogue of Theorem 3.16.

Theorem 3.22. *For an arbitrary query ϕ, the absolute error $|e(\phi)|$ of* TRUNCATE SUM *is bounded by*

$$2^{|U|-1} \binom{|\mathrm{Bd}^-|}{\lceil |\mathrm{Bd}^-|/2 \rceil} |\mathrm{Bd}^-|^{-1} \sum_{X \in \mathrm{Bd}^-} f(X).$$

The bound in the general case is even larger than the one in the disjunction case. The disjunctive bound is tight, but how close to the general bound can we come? Consider the sum (3.17). The form of the alternating sum over Y suggests that a parity-like function would be a difficult case: if $\zeta(Y) = 1$ if and only if $|Y|$ is even, the sum becomes

$$\nu(X) = (-1)^{|X|} \sum_{Y \subset X} [|Y| \text{ even}]\, S,$$

where S is the inclusion-exclusion sum (3.18). The bound for $|S|$ used Lemma 3.13, where it is easy to see that equality holds if the upwards-closed family (3.19) consists of those sets $\mathcal{F} \subset \mathrm{Bd}^-_X$ that have $|\mathcal{F}| = \lceil |\mathrm{Bd}^-_X|/2 \rceil$. But for $Y = \emptyset$ exactly this is achieved by the construction in the proof of Theorem 3.18. For larger sets $Y \subset X$, S is smaller, implying that this example falls short of the bound; however, the example suffices to show that if the statement of Theorem 3.22 is to be strengthened, one cannot simply decrease the general bound for $|S|$, but more careful analysis of the double sum (3.17) would be required.

3.7 Experimental evaluation with arbitrary queries

The results of the previous section indicate that the TRUNCATE SUM algorithm can give significantly worse results in the general case, compared to the disjunction case. However, from the experimental evaluation in Section 3.5 we see that the results in that case are much better than the theoretical bounds. Thus it is interesting to see how the algorithm performs on arbitrary queries.

Queries. It is obviously impossible to test any significant fraction of the $2^{2^{|U|}}$ possible queries. Instead, queries were generated randomly for each data set from several variations of a query model. In order to mimic queries that are likely to occur in practice, the models generate queries in conjunctive normal form, and are more likely to generate queries with relatively short normal forms. In the first model, which we term *Model 1*, a query is a conjunction of disjunctions of attributes; for example, the query

$$((A \vee B \vee C \vee D \vee E) \wedge (A \vee C \vee F \vee G) \wedge (H))$$

could be generated from this model. The number of disjunctions is distributed exponentially with parameter $\lambda = 1/5$; thus the average number of disjunctions is 5, but the number varies considerably for individual queries. The number of attributes in each disjunction is also distributed exponentially with parameter $\lambda = 1/5$, and each attribute is selected randomly from the set U; the selection is done without replacement within each disjunction, but the disjunctions are allowed to overlap. *Model 2* is otherwise similar, but it allows negations and can thus in principle generate any Boolean query. In each disjunction each attribute is negated with probability $1/2$. *Model 3* modifies *Model 1* by weighting the selection of attributes: the probability that an attribute A is picked is again proportional to $f(A)$. *Model 4* combines the modifications of *Models 2* and *4*: for each disjunction, the attributes are randomly negated, and they are picked with probability proportional to their frequency.

Algorithms. The algorithms evaluated were TRUNCATE SUM both with and without the border sets, and ITERATIVE PROPORTIONAL FITTING for queries with at most 13 attributes. The INDEPENDENCE algorithm is not directly applicable to general queries. Recall that the idea in this algorithm is to assume that $f(\bigvee D) = 1 - \prod_{A \in D}(1 - f(A))$. It could be generalized for arbitrary queries ϕ by approximating $g(X)$ in a similar manner for all $X \models \phi$ and summing these approximations (or approximating $g(X)$ for all $X \not\models \phi$ and subtracting from 1), but since there are in general an exponential number of summands, this was not considered an improvement upon ITERATIVE PROPORTIONAL FITTING.

Results. Figures 3.11 to 3.14 show an overview of all results. The general trend is that adding negations improves the results of all algorithms, and weighting the attributes by frequency makes the results worse. Overall, the TRUNCATE SUM algorithm performs much better on the sparse data sets than on the dense ones. This is natural, since the frequency thresholds that are usable on dense data sets are very high.

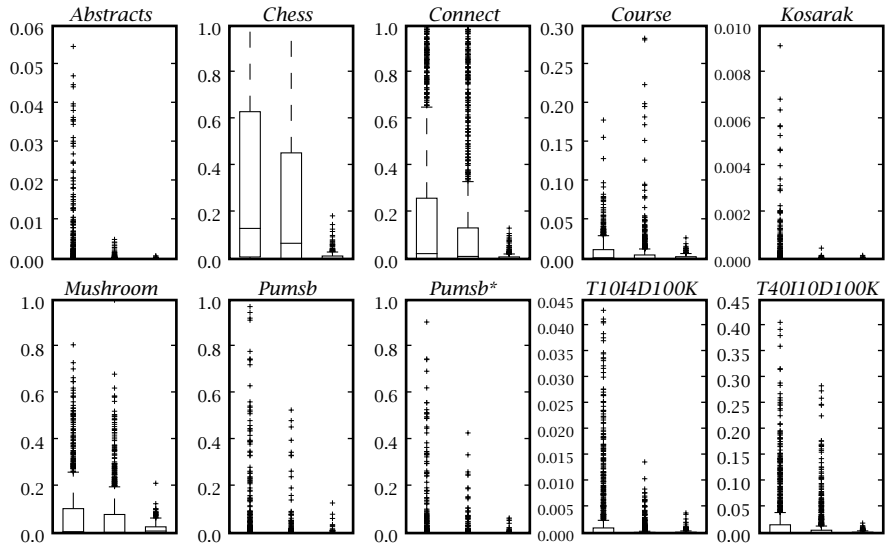


Figure 3.11. Results on *Model 1* in all data sets. In each subfigure, the first plot shows the absolute error of Truncate Sum without border sets, the second plot the results with the border sets included, and the third plot the absolute error of Iterative Proportional Fitting (for queries with at most 13 attributes, 36.6% of all queries on average).

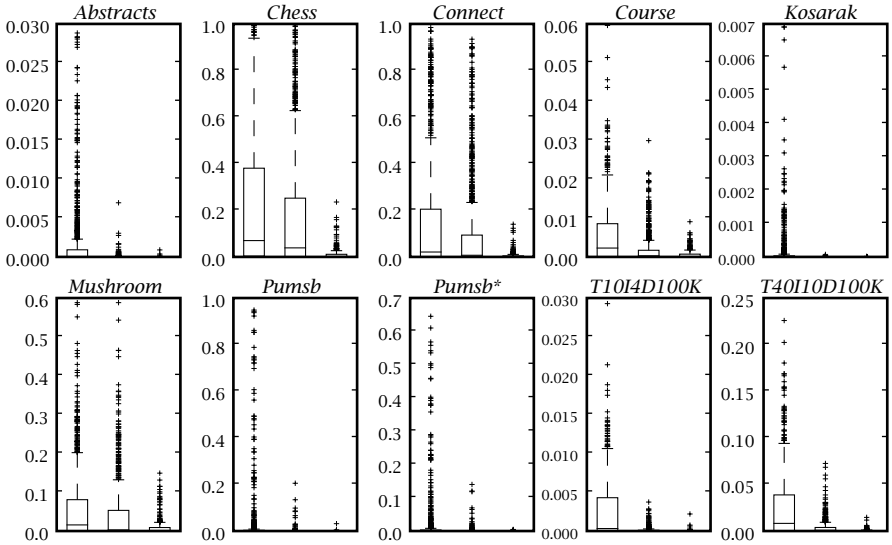


Figure 3.12. Results on *Model 2* in all data sets. As in Figure 3.11, the plots show the absolute errors of Truncate Sum without and with the border sets, and Iterative Proportional Fitting for the queries with at most 13 attributes, which is 36.2% of all queries on average.

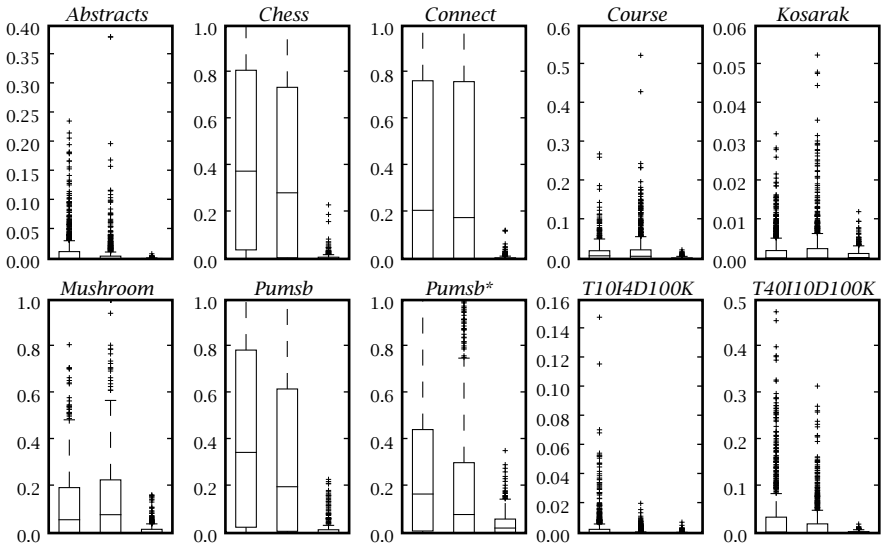


Figure 3.13. Results on *Model 3* in all data sets. As in Figure 3.11, the plots show the absolute errors of Truncate Sum without and with the border sets, and Iterative Proportional Fitting for the queries with at most 13 attributes, which is 36.7% of all queries on average.

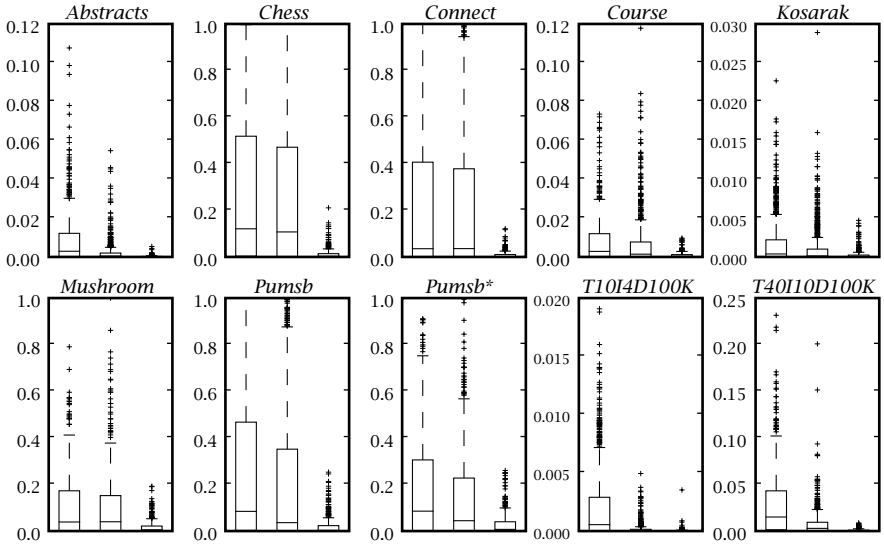


Figure 3.14. Results on *Model 4* in all data sets. As in Figure 3.11, the plots show the absolute errors of Truncate Sum without and with the border sets, and Iterative Proportional Fitting for the queries with at most 13 attributes, which is 36.5% of all queries on average.

With these query models there is no strong dependence between the number of attributes in a query and the error made by TRUNCATE SUM. This is evidenced by Figures 3.15, which shows the results of *Abstracts* as an example of a sparse data set, and 3.16, which shows the results of *Pumsb* as an example of a dense one. Since box-and-whiskers plots would be poorly visible with this range of x-coordinates, the interquartile range is simply drawn as a line, and points falling outside the range are plotted. Where there are at most five points at a single x-coordinate, all of them are plotted. Figures 3.17 and 3.18 show the (unclipped) errors of TRUNCATE SUM divided by the sum of the frequencies of the sets in the border on the same two data sets. Again we notice that these errors start out smaller than the theoretical bound and decrease rapidly with the increasing number of attributes.

Conclusions. The very large error bound of Theorem 3.22 is not even approached with these query models. Also the absolute errors are a lot of the time comparable to the threshold σ. The ITERATIVE PROPORTIONAL FITTING algorithm works very well but is only usable for a fairly small number of attributes in the query. Also at least the obvious generalization of the INDEPENDENCE algorithm would incur an exponential slowdown as a function of the number of attributes.

3.8 Correcting the truncation

In this section we present a fairly obvious modification of the TRUNCATE SUM algorithm motivated by two issues discovered in the experiments: the fact that adding information (the border sets) can degrade the results of TRUNCATE SUM, and the instability of the algorithm as a function of the threshold σ. We motivate the algorithm and give experimental results that show that while the algorithm does address the issues to some degree, it is not consistently better than TRUNCATE SUM.

Motivation. The TRUNCATE SUM algorithm essentially makes the assumption that the unknown frequencies are zero. In reality, they are of course some small numbers from zero to σ. Another reasonable assumption could be that the unknown frequencies are all σ', for e.g. $\sigma' = \sigma$ or $\sigma' = \sigma/2$. We thus have yet another algorithm, which we name TRUNCATE AND CORRECT: compute first $\hat{f}(\phi)$ according to TRUNCATE SUM, and then add a correction term to counteract the error caused by the non-frequent itemsets. Which exact value of σ' to use is not obvious. Note that for the upper bound construction of Theorem 3.18, a correction

using $\sigma' = \sigma$ would be right. In practice, we can try to strike a balance between the basic version of Truncate Sum and the one with the border sets included: let σ' be the largest frequency of the sets in the border. This makes the estimate of Truncate And Correct a continuous function of σ: when σ increases past the frequency of an itemset, this set is removed from the estimate of Truncate Sum but the correction term counterbalances the change.

To compute the correction, we use Corollary 3.7 to obtain the sum of all frequency coefficients, and then subtract the coefficients of frequent itemsets. In other words, we compute

$$\begin{aligned} e(\phi) &= \sum_X [X \in \mathcal{G}]\, \xi_\phi(X) \sigma' \\ &= \sigma' \Big(\sum_X [X \subset U]\, \xi_\phi(X) - \sum_X [X \in \mathcal{F}]\, \xi_\phi(X) \Big) \\ &= \sigma' \Big(\zeta_\phi(U) - \sum_X [X \in \mathcal{F}]\, \xi_\phi(X) \Big) \end{aligned}$$

where $\zeta_\phi(U)$ is computable by evaluating the query on an all-attributes tuple, and the remaining sum is easy to compute at the same time as the truncated sum.

Experiments. Similar experiments were performed for Truncate And Correct as for the other algorithms. Figures 3.19 and 3.20 compare the errors to those of Truncate Sum with and without the border sets on *Disjunctions 2* and *Model 4*; the other query sets had fairly similar results. Generally, it seems that Truncate And Correct has a smaller maximum error than Truncate Sum at the cost of increasing the error on the bulk of the results. However, in the most notable cases where including the border sets made the results worse (*Course* and *Kosarak*, *Disjunctions 2*) the new algorithm is a definite improvement. It is also better on some of the denser data sets, but not consistently. Its results on the two generated data sets are notably bad.

Figures 3.21 and 3.22 show the estimates for a few single queries as a measure of σ for the *Course* and *T40I10D100K* data sets and the two query sets shown in the preceding figures. The queries were selected using the difference in errors of Truncate Sum with the border sets and Truncate And Correct: in each row, the first subfigure shows the case where Truncate Sum won Truncate And Correct by the largest margin, the second the median case, and the third the case where Truncate And Correct won by the largest margin. The figures show that Truncate And Correct tends to behave in a more stable manner than Truncate Sum, even if its solution is not better.

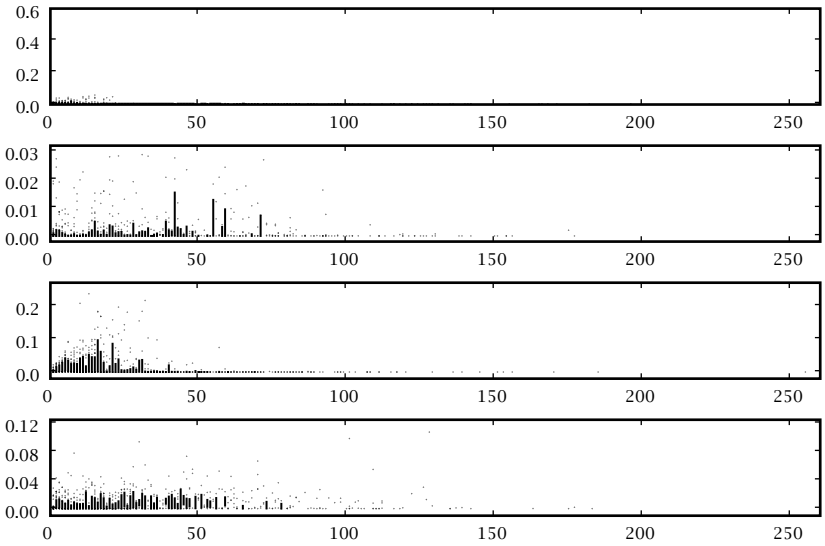


Figure 3.15. Dependence of error of Truncate Sum on the number of attributes in a query in *Model 1* through *4* for *Abstracts*. Number of attributes on x-axis, error (after clipping result of Truncate Sum to $[0, 1]$) on y-axis. Lines show the interquartile range, unless there are at most five points at a single x-coordinate. Other points are plotted. Note the varying scales of the subfigures.

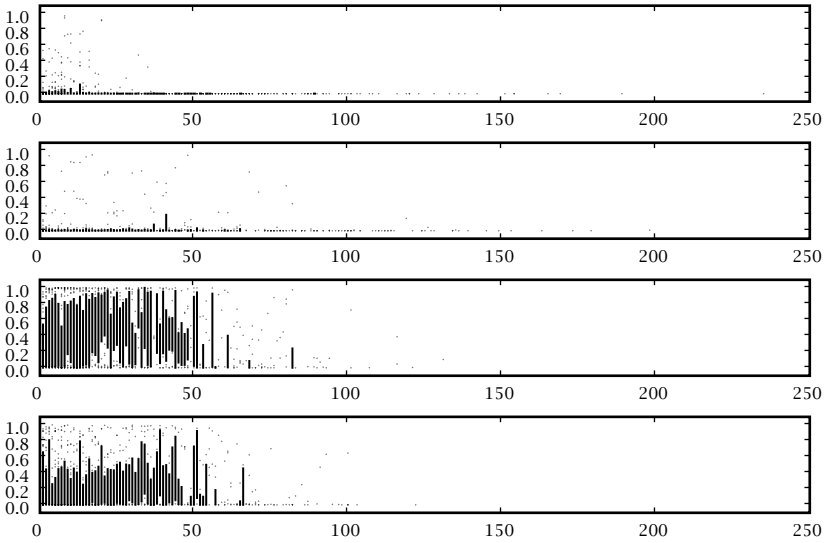


Figure 3.16. Dependence of error of Truncate Sum on the number of attributes in a query in *Model 1* through *4* for *Pumsb*. Number of attributes on x-axis, error (after clipping result of Truncate Sum to $[0, 1]$) on y-axis. Lines show the interquartile range as in Figure 3.15.

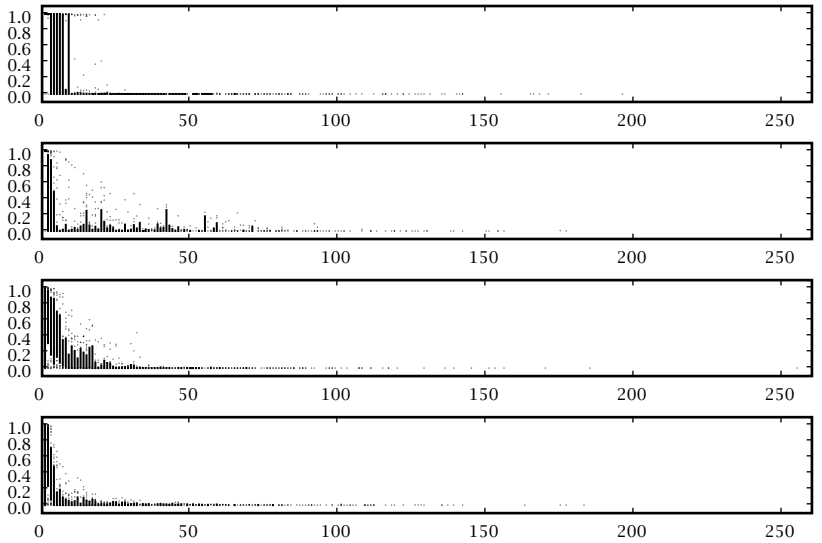


Figure 3.17. Error of Truncate Sum relative to border for *Model 1* through *4* for *Abstracts*. Number of attributes on x-axis, error relative to the sum of frequencies of sets in the border (as in Figure 3.6) on y-axis. Lines show the interquartile range as in Figure 3.15. Result of Truncate Sum is not clipped into $[0, 1]$.

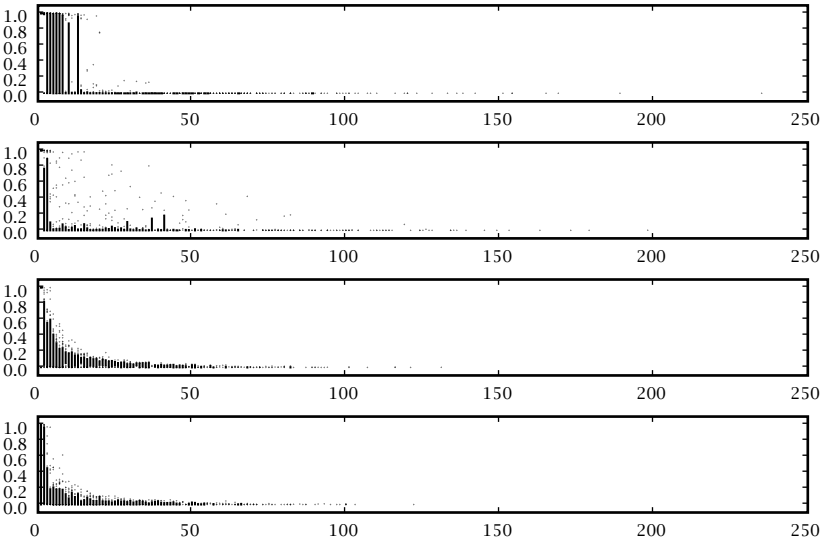


Figure 3.18. Error of Truncate Sum relative to border for *Model 1* through *4* for *Pumsb*. Number of attributes on x-axis, error relative to the sum of frequencies of sets in the border (as in Figure 3.6) on y-axis. Lines show the interquartile range as in Figure 3.15. Result of Truncate Sum is not clipped into $[0, 1]$.

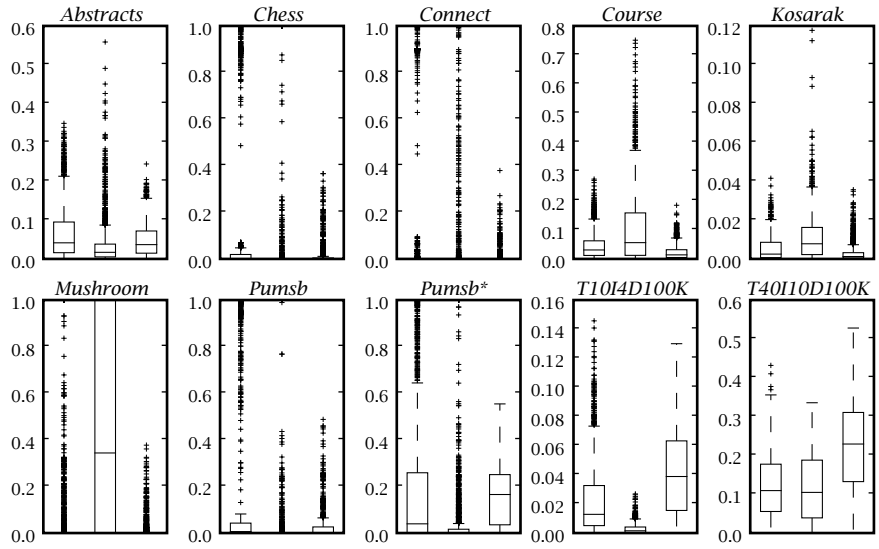


Figure 3.19. Error of Truncate And Correct on *Disjunctions 2* for all the data sets. In each subfigure, the leftmost column shows the error of Truncate Sum without the border sets, the second with the border sets, and the third the error of Truncate And Correct. Note the different scales in the subfigures.

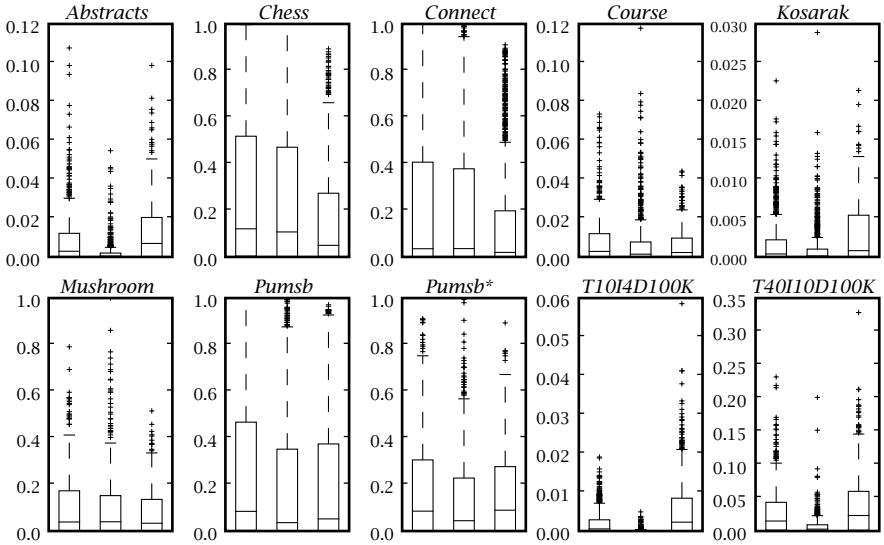


Figure 3.20. Error of Truncate And Correct on *Model 4* for all the data sets. In each subfigure, the leftmost column shows the error of Truncate Sum without the border sets, the second with the border sets, and the third the error of Truncate And Correct. Note the different scales in the subfigures.

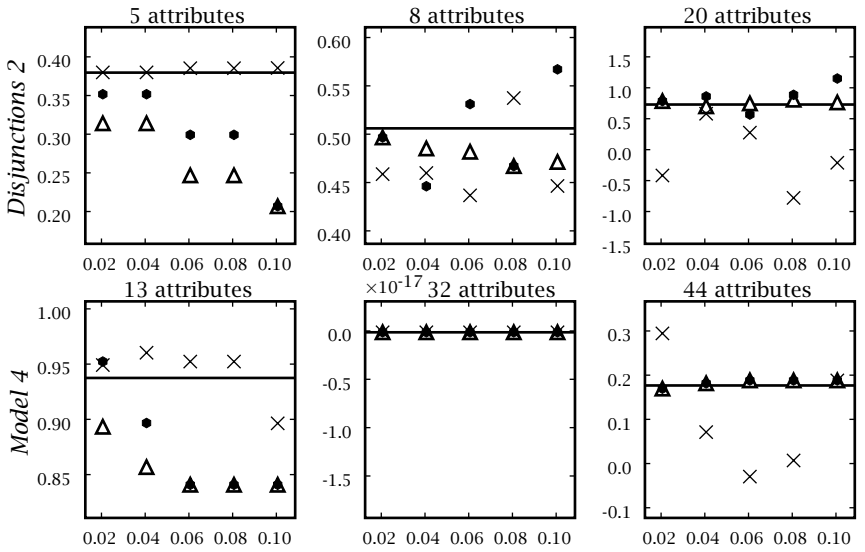


Figure 3.21. Estimates for single queries in *Course* with varying thresholds σ. Solid line: real frequency; circles: Truncate Sum; crosses: Truncate Sum with border sets; triangles: Truncate And Correct. Note the different vertical scales in subfigures, and see the text on how the queries were selected.

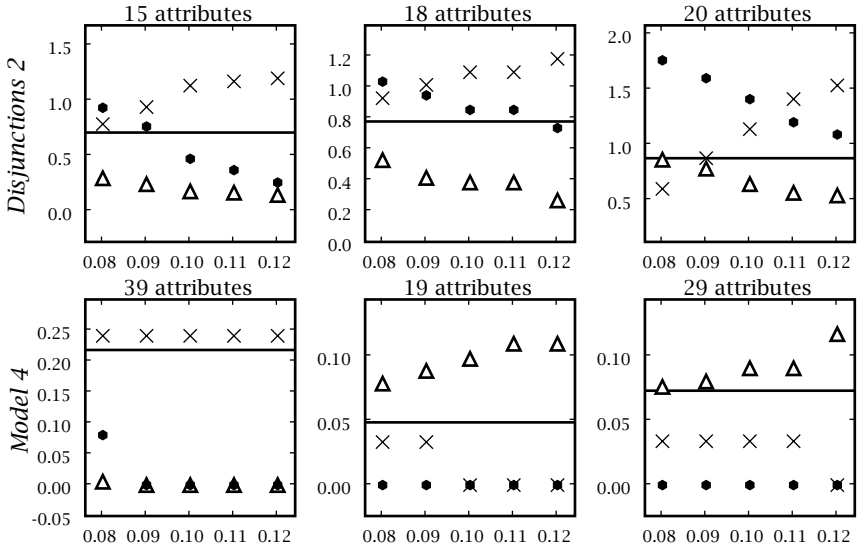


Figure 3.22. Estimates for single queries in *Course* with varying thresholds σ. Symbols as in Figure 3.21. Note the different vertical scales in subfigures, and see the text on how the queries were selected.

3.9 Other approaches to Approximate Query

In this section we briefly examine other possible approaches to the APPROXIMATE QUERY problem, both related work and possible future directions.

Given the frequencies of the frequent itemsets and a Boolean query, the approach we have taken is to start from a formula describing the query (the principle of inclusion and exclusion) and discarding the unknown terms. This sometimes leads to approximations that are outside the interval $[0,1]$ of possible frequencies. A question that has not been addressed in this chapter is that of what frequencies are possible for the query given the frequent itemsets. This question is easily formulated as a linear program, using the inner product representation:

$$\begin{aligned} &\text{minimize/maximize } \vec{\zeta_\phi} \cdot \vec{g} \\ &\text{subject to } \vec{f_\sigma} = A\vec{g} \\ &\text{and } \vec{1} \cdot \vec{g} = 1. \end{aligned}$$

Here $\vec{f_\sigma}$ is a vector of length $|\mathcal{F}_\sigma|$, containing the frequencies of frequent itemsets, and $A = (a_{X,Y})$ is a matrix with entries $a_{X,Y} = [X \subset Y]$. This approach was taken in [BSH02, BSH04], but in fact it has a long history, dating back at least to [Hai65]. There the main result is the use of linear programming to derive bounds for the probability of a Boolean formula over a set of variables, given the probabilities of the variables, corresponding in our case to the frequencies of single attributes.[1] In [Hai65] Hailperin also gives the generalization to the case where the input consists of the probabilities of arbitrary Boolean formulas (or upper and lower bounds thereof) and traces the ideas back to George Boole himself.

Solving the linear program numerically is, of course, infeasible for even moderate-sized data, since the length of $\vec{g}$ is exponential in the number of attributes. However, if the query ϕ contains a small number of attributes, the program may be reduced. Interestingly, it is not safe to drop all attributes that do not appear in ϕ, but in some conditions some attributes may be dropped [Tat06].

The theory of linear programming can also be applied to obtain various kinds of extensions of the Bonferroni bounds, e.g. [Kwe75, KM76, TM80, Pré88, Pré90, LN90, GP02, PG05]. All of these approaches attack the Bonferroni case where the frequencies of all k-wise conjunctions are known for $k = 1, \ldots, m$. However, they yield coefficients for the

[1] The three-variable case was solved in "about one minute" on an "electronic computer", but the four-variable case was deemed too complex.

frequencies that are not ± 1 and obtain better approximations than the truncated inclusion-exclusion sum. Similarly, Kessler and Schiff use a method that is not based on linear programming but still only handles the Bonferroni case [KS02]. It would thus be interesting if these methods could be generalized to the case of arbitrary frequent itemset collections.

An interesting approach is taken by Jaroszewicz et al. [JSR04], who use Bonferroni-type inequalities but recursively estimate the values that are missing from the Bonferroni case. A concern here is that using the information from, say, one ten-element itemset requires filling in estimates for $\binom{n}{10}$ other itemsets where n is the length of the query. However, in some contexts a solution like this could be useful.

The TRUNCATE SUM algorithm is combinatorial in nature. In Sections 3.5 and 3.7 we compared it to the iterative scaling algorithm, which is model-based: it finds the maximum-entropy distribution that can give rise to the observed frequencies. This can be seen as a Markov Random Field model [PMS00]. The curse of dimensionality is a problem for this approach, because the iterative scaling algorithm represents the probability distribution as a vector of $2^{|U|}$ numbers.

A simpler model was also seen in the experiments: if we use only the frequencies of attributes and apply maximum entropy, we obtain the independence model. This model ignores the higher-order information, but is applicable in the disjunction case to high dimensions.

An important open question for future research is whether a middle ground between the combinatorial and model-based approaches can be found. One way to seek such a compromise could be as follows: if we could detect that a family $\mathcal{F}_\sigma$ is structured so that TRUNCATE SUM yields a large error, but filling in a small number of the truncated frequencies would change the structure so as to reduce the error, we could do this filling-in by applying a probabilistic model, such as the independence assumption or suitably restricted maximum entropy.

A different route to better approximations is to change the setting of the problem: what if we were not restricted to using the frequencies of the frequent itemsets but could ask for a little extra information that could be gathered at the same time as the itemsets? Such information might be the exact frequencies $g(X)$, or the average number of attributes in a tuple containing X, or perhaps the parity functions

$$t(X) = \sum_Y [|Y \cap X| \text{ is odd}]\, g(Y).$$

As an example of using $g(X)$ as extra information, note that

$$f(\bigvee D) = \sum_{X \in M} g(X)$$

with

$$M = \{ X \subseteq U \mid X \cap D \neq \emptyset \}.$$

Therefore,

$$s \leq f(\bigvee D) \leq s + e,$$

where

$$s = \sum_X [X \in \mathcal{F}_\sigma \cap M]\, g(X)$$

and

$$e = \sum_X [X \text{ minimal in } M \setminus \mathcal{F}_\sigma]\, f(X).$$

The sum s could be used as the estimate for $f(\bigvee D)$, if the error e can be shown to be small. It will of course be large in the presence of noise, but with some sparseness assumptions this approach might be made practical.

More generally, we would like to divide the data into a large part where we can obtain good approximations and a small part that we can bound. For example, if most of the tuples contain few attributes, TRUNCATE SUM could be used on those, and the remaining tuples could be remembered as extra information.

Chapter 4

Dense itemsets

This chapter addresses a limitation of frequent itemsets: that they require all their attributes to co-occur perfectly, without any provision for errors or noise. For example, Figure 4.1 shows some binary data where there are three local patterns, highlighted in the figure. Such co-occurrence patterns are traditionally mined using frequent itemsets, but in this case the itemsets corresponding to the patterns have very low frequency: it is the occurrence of their various subsets that characterizes the patterns. The aim of this chapter is to generalize frequent itemsets to allow for this kind of patterns. Because of the ordering of rows and columns, the patterns are contiguous, and thus easily visible; however, we do not assume this contiguity to hold in real data, or indeed that the rows and columns have any natural ordering.

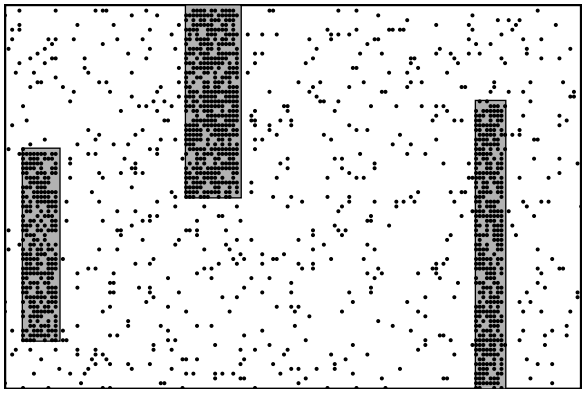


Figure 4.1. Synthetic data displaying three obvious local patterns not discoverable as frequent itemsets.

We start in Section 4.1 by developing a framework for talking about approximate frequency, where the central concept is that of *intersection counts*. We show in Section 4.2 how some existing definitions of approximately frequent itemsets fall into the framework, and discuss their shortcomings. We then define our new solution, *dense itemsets,* avoiding these shortcomings. We discuss algorithms for mining dense itemsets in Section 4.3, and show empirical results in Section 4.4. We show using these results how one can select a concise subfamily of the dense itemsets that describes a large part of the data. Finally, in Section 4.5 we discuss various possible extensions of the ideas.

4.1 Intersection counts

To generalize the concept of frequent itemsets, we first need to generalize the concept of subset containment. A tuple provides support to those itemsets that are its subsets; this definition must be generalized so that a tuple provides some kind of partial support to itemsets that are close to being its subsets. Such closeness needs to be measured in a numerical way, and a numerical definition of the subset relation is obtained via the intersection operator: a set I is a subset of J if and only if the intersection $I \cap J$ has the same cardinality as I. We thus define *intersection counts* as follows.

Definition 4.1. *Let I be an itemset over a relation r. Then the* intersection counts *of I in r are the numbers*

$$i_\nu = \sum_{T \in r} [|I \cap T| = \nu],$$

for $\nu \in \mathbb{N}$. We denote by $\vec{\imath}(I, r)$ the vector $(i_0, i_1, \ldots, i_{|I|})$.

That is, the intersection count i_ν of I in r is the number of tuples T in r such that $|I \cap t| = \nu$. To simplify examples, we use the convention that I and J are itemsets, $\vec{\imath}$ stands for $\vec{\imath}(I, r)$ and $\vec{\jmath}$ for $\vec{\imath}(J, r)$. The intersection count vector $\vec{\imath}$ summarizes the occurrence of I's items in the relation r. This summary loses some information in that it does not indicate *which* of I's items tend to occur in the relation.

An example of intersection counts is shown in Table 4.1. The column titled # in subtable (a) shows the number of times that the row in question is repeated in the data, and the other columns correspond to items. When there are only three attributes, it is possible to list all 2^3 possible tuples and give the number of repeats, but in general a vector of $2^{|U|}$ numbers is too large to be practical. In subtable (b), each itemset is displayed along

#	A	B	C
5	1	1	1
5	1	1	0
15	1	0	1
10	0	1	1
5	1	0	0
20	0	1	0
10	0	0	1
30	0	0	0

(a) Example data.

$\emptyset$ (100)		
A (70, 30)	B (60, 40)	C (60, 40)
AB (40, 50, 10)	AC (55, 25, 20)	BC (35, 50, 15)
	ABC (30, 35, 30, 5)	

(b) Intersection counts.

Table 4.1. Example intersection counts.

I	i_0	i_1	i_2	i_3
$\emptyset$	100			
	↓ ↘			
A	70	30		
	↓ ↘	↓ ↘		
AB	40	50	10	
	↓ ↘	↓ ↘	↓ ↘	
ABC	30	35	30	5

Table 4.2. Evolution of intersection counts as we add the attributes A, B, C to the empty set in the data of Table 4.1.

with its intersection counts: for example, since A appears in $5+5+15+5 = 30$ tuples, we have $i_1(A) = 30$, and since the number of tuples with exactly one of A, B, and C is $5 + 20 + 10 = 35$, we have $i_1(ABC) = 35$. The intersection counts are clearly easier to store than the repeat counts of all tuples, but they lose some information.

Notice that the intersection counts of a given set and its immediate superset cannot be very far apart. This is explained by the following simple observation: given two itemsets $I \subset J$ with $|J| = |I| + 1$, the database rows that contribute to i_ν must contribute to either j_ν or $j_{\nu+1}$. This is illustrated in Table 4.2, which shows the evolution of intersection counts as the items A, B, and C are added to the empty set. The arrows leading out from each count denote that each row contributing to that count must contribute to one of the numbers pointed to; likewise, each row contributing to a count with incoming arrows must contribute to one of the counts where the arrows originate.

4.2 Intersection statistics

Even though intersection counts compress the subset occurrence information exponentially, summaries that are single numbers would often be even more useful. In this section we define such a concept of *intersection statistics* and discuss several such statistics, among them two existing measures of approximate frequency. Then we derive a new measure that avoids some problems of the existing measures.

Definition 4.2. *Consider itemsets I over a relation r. An* intersection statistic *is any real-valued function defined on the intersection vectors $\vec{\imath}(I, r)$.*

Some simple examples of intersection statistics are $|I|$, the length of the vector minus one, and $|r|$, the sum of the vector. It is also easy to see that itemset frequency is an intersection statistic, as it can be defined as[1]

$$f(I) = i_{|I|}(I, r).$$

An abuse of notation that will often simplify formulas is to write statistics as functions of I and r directly:

$$f(I, r) = f(\vec{\imath}(I, r)).$$

The property of intersection counts that was illustrated in Table 4.2 yields a very simple proof of the monotonicity of itemset frequency: a diagonal line drawn under the last elements of the intersection count vectors only meets arrows that point pointing away from these elements. The same proof, which we formalize in the following lemma, can be used to show the monotonicity of a slightly more general intersection statistic, which we call *k-fault frequency* after [PTH01].

Lemma 4.3. *For any integer k, the k-*fault frequency *intersection statistic defined by*

$$\text{fault}_k(\vec{\imath}) = \sum_{\nu=|I|-k}^{|I|} i_\nu$$

is decreasing with respect to subset inclusion.

Proof. We break the definition of the intersection statistic into a sum over all tuples,

$$\text{fault}_k(I) = \sum_{T \in r} \sum_{\nu=|I|-k}^{|I|} [|T \cap I| = \nu].$$

[1] Frequency, like many intersection statistics, could be defined either absolutely (as a number of tuples) or relatively (as a fraction of tuples), as in Chapter 2. In this chapter we use the absolute variant, which simplifies many formulas.

We wish to prove that for any $J = I \setminus \{A\}$ for $A \notin I$,

$$\sum_{T \in r} \sum_{\nu=|I|-k}^{|I|} [|T \cap I| = \nu] \leq \sum_{T \in r} \sum_{\nu=|J|-k}^{|J|} [|T \cap J| = \nu].$$

Consider a tuple T whose intersection with I is of size $\nu \geq |I| - k$. If $A \in T$, then the intersection $T \cap J$ has size $\nu - 1$, and if $A \notin T$, the intersection $T \cap J$ has size ν. In both cases, T contributes to the right-hand side sum. (The reverse is not true: if $|T \cap J| = |J| - k$ and $A \notin T$, T does not contribute to the sum on the left-hand side. Thus equality does not hold in general.) □

Ordinary itemset frequency is the special case of 0-fault frequency. With larger values of k, we obtain a statistic that counts lines on which the itemset occurs with at most k omitted attributes. Itemsets fulfilling this property, together with another requirement (described in the next paragraph) were called *fault-tolerant itemsets* by Pei et al. [PTH01].

The monotonicity property allows using an APRIORI-like algorithm to compute fault-tolerant itemsets; however, this is not very practical. At the root of the problem is the fact that any set of at most k attributes has a k-fault frequency of 100%. Thus the algorithm must examine all itemsets of k elements in order to find the $(k + 1)$-element fault-tolerant itemsets. In fact, the extra requirement of Pei et al. is that each item appear on sufficiently many rows in the "fault-tolerant support" of the itemset, which allows them to restrict the search to frequent items. Even then, in most practical cases the search will be infeasible with large k.

If the problem with fault-tolerant itemsets lies in the fixed number k of errors allowed, it would seem reasonable to change the definition to allow a fixed *fraction* of errors. This leads to a very appealing concept that we call ϵ*-approximate frequency:* let $d = |I|$ and $c = \lfloor (1 - \epsilon)d \rfloor$, and define

$$f_\epsilon(I) = \sum_{\nu=c}^{d} i_\nu(I).$$

This definition has been used by Yang et al. [YFB01], in whose terminology an ϵ-approximately frequent itemset would be called a "strong error-tolerant itemset", or strong ETI for short. This approach also has at least two problems. First, it admits "free riders", an example of which is displayed in Table 4.3(a). In the depicted database, the itemset $ABCDE$ is a frequent one, so $\epsilon = 0$ would be a suitable value for discovering the itemset. If, however, we select $\epsilon = 0.17$, the sets $ABCDEF$, $ABCDEG$, and $ABCDEH$ will also be ϵ-approximately frequent, but the extra attributes do not reflect any pattern in the data—they have gained a free ride

A	B	C	D	E	F	G	H
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0

(a) The free-rider problem

A	B	C	D
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1

(b) The non-monotonicity problem

Table 4.3. Two example databases that illustrate two problems with different definitions of approximate frequency.

along with the actual pattern $ABCDE$. As ϵ increases, we get more and more sets: with $\epsilon = 0.5$, there are 187 itemsets, not subsets of $ABCDE$, that are ϵ-approximately frequent, and none of them tells us anything interesting about the data. Of course, in this example data the pattern is uniquely found using the correct value of $\epsilon = 0$, but this is no help in real data: first, the correct value is not known *a priori;* second, no single correct value may exist for a data set with many patterns at different noise rates.

Of course, it would be easy to remove attributes that only have zero values in the whole database, but this does not help. A more realistic variant of Table 4.3(a) would be one where the extra attributes F, G, and H do occur in some parts of the relation, just not usually together with the other attributes.

The second problem is illustrated by Table 4.3(b). The itemset $ABCD$ has 0.5-approximate frequency 6, but its subsets have lower frequencies: for example, the set ABC has half of its elements present on 3 rows only. Thus, ϵ-approximate frequency is not decreasing. This non-monotonicity becomes a problem if we want to use an APRIORI-style mining algorithm: to find 4-element sets at 0.5-approximate frequency σ, we must first find 3-element sets at 0.5-approximate frequency $\sigma/2$.

The non-monotonicity problem is essentially caused by the row-by-row definition of ϵ-approximate frequency: "half of 3" becomes 2, since a row cannot have a fractional number of items present. A natural solution, therefore, is to change the definition from a row-by-row basis to an average-of-rows one. This is a slightly more complicated concept than the previous ones, and is defined in two parts. The first part takes an average over the whole relation, and the second part localizes the average to the best part of the relation.

Definition 4.4. *The* weak density *of an itemset I* in a relation r *is the average fraction of attributes of I in the tuples of r:*

$$\operatorname{wdens}(I, r) = (|I| \cdot |r|)^{-1} \sum_{T \in r} |I \cap T|.$$

Thus, weak density in a relation is a measure of overall density of an itemset. It is an intersection statistic, having the form of a first moment:

$$\operatorname{wdens}(\vec{\imath}(I, r)) = (|I| \cdot |r|)^{-1} \sum_{\nu} \nu i_\nu.$$

The second part of the definition brings the frequency parameter into play: to mine local patterns, a support threshold is introduced. Again, in this chapter it simplifies formulas to use frequencies and thus thresholds σ that are integers from 0 to $|r|$ instead of fractions from 0 to 1.

Definition 4.5. *Let I be an itemset over relation r. The* weak density of I at support σ *is the maximum weak density in all subrelations of r that have size at least σ,*

$$\operatorname{wdens}(I, \sigma) = \max_{\substack{r' \subseteq r \\ |r'| \geq \sigma}} \operatorname{wdens}(I, r').$$

We next show that weak density is an intersection statistic, i.e., it can be computed from the intersection counts.

Proposition 4.6. *Weak density at support σ is an intersection statistic.*

Proof. Consider an intersection count vector $\vec{\imath}(I, r)$. Define the *inverse intersection count* i'_σ for every integer $0 \leq \sigma \leq |r|$ as the greatest integer such that

$$\sum [\nu \geq i'_\sigma]\, i_\nu \geq \sigma.$$

In other words, if we take all the rows whose intersection with I is at least of size i'_σ, the number of rows is at least σ, but if we leave out the rows with intersection size exactly i'_σ, the number of rows is less than σ.

For computing the weak density, we need a collection of exactly σ rows that have maximal intersection with I. In that collection we first include all rows with intersection size greater than i'_σ, and then top it up with rows of intersection size i'_σ until we have σ rows. Then the weak density at support σ is

$$\sigma^{-1} |I|^{-1} \left(\left(\sum_{\nu} [\nu > i'_\sigma]\, \nu \right) + \left(\sigma - \sum_{\nu} [\nu > i'_\sigma]\, i_\nu \right) i'_\sigma \right),$$

which is an intersection statistic. □

The reason for labeling this statistic "weak" density is that it suffers from the free-rider and non-monotonicity problems, just as ϵ-approximate frequency does. However, the situation is not quite as bad, since we can prove a weak form of monotonicity.

Proposition 4.7. *If an itemset X with $|X| \geq 2$ is weakly dense, then it has a weakly dense subset $Y \subset X$ with $|Y| = |X| - 1$.*

Proof. Let $r' \subseteq r$, $|r'| \geq \sigma$, be a subrelation where $\text{wdens}(X, r') \geq \delta$. Consider the frequencies in r' of singletons $\{A\} \subseteq X$; at least one of them must be at most average. Removing such a singleton yields a set $Y = X \setminus \{A\}$ with $\text{wdens}(\sigma, Y, r) \geq \text{wdens}(\sigma, Y, r') \geq \text{wdens}(\sigma, X, r') \geq \delta$. □

Corollary 4.8. *Every weakly dense itemset X is* accessible *by starting from the empty set and adding the items of X in some order, so that each intermediate itemset is weakly dense.*

Accessibility allows one to construct an algorithm for mining all weakly dense itemsets, but in any data set there are so many of these sets and even more candidates that need to be tested that the algorithm is not practical. This is partly due to the free-rider problem, which we avoid by the following definition.

Definition 4.9. *Let I be an itemset over relation r. The* strong density *of I in r is the minimum of the weak densities of all the subsets of I,*

$$\text{dens}(I, r) = \min_{J \subseteq I} \text{wdens}(J, r),$$

and likewise the strong density *of I* at support σ*,*

$$\text{dens}(I, \sigma) = \min_{J \subseteq I} \text{wdens}(J, \sigma).$$

Strong density is not an intersection statistic since it depends on the intersection counts of all subsets of the itemset in question.[2] However, strong density is easy to compute using an Apriori-like algorithm by keeping track of the minimum weak density of all subsets, as shown in Section 4.3. In the sequel, we will call strong density simply *density.*

Yang et al. attack the problems of free riders and non-monotonicity of strong etis (i.e., ϵ-approximately frequent sets) in a somewhat similar way: they define "weak etis", which are the same concept as our weakly

[2] As a simple example, if the itemset AB has an intersection count vector $(0, 100, 0)$, there is no way based on this fact alone to tell the difference between the situations where the itemset A has a vector of $(0, 100)$, or $(50, 50)$, or $(100, 0)$.

dense itemsets, and prove that the strong ETI property implies the weak ETI property. However, the free-rider problem makes finding all weak ETIs a daunting task despite the accessibility result, so Yang et al. resort to a heuristic algorithm that only discovers a subset of all weak ETIs and then prunes them for the strong ETI property.

The various intersection statistics covered in this section are summarized in Table 4.4.

Intersection statistic	Function of $\vec{\imath}(I)$
Size of relation, $\lvert r\rvert$	$\sum_\nu [0 \le \nu \le \lvert I\rvert]\, \imath_\nu$
Frequency, $f(I)$	$\imath_{\lvert I\rvert} \,/\, \lvert r\rvert$
k-fault tolerant frequency, $\text{fault}_k(I)$	$\sum_\nu [\nu \ge \lvert I\rvert - k]\, \imath_{\lvert I\rvert}$
ϵ-error tolerant frequency, $f_\epsilon(I)$	$\sum_\nu [\nu \ge (1-\epsilon)\lvert I\rvert]\, \imath_\nu$
Weak density in r, $\text{wdens}(I, r)$	$(\sum_\nu \nu \imath_\nu) \,/\, (\lvert r\rvert \cdot \lvert I\rvert)$
Inverse intersection count at support σ, $\imath'_\sigma$	$\arg\max_{\imath'_\sigma \in \mathbb{Z}} \sum_\nu [\nu \ge \imath'_\sigma] \ge \sigma$
Weak density at support σ, $\text{wdens}(I, \sigma)$	$\left[\left(\sum_{\nu > \imath'_\sigma} \nu \right) + \left(\sigma - \sum_{\nu > \imath'_\sigma} \imath_\nu \right) \imath'_\sigma \right] \Big/ \left(\sigma \lvert I\rvert \right)$

Table 4.4. Summary of intersection statistics.

4.3 Algorithms

In this section we consider the task of mining dense itemsets from data. The basic algorithm is a simple generalization of APRIORI, but we also give a more complicated top-k algorithm for setting one of the two parameters automatically.

Intersection counts are computed by Algorithm 4.1, a straightforward generalization of the database pass of APRIORI (Algorithm 2.2). Instead of one counter per itemset, it holds $|I| + 1$ counters, corresponding to the $|I| + 1$ values in the intersection count vector. From the intersection counts, it is easy to find the weak densities of itemsets using the result of Proposition 4.6. Algorithm 4.2 is an easy translation of the formula into pseudocode.

INTERSECTION-COUNTS(r, C)
Input: A binary relation r, a collection C of itemsets I
Output: The intersection count vectors $\vec{\imath}(I, r)$ of the given itemsets

for each $I \in C$:
  $\vec{\imath}(I, r) \leftarrow [0, 0, \ldots, 0]$
for each $T \in r$:
  for each $I \in C$:
    $S \leftarrow |T \cap I|$
    $i_S(I, r) \leftarrow i_S(I, r) + 1$
return $\vec{\imath}(I, r)$ for all $I \in C$

Algorithm 4.1. Find the intersection counts of itemsets.

WEAK-DENSITIES(r, C, σ)
Input: A binary relation r, a collection C of itemsets, a support threshold σ
Output: The weak densities at σ of the given itemsets

$H \leftarrow$ INTERSECTION-COUNTS(r, C)
for each $X \in C$:
  $N_{\text{rows}} \leftarrow 0$
  $N_{\text{items}} \leftarrow 0$
  for $j \leftarrow |X|$ **to** 0:
    $N_{\text{rows}} \leftarrow N_{\text{rows}} + H(X)[j]$
    $N_{\text{items}} \leftarrow N_{\text{items}} + j \cdot H(X)[j]$
    if $N_{\text{rows}} \geq \sigma$:
      $N_{\text{items}} \leftarrow N_{\text{items}} - j \cdot (N_{\text{rows}} - \sigma)$
      break
  if $N_{\text{rows}} < \sigma$:
    error "σ larger than $|r|$"
  else:
    $W(X) \leftarrow N_{\text{items}} / (\sigma \cdot |X|)$
return W

Algorithm 4.2. Find the weak densities of itemsets.

DENSE-SETS(r, δ, σ)
Input: A binary relation r, a density threshold δ, a support threshold σ
Output: All (σ, δ)-dense itemsets in r

$C \leftarrow \{ \{A\} \mid A \in U \}$
$\mathcal{F} \leftarrow \emptyset$
while C is nonempty:
  $W \leftarrow$ WEAK-DENSITIES(r, C, σ)
  $\mathcal{D} \leftarrow \{ X \in C \mid W(X) \geq \delta \}$
  $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{D}$
  $C \leftarrow$ GENERATE-CANDIDATES($\mathcal{D}$)
return $\mathcal{F}$

Algorithm 4.3. Find strongly dense itemsets.

Since strong density is decreasing with respect to set inclusion, we can easily modify APRIORI for use as the high-level algorithm for dense itemset mining, yielding Algorithm 4.3. This algorithm returns the weak densities of the itemsets; the strong densities are easily obtained from these, but not vice versa, and the weak densities may be useful for some of the filtering techniques of Section 4.5. The candidate generation subroutine is exactly the same as in APRIORI (Algorithm 2.3).

The APRIORI approach to mining dense itemsets has a weakness: it is very sensitive to the two parameters σ and δ. An example of this is given in Section 4.4. Searching for suitable values of the parameters for a given data set can be difficult. One way to solve this problem is to fix not the two parameters but only one of them, and start finding the itemsets for which the other parameter has the highest possible value. This is done by Algorithm 4.4 in the case of fixed σ. A corresponding algorithm for fixed δ is symmetric. The algorithm was partially motivated by [HWLT02, TYH03].

Algorithm 4.4 maintains a collection of itemsets to be explored next. They are stored on a heap (i.e., priority queue) H, whence the operation HEAP-POP removes and returns the set X that has the highest density. Whenever a set X is added to the result $\mathcal{F}$, new candidates are generated by the subroutine GENERATE-CANDIDATES$'$. It can in principle be almost the same as Algorithm 2.3, with the exception that it must search among all sizes of itemsets. However, a simple optimization is to only pair the recently-added set X with other sets, since any newly introduced candidates must have X as a subset.

TOP-K-GIVEN-SUPPORT(r, σ, k)
Input: A binary relation r, a support threshold σ, and a number k
Output: Some k itemsets in r such that they are the (σ, δ)-dense itemsets for the given σ and some δ

$\mathcal{F} \leftarrow \{\emptyset\}$, $\delta \leftarrow \infty$, $H \leftarrow$ empty heap
$C \leftarrow$ all single-item sets
while $k > 0$:
    $W \leftarrow$ DENSITIES(r, C, σ)
    for each $X \in C$:
        HEAP-PUSH($H, W(X), X$)
    $(d, X) \leftarrow$ HEAP-POP(H)
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{X\}$
    $\delta \leftarrow \min(\delta, d)$
    $C \leftarrow$ GENERATE-CANDIDATES$'(\mathcal{F}, X)$
    $k \leftarrow k - 1$
if consistent answer is required:
    repeat:
        $(d, X) \leftarrow$ HEAP-POP(H)
        if $d < \delta$:
            break
        else:
            $\mathcal{F} \leftarrow \mathcal{F} \cup \{X\}$
            $C \leftarrow$ GENERATE-CANDIDATES$'(\mathcal{F}, X)$
return $\mathcal{F}$

Algorithm 4.4. Find top-k dense itemsets given σ.

There is a slight complication in that there might not be a family of (σ, δ)-dense itemsets that has exactly size k and some density threshold δ. This is because weak density is not monotonic, and once the kth itemset X is found, there can be candidates left whose weak density is higher than (or equal to) that of X, and thus equal strong density. The last part of the algorithm finds the "extra" sets that have weak density at least δ.

4.4 Empirical results

This section describes results from experiments using the algorithms described in the preceding section. The algorithms were implemented in the Python language, using version 2.4 of the interpreter. Experiments were performed on the data sets of Chapter 2.

As an example of the difficulties that lead to Algorithm 4.4, Table 4.5 shows how the number of dense itemsets varies with σ and δ in the *Mushroom* data set. Missing values in the table indicate that the run was taking too long and was interrupted. Obviously, the number of dense itemsets is highly sensitive to the two parameters. However, there are some values of the parameters for which the number of dense itemsets is reasonable.

To test Algorithm 4.4, first APRIORI was used to determine for each data set the value of σ that yields 100 frequent itemsets. Algorithm 4.4 was then run with this value of σ and $k = 1000$ to obtain at least 1000 dense itemsets. Table 4.6 shows the resulting values of σ, δ, the number of dense itemsets found[3] and the size of the border, i.e., the number of minimal non-dense sets, which the algorithm must process in addition to the dense sets. The effect of the border sets is considerable in some of the data sets. In *Abstracts* and *Kosarak* the border is naturally large, because it includes all one-element sets, but the border in the generated data sets (*T10I4D100K* and *T40I10D100K*) consists of larger itemsets. In contrast, the effect due to the "extra" sets over $k = 1000$ is small across all data sets.

Figures 4.2 and 4.3 plot the weak density of itemsets X against the weak densities of their immediate subsets $X \setminus \{A\}$ for all $A \in X$ in the same two data sets as in the previous figures. The sets in *Course* seem to have more uniform structure. Adding items to a set is seen to typically decrease its weak density, but in some cases the density is increased.

Course data and set cover. We examine more closely the *Course* data, which contains information about course enrollment at the University of Helsinki Computer Science Department. Mining dense itemsets with $\sigma = 176$ (5% of $|r|$), $\delta = 0.75$ we found 10243 sets, the largest of which had 11 items. To better understand this large collection of itemsets, we selected a small subcollection using a greedy set cover approach.

In the set cover approach, we first select the dense itemset that covers the largest number of items in a maximal subrelation where the itemset

[3] As explained in the description of Algorithm 4.4, one usually obtains more than k sets, since no collection of exactly k sets might be a consistent result for a single value of δ.

	δ					
σ	0.5	0.6	0.7	0.8	0.9	1.0
8124	8191	255	31	31	15	1
7312		1791	127	31	31	9
6450		20479	575	95	31	25
5687			4607	255	79	31
4875			18335	1663	207	51
4062				6559	895	153

Table 4.5. Example of the sensitivity of dense itemsets to the parameters: number of dense itemsets in *Mushroom*. The data set has 8214 tuples, and σ varies from 0.5×8124 to 8124 with 0.1×8124 increments. Empty entries signify that the computation was interrupted because it was taking too long.

Data set	σ/ $\lvert r \rvert$	δ	Dense sets + border
Abstracts	14622/128820	0.748	1000 + 31562
Chess	3021/ 3196	0.977	1023 + 82
Connect	66443/ 67557	0.994	1007 + 125
Course	210/ 3506	0.829	1044 + 481
Kosarak	22466/990002	0.683	1003 + 41733
Mushroom	4464/ 8124	0.854	1023 + 145
Pumsb	46972/ 49046	0.990	1007 + 2157
*Pumsb**	30952/ 49046	0.892	1014 + 2150
T10I4D100K	2614/100000	0.566	1009 + 29762
T40I10D100K	9328/100000	0.664	1006 + 25868

Table 4.6. Parameters found for all data sets with Algorithm 4.4. Parameter σ was chosen by using Apriori to find the threshold where the number of frequent itemsets is 100. Parameter δ was then chosen with Algorithm 4.4 with the chosen σ and $k = 1000$.

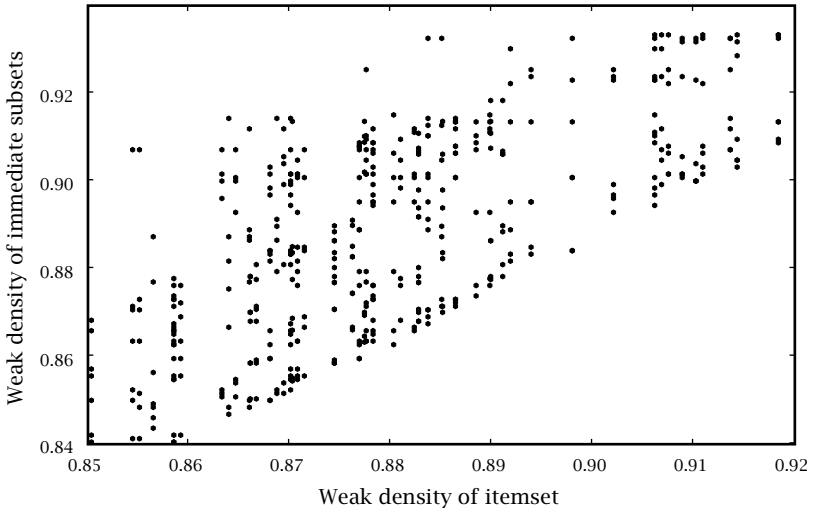


Figure 4.2. Weak density of itemsets X with $|X| \geq 7$ on x-axis, weak densities of immediate subsets $X \setminus \{A\}$ on y-axis, in *Course* data.

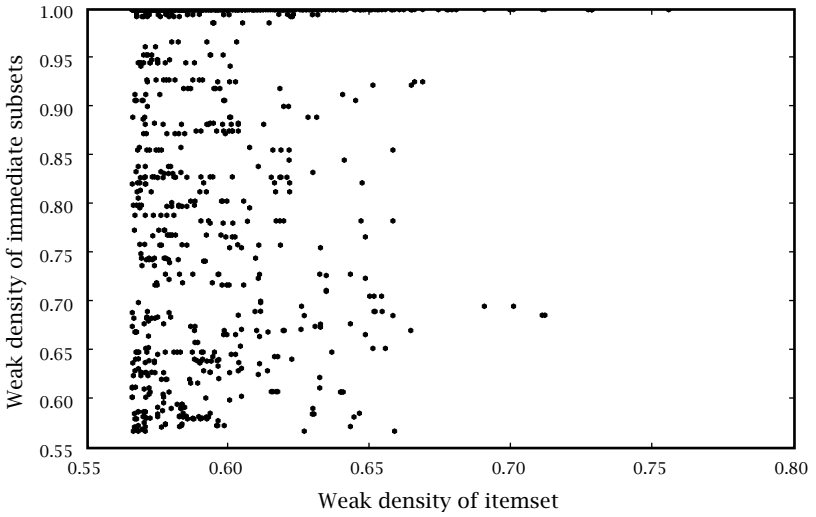


Figure 4.3. Weak density of itemsets X with $|X| \geq 2$ on x-axis, weak densities of immediate subsets $X \setminus \{A\}$ on y-axis, in *T10I4D100K* data.

	Itemset			
Course	1	2	3	4
Introduction to the use of computers	*			
Introduction to Databases	*			
Introduction to Programming	*			
Programming in Java	*			
Introduction to Application Design	*			
Software Engineering		*	*	
Data Structures		*		
Models for Programming and Computing		*	*	
Data Communications		*	*	
Database Management		*		
Database Application Project		*		
Concurrent Systems		*	*	
Data Structures Project			*	
Tutoring			*	
Operating Systems				*
Computer Organization I				*

Table 4.7. Four "best" itemsets found in *Course* using greedy set cover.

has (weak) density $\delta = 0.75$. The items of the set are then removed from the tuples in the subrelation, and the next itemset is selected. This is continued until most items have been removed. The four first itemsets found cover 39% of the all items, and are listed in Table 4.7. The first itemset contains introductory courses; the second and third contain slightly more advanced courses that are common to most subprogrammes at the Department, with the second itemset slightly oriented toward theoretical and the third toward practical courses; the fourth itemset has two courses that discuss low-level knowledge about operating systems. Most of the itemsets that are selected after these four contain only one course, with some two-course exceptions. Thus, the introductory and otherwise popular courses dominate the dense itemsets found.

In order to discover more interesting details of the students, another data set was produced, restricting to advanced courses, where the students have greater freedom of choice. This data set contains information on 1739 students (tuples) and 102 courses (items).[4] Dense itemsets were mined with parameters $\sigma = 87$, $\delta = 0.5$, producing 4657 sets. The set cover algorithm was run again, but in order to localize the effects of removing items, the criterion for ordering itemsets was the number of

[4]This data was gathered and anonymized at a different time from the main *Course* data; thus it is not a subset. In particular, it contains many more courses that had been introduced between the times when the data were gathered.

	Dense itemset							
Course	1	2	3	4	5	6	7	8
Theory of computation	*	*	*		*	*	*	
Software architectures	*				*		*	
Software processes & quality	*				*			
Software testing	*							
Telecommunications II	*					*		
Computer hardware	*					*	*	
Operating systems II	*					*		
Distributed systems	*					*		
Structured documents		*				*		
DB structures & algorithms		*						
Data mining		*	*					
Data warehousing		*						
Large document collections		*						
Text mining		*						
Algorithm design & analysis			*					
Artificial intelligence			*					*
Computer graphics			*				*	
String algorithms			*					
Computer graphics				*	*			
User interfaces				*	*			*
Distributed operating systems				*			*	
Data management II				*				*
Information retrieval				*				
Computer-aided learning					*			
Database modeling						*		
Compilers							*	
Object-oriented programming								*

Table 4.8. Eight "best" itemsets found in advanced course data using greedy set cover.

items covered in σ tuples (instead of an arbitrarily large number of tuples with density δ).

The first eight sets output by this algorithm are shown in Table 4.8. The densities of these sets vary from 0.52 to 0.58. The first six sets have frequency 0, i.e., the items never co-occur in the database, the seventh occurs once and the eighth occurs twice. None of these sets would thus have been found by a frequent itemset mining algorithm with frequency threshold more than 0.12.

Some of the sets have clear meanings: the first one represents software engineering and operating systems, the second information systems, and the third algorithms. The fourth one is less obvious but equally interesting: the courses might be seen to have value for programming web applications. The set excludes the Theory of Computation course,

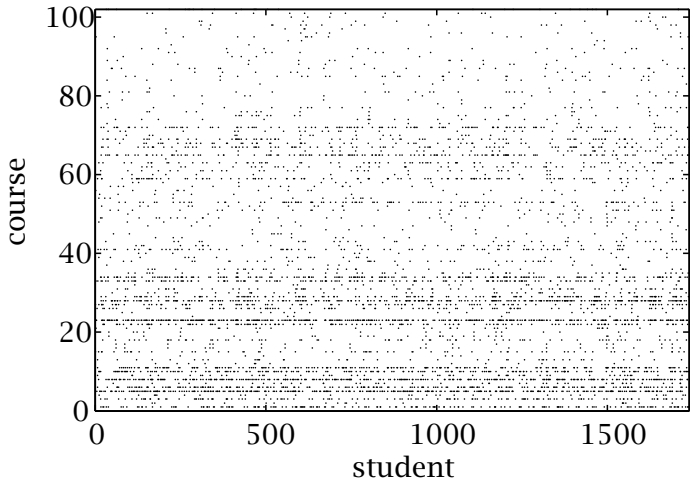
which all the preceding sets include; this course is required for graduation but is widely considered to be difficult. A possible conclusion is that some students have concentrated on web programming, and once they have the skills needed for employment, they forget about graduation and the Theory course.

As an illustration of the use of dense itemsets, we also used the greedy approach to reorder the data matrix. While executing the set cover algorithm, we reordered the rows and columns to keep the covered 1s as contiguous as possible; for rows and columns shared between different sets, ties were broken in favor of the "best" set, i.e., the one chosen earliest. Figure 4.4a shows the original data, and Figure 4.4b shows the resulting reordered data matrix.

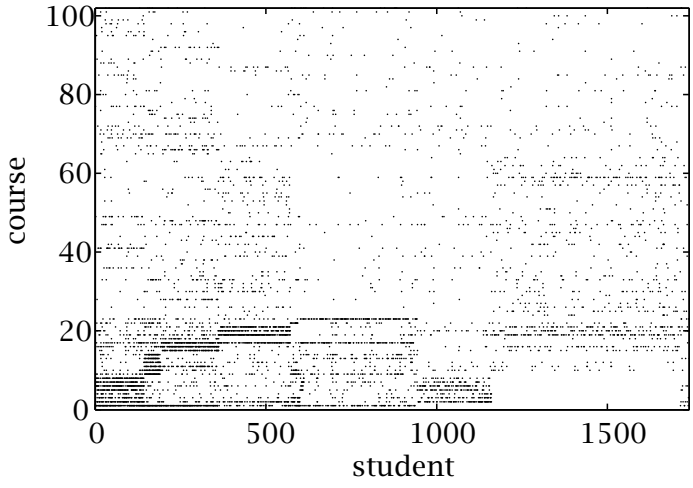
4.5 Discussion

When mining a data set for frequent patterns, be they frequent itemsets, dense itemsets, or whatever else, one often obtains a very large collection that itself is difficult to understand. An interesting solution is to approximate the collection by a smaller one [AGM04]. Another is to order the patterns so that each pattern conveys the most information given all its predecessors: when the user grows tired of looking at patterns, she will have as much information as possible [MM03]. In Section 4.4 we used essentially this kind of a method to obtain from the *Course* data the concise descriptions in Table 4.8 and Figure 4.4(b). Here we discuss briefly other possibilities for filtering dense itemsets, and close by mentioning some possible optimizations for mining the error-tolerant itemsets of [YFB01].

Density Gap. Consider two disjoint dense itemsets I and J, whose union $K = I \cup J$ is also dense. Should we include I and J in the output, or only K? A possible way to address this question is provided by Proposition 4.10 below. This proposition gives an upper bound for the weak density of K given the weak densities of I and J; if the actual density of K is close to the upper bound, this means that the two sets are highly correlated, but if it is much lower, even if over the threshold δ, we might keep the two disjoint sets. Computing the gap between the upper bound and the actual weak density is a simple postprocessing step to the mining algorithms. One could thus add another parameter γ to the algorithm: if a set has density gap $\geq \gamma$, do not output it. However, since the results are sensitive to the values of the two parameters, adding a third one could make it even more difficult to determine suitable values.



(a) Original data.



(b) Reordered data.

Figure 4.4. Advanced course data plotted as a matrix. Subfigure (a) shows original data in an arbitrary order, subfigure (b) reordered using the 8 itemsets in Table 4.8.

Proposition 4.10. *If I and J are itemsets with $I \cap J = \emptyset$,*

$$\operatorname{wdens}(\sigma, I \cup J) \leq \frac{|I| \operatorname{wdens}(\sigma, I) + |J| \operatorname{wdens}(\sigma, J)}{|I| + |J|}.$$

Proof. Let r_I, r_J, and $r_{I\cup J}$ be tuple-multisets of size σ where the weak densities of I, J, and $I \cup J$, respectively, are maximized. Thus

$$\operatorname{wdens}(I, r_{I\cup J}) \leq \operatorname{wdens}(I, r_I) = \operatorname{wdens}(\sigma, I),$$

so the number of attributes $A \in I$ contained in the rows of $r_{I\cup J}$ is at most $\sigma \cdot |I| \cdot \operatorname{wdens}(\sigma, I)$, and symmetrically for J. Thus

$$\begin{aligned}\sum_{T \in r_{I\cup J}} |(I \cup J) \cap T| &= \sum_{T \in r_{I\cup J}} \left(|I \cap T| + |J \cap T|\right) \\ &\leq |I| \operatorname{wdens}(\sigma, I) + |J| \operatorname{wdens}(\sigma, J),\end{aligned}$$

whence the claim follows. □

By induction we immediately obtain the following generalization.

Corollary 4.11. *If $I = \bigcup_{j=1}^{n} I_j$ is a partition of the itemset I, then*

$$\operatorname{wdens}(\sigma, I) \leq \left(\sum_{j=1}^{n} |I_j|\right)^{-1} \sum_{j=1}^{n} |I_j| \operatorname{wdens}(\sigma, I_j).$$

ROC Curves. A somewhat more principled approach to selecting "good" itemsets could be based on ROC curves.[5] As the fixed support threshold σ causes problems by forcing us to only select patterns of a fixed size, we will compare itemsets across all values of σ. A ROC curve plots true positives against false positives as functions of some diagnostic parameter. In our case, this parameter is σ, true positives are the 1s covered by the itemset in the σ best tuples, and false positives are the 0s covered in the same tuples. We can easily compute the points on the ROC curve using the intersection counts of a set: point number $k = 0, \ldots, |I|$ corresponds to covering

$$\sum [\nu \geq |I| - k] \nu$$

ones and

$$\sum [\nu \geq |I| - k] (|I| - \nu)$$

[5] The ROC abbreviation means either Receiver Operating Characteristic or Relative Operating Characteristic.

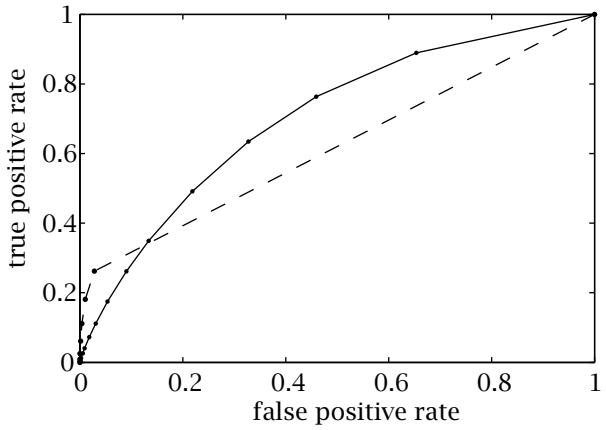


Figure 4.5. ROC curves in Course data: all-courses itemset (solid line), itemset 1 from Table 4.8 (dashed line).

zeros, and dividing the numbers by the total number of ones and zeros, respectively, yields the required coordinates. Since ROC curves are required to be convex and pass through $(0,0)$ and $(1,1)$, we add these points to the curve and take the convex envelope. A popular way of comparing ROC curves is the area under the curve, which is also known as the Wilcoxon statistic [Swe88].

Figure 4.5 shows the ROC curves of two different sets in the advanced *Course* data: the full itemset that includes all the courses, and the first "software engineering" itemset from Table 4.8. We see that the full itemset gets better than the engineering itemset at about a false-positive rate of 0.15. The area under the curve is clearly larger for the full itemset, a result that seems to indicate that this is not the best way of comparing single itemsets: a small itemset can never reach a large number of true positives, even if it has very few false ones.

It is also possible to define a ROC curve for a collection of itemsets in a similar way. Given the intersection counts of the sets in the family and all their intersections, we can compute the area under the ROC curve. Using this measure would allow for a more sophisticated comparison, where the all-items itemset could be outperformed by a good collection of itemsets. A drawback is that the computation time is exponential in the number of itemsets in the collection, so that only small collections could be compared. An interesting question for future research is whether there is a fast and useful upper-bounding method that could be used to search among collections of dense itemsets.

Binary Matrix Factorization. We next consider a task that can be seen as an adaptation of principal component analysis [DH73] or nonnegative matrix factorization [LS00] for binary data, and similar to ideas recently used by Geerts et al. [GGM04]. Given a sparse binary matrix M of dimensions $p \times q$, we wish to approximate it by a product of binary matrices R and S, $M_{p\times q} \approx R_{p\times k}S_{k\times q}$, with small internal dimension k. Thus for each $j = 1, 2, \ldots, k$, the jth column of R and jth row of S define a matrix that has only one block of 1s, and the approximation is a sum of these one-block matrices. (To allow for overlapping blocks, we interpret the sum as a "logical or" operation.) To evaluate results, we will use a score function computed over the 1s in RS: if M also has a 1, we add one point, and if M has a 0, we deduct one point. By extending the definition of intersection counts we can algorithmically find upper bounds for the scores of sets, given the extended counts of their subsets. We explain the idea using a small example.

Consider the 7-item itemset $I = ABCDEFG$. If only I is used to approximate M, we have $k = 1$, and the single row of S has 1s in the columns corresponding to the items in I. To optimize the score function defined above we must let those rows of the single-column matrix R have value 1 where approximating the corresponding row of M yields a positive score. This is true of those rows that contain sufficiently many 1s in the columns corresponding to I: if the number of 1s is 4, the contribution to the score is 1; if the number is 5, the contribution is 2; etc. Therefore the score of this approximation is

$$\text{score}(I) = i_4 + 3i_5 + 5i_6 + 7i_7.$$

Similarly, the score of I's 8-item superset $J = I \cup \{H\}$ is

$$\text{score}(J) = 2j_5 + 4j_6 + 6j_7 + 8j_8,$$

and we wish to bound this score without knowing the intersection counts of J. Define the *extended intersection counts* $E(I)$ as a two-dimensional array: $E(I)(\mu, \nu)$ counts the rows that contain exactly μ items of I and exactly ν items of the complement of I. It turns out that the score difference can be bounded using the extended profile of I:

$$\begin{aligned}\text{score}(J) - \text{score}(I) \le &\sum_{j=1}^{|R\setminus I|} E(I)(4,j) + \cdots + \sum_{j=1}^{|R\setminus I|} E(I)(7,j) \\ &- E(I)(4,0) - \cdots - E(I)(7,0).\end{aligned}$$

This allows a levelwise algorithm to find a family of sets (not necessarily dense sets) whose supersets are guaranteed to have a score below some

threshold. These could then be used as input to a greedy algorithm for the matrix-approximation problem. An obvious drawback is that storing the extended profiles requires quadratic space in the length of the itemset. Methods such as those in [GGM04] could allow us to find good approximations.

Decomposition Problems. The matrix-approximation problem suggests a general class of problems that could be called *decomposition problems*, as an extension of Kleinberg et al.'s *segmentation problems* [KPR04]. Given n functions $f_1, \ldots, f_n$, a set D of possible decisions, and an integer k, the general segmentation problem is to find a set of k decisions $x_1, \ldots, x_k$ to maximize the sum

$$\sum_{i=1}^{n} \max_{1 \le j \le k} f_i(x_j).$$

A concrete interpretation of the problem is that each function f_i represents the preferences of a customer, and the decisions are possible catalogs of items: a business can afford to print k different catalogs (but copies of the same catalog are cheap), so it clusters the customers into k segments and prints the optimal catalog for each segment. In a decomposition problem, the business again prints k catalogs but then sends a *combination* of catalogs to each customer. Again, the task is to choose k decisions $x_1, \ldots, x_k$, but what needs to be maximized is the sum

$$\sum_{i=1}^{n} \max_{J \subseteq [k]} f_i\Big(\bigoplus_{j \in J} x_j\Big), \tag{4.1}$$

where for each customer a *set* $J \subseteq [k] = \{1, 2, \ldots, k\}$ is chosen to maximize the benefit of the *sum* of the decisions in the set J. The meaning of the sum operator $\bigoplus$ may vary between different application domains, but when dealing with binary vectors the logical *or* operator suggests itself.

We define Catalog Decomposition as the following problem: Printing a catalog of r items for n customers costs $C + nr$ euros, where C is a constant. Given an advertising budget B and the preferences of all customers, send each customer a combination of catalogs to maximize the number of matches between catalog combinations and preferences. The greedy algorithm for Budgeted Maximum Coverage would favor catalogs I with a high value of

$$\alpha(I) = \frac{\text{items covered}}{C + nr} = \text{wdens}(I) \cdot \frac{1}{1 + C/(nr)}.$$

Thus we could feed the greedy algorithm with output from a modified version of Algorithm 4.3 where the density threshold is increased with

the size of the set. However, the greedy algorithm can be arbitrarily bad for BUDGETED MAXIMUM COVERAGE, and a better algorithm, such as those in [KMN99], is required; greedy optimization is an important part of those, but for even the $(1 - 1/e)/2$-guaranteed algorithm we also need to find the submatrix with the maximal number of ones.

Mining weakly dense itemsets. As noted in Section 4.2, accessibility is a weak form of monotonicity. It therefore allows a weak form of pruning: if all proper subsets J of a set I have $\text{wdens}(J, \sigma) < \delta$, then I cannot be weakly dense. Thus one can find all weakly dense sets in a levelwise manner: Start from the empty set. On level k, add as candidates all sets of size k which contain as a subset at least one weakly dense set of size $k-1$. This approach is essentially equivalent to the EXHAUSTIVE GROWING ALGORITHM of Yang et al. [YFB01].

As noted in [YFB01], a serious problem with this method is the number of candidates it processes. An optimization is provided by Proposition 4.10: since the algorithm has already computed the densities of all singletons, both $\text{wdens}(\sigma, I)$ and $\text{wdens}(\sigma, \{A\})$ will be known, and if $|I| \text{wdens}(\sigma, I) + \text{wdens}(\sigma, \{A\}) \leq (|I| + 1)\delta$, the set $I \cup \{A\}$ can be pruned without looking at the database. This optimization is, however, rather weak: for pruning to occur, a necessary but not sufficient condition is that the weak density of A is lower than δ. A stronger optimization is possible at a significant cost in both time and space: the algorithm can examine every partition $I = I_1 \cup I_2$ of I, and using the densities (or previously computed upper bounds thereof) of $I_j \cup \{A\}$ $(j = 1, 2)$, check if the upper bound $|I_1 + 1| \text{wdens}(\sigma, I_1 \cup \{A\}) + |I_2| \text{wdens}(\sigma, I_2)$ is lower than δ. Even this does not seem to suffice to make it practical to mine weakly dense itemsets.

Chapter 5

Tile models

This chapter introduces a class of *tile models*, which have a similar aim as dense itemsets: to find concentrations of attributes. However, with tile models we treat the data as a zero–one matrix, and seek submatrices whose density differs from their surroundings. This has two implications. First, matrices do not have attributes and tuples but rows and columns, which can be transposed. Second, we consider a submatrix interesting if its density is either smaller or larger than that of its surroundings. Thus, also the zeros and ones of the matrix are interchangeable.

Of the two kinds of tile models we consider, simplest to understand are *geometric tiles* where the submatrices are contiguous; these are described in Section 5.1. In the general case, the ordering of rows and columns may be irrelevant; then the *combinatorial tiles* of Section 5.2 are needed. In both cases, our models are hierarchical, meaning that also tiles within tiles are mined. We give empirical results in Section 5.3 and conclude in Section 5.4, where we also review related work.

Throughout this chapter, we consider a data matrix D with n rows and m columns, which are numbered $1, \ldots, n$ and $1, \ldots, m$, respectively. The value at the intersection of row i and column j is denoted by $D[i, j]$.

5.1 Geometric tiles

Consider the data matrix of Figure 5.1, where the ones and zeros are encoded as dots and empty spaces. The question “where are the dots?” can be answered in a fairly obvious way: there are two regions where most of the dots are, but inside one of the regions there is an almost-empty region. If the columns are web sites, arranged by the network proximity of the servers they reside on, and the rows are time intervals, then a

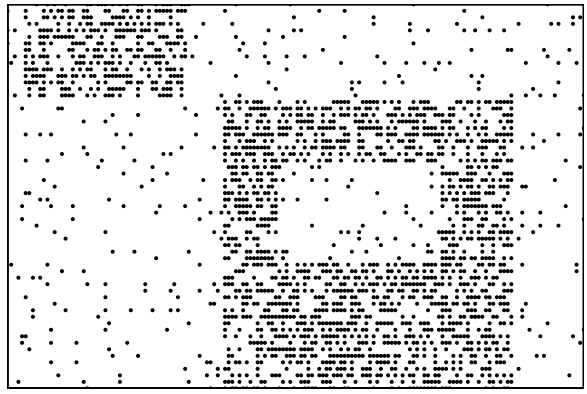


Figure 5.1. Example data matrix.

dot means that some number of users visited that site during that time interval. We can see two sets of nearby sites that have both been very popular at some time, except that some subset of one has fallen out of favor for a while (perhaps a network problem has affected a set of servers).

To find this kind of patterns automatically, we formulate a model for *geometric tiles*. Tiles are defined in terms of *rectangles*, which we define first.

Definition 5.1. *A* rectangle *of D is a pair (X, Y), where $X \subseteq [n]$ is a set of consecutive row numbers and $Y \subseteq [m]$ is a set of consecutive column numbers. A* proper subrectangle *of (X, Y) is a rectangle (X', Y') such that $X' \subseteq X$, $Y' \subseteq Y$, and $(X', Y') \neq (X, Y)$. Two rectangles (X, Y) and (X', Y') are* disjoint *if either $X \cap X' = \emptyset$ or $Y \cap Y' = \emptyset$ (or both). The coordinate pair (i, j)* falls within *(X, Y), if $i \in X$ and $j \in Y$.*

A rectangle coupled with a density parameter is a tile. Tiles may also contain other tiles, which may in turn contain yet other tiles. Thus the following definition is recursive, with the base case occurring when the set of contained tiles is empty.

Definition 5.2. *A* geometric tile *of D is a quadruple $\tau = (X, Y, p, \mathcal{T})$, where*

(a) (X, Y) is a rectangle of D, which we call the domain *of τ,*
(a) p is a number in $[0, 1]$, and
(a) $\mathcal{T}$ is a set $\{\tau_1, \ldots, \tau_k\}$ of geometric tiles, whose domains are pairwise disjoint proper subrectangles of (X, Y).

For modeling the whole data matrix D, we simply use a tile whose domain is D, which we call a *hierarchical geometric tile model*. For

example, the model depicted in Figure 5.1 is $\tau_0 = (X_0, Y_0, p_0, \{\tau_1, \tau_2\})$, where X_1 and Y_2 defines the whole rectangle, and the submodels are $\tau_1 = (X_1, Y_1, p_1, \emptyset)$ and $\tau_2 = (X_2, Y_2, p_2, \{\tau_3\})$, which has the single submodel $\tau_3 = (X_3, Y_3, p_3, \emptyset)$. To fill in the parameters p_j, we must define the semantics of tile models. To facilitate this definition, we introduce one more concept: the *exclusive domain* of a tile τ is the domain of τ with the domains of its subtiles removed.

Definition 5.3. *Let $\tau = (X, Y, p, \mathcal{T})$ be a tile and let the coordinate pair (i, j) fall within the domain of τ. If (i, j) falls within the exclusive domain of τ, the* prediction *of τ at (i, j), denoted $\tau[i, j]$, is defined to be p. Otherwise, the prediction of τ is defined recursively: $\tau[i, j] = \tau_k[i, j]$ for the sub-tile $\tau_k \in \mathcal{T}$ in whose domain (i, j) falls.*

By this definition, a hierarchical tile model τ has a prediction for every coordinate pair (i, j) of the data matrix. A good model should predict values $\tau[i, j]$ that are close to the actual data values $D[i, j]$. To quantify the goodness of a model, we interpret the prediction as a Bernoulli parameter, and thus we get the simple likelihood function

$$L(\tau \mid D) = \prod_{i,j} \tau[i, j]^{D[i,j]} (1 - \tau[i, j])^{1 - D[i,j]}. \tag{5.1}$$

In other words, the values in the data matrix are assumed to be independent, with the probability of the event $D[i, j] = 1$ given by the prediction $\tau[i, j]$. It is often easier to work with the log-likelihood,

$$\log L(\tau \mid D) = \sum_{i,j} \Big(D[i, j] \log \tau[i, j] + (1 - D[i, j]) \log(1 - \tau[i, j]) \Big). \tag{5.2}$$

How should we choose the parameters p_j of the example model to fit the data as well as possible, i.e., to maximize the likelihood (5.1)? The following result shows that we must simply take the average of each tile's exclusive domain.

Proposition 5.4. *To maximize the likelihood of a hierarchical (geometric) tile model when the domains of all tiles are fixed, each tile τ should have prediction p = the average of $D[i, j]$ for all coordinates (i, j) in the exclusive domain of τ.*

Proof. We will break down the log-likelihood (5.2) by the exclusive domain of each tile. Consider a tile τ with exclusive domain r and prediction p. The part of the log-likelihood contributed by τ is

$$\log p \sum_{(i,j) \in r} [D[i, j] = 1] + \log(1 - p) \sum_{(i,j) \in r} [D[i, j] = 0]$$
$$= N_1 \log p + N_0 \log(1 - p),$$

where N_1 is the number of 1s in r and N_0 is the number of 0s. The derivative with respect to p is

$$\frac{N_1}{p} - \frac{N_0}{1-p} = \frac{N_1 - p(N_0 + N_1)}{p(1-p)},$$

which is 0 if and only if

$$p = \frac{N_1}{N_0 + N_1} = \frac{1}{|r|} \sum_{(i,j) \in r} D[i,j],$$

which is the average claimed. If p is smaller, the derivative is positive, and if p is greater, the derivative is negative; thus, the log-likelihood attains a maximum at this value of p. □

The log-likelihood of a geometric tile that has no sub-tiles can be computed from the data in constant time, after a linear-time preprocessing step. The preprocessing, done by Algorithm 5.1, consists of finding the cumulative sums

$$D^+[i,j] = \sum_{\nu=0}^{i} \sum_{\mu=0}^{j} D[\nu,\mu]$$

for all $0 \le i \le n$, $0 \le j \le m$. (The values $D^+[0,j] = D^+[i,0] = 0$ help avoid special cases.) Once these sums are known, the number N_1 of ones inside a tile $\{a,\ldots,b\} \times \{c,\ldots,d\}$ is simply

$$\begin{aligned} N_1 &= \sum_{\nu=a}^{b} \sum_{\mu=c}^{d} D[\nu,\mu] \\ &= D^+[b,d] - D^+[b,c-1] - D^+[a-1,d] + D^+[a-1,c-1]. \end{aligned}$$

Then the log-likelihood of the tile is given by

$$\log L(X,Y) = N \cdot H(N_1/N), \tag{5.3}$$

where $N = |X| \cdot |Y|$ is the overall size of the tile, and H is the negative entropy function for a Bernoulli distribution, $H(p) = p \log p + (1-p)\log(1-p)$. We denote by GEOMETRIC-LOG-LIKELIHOOD the simple subprogram for computing (5.3).

We are now ready to describe our first algorithm for discovering geometric tiles. For the sake of simplicity, we only show algorithms for discovering a single tile with a high contribution to the log-likelihood of the model. A complete algorithm for finding hierarchical tiles must of course find several tiles, an issue addressed in Section 5.3. Also, when selecting a single tile, the algorithm must take into account the

CUMULATIVE-SUMS(D)
Input: A data matrix D of dimensions $n \times m$
Output: A cumulative-sums matrix D^+

for $\nu \leftarrow 0$ **to** n:
  $D^+[\nu, 0] \leftarrow 0$
for $\mu \leftarrow 0$ **to** m:
  $D^+[0, \mu] \leftarrow 0$
for $\nu \leftarrow 1$ **to** n:
  for $\mu \leftarrow 1$ **to** m:
    $D^+[\nu, \mu] \leftarrow D[\nu, \mu] + D^+[\nu - 1, \mu]$
      $+ D^+[\nu, \mu - 1] - D^+[\nu - 1, \mu - 1]$
return D^+

Algorithm 5.1. Compute cumulative sums for a data matrix.

log-likelihood of the surrounding tile. This is a simple modification to most of our algorithms. The number of rectangles in a data set of dimensions $m \times n$ is $\Theta(m^2 n^2)$. Thus an exhaustive search can be done in polynomial time, as illustrated by Algorithm 5.2.

However, with realistic-sized data sets, quadratic time can be too much. Thus we next describe a local search approach and prove that it can at least in some "easy" cases discover good tiles. The overall approach is shown in Algorithm 5.3. The variable b holds the best log-likelihood seen so far, and the variable B the corresponding tile. The algorithm tries moving all edges of the rectangle by one step, and selects the one with best log likelihood.

Naturally, nothing guarantees that the local search does not get stuck at a local optimum. However, we can prove the following result, which shows that when data matrices are sampled from a model and the size of the data approaches infinity, the method does have a high probability of ending up at the global optimum. Here we must take into account the log-likelihoods of both the tile and the rest of the matrix.

Lemma 5.5. *Assume that the data matrix D contains i.i.d. bits with probability q, with the exception of one geometric rectangle R, which contains i.i.d. bits with probability $p \neq q$ and whose number of rows and columns is a constant fraction of the number of rows and columns of the matrix D. Then, the local search method with random restarts finds R with high probability, i.e., probability bounded away from zero in the limit of infinite data.*

GEOMETRIC-EXHAUSTIVE-SEARCH(D)
Input: A data matrix D of size $n \times m$
Output: A geometric rectangle (X, Y)

```
D⁺ ← CUMULATIVE-SUMS(D)
best ← −∞
for X₁ ← 1 to n:
    for X₂ ← X₁ to n:
        for Y₁ ← 1 to m:
            for Y₂ ← Y₁ to m:
                L ← GEOMETRIC-LOG-
                    LIKELIHOOD(X₁, X₂, Y₁, Y₂, D⁺)
                if L > best:
                    best ← L
                    X ← { X₁, X₁ + 1, ..., X₂ }
                    Y ← { Y₁, Y₁ + 1, ..., Y₂ }
return (X, Y)
```

Algorithm 5.2. Examine all geometric rectangles and return one that supports a tile with maximum log-likelihood.

Proof. Since the rectangle R has width and height that are constant fractions of the data width and height, there is a constant probability that a random rectangle is either completely inside R, or completely contains R. We will argue that the algorithm then finds R with high probability.

Consider the case where the random starting rectangle, which we denote by r, falls completely within R. If $r \neq R$, at least one of its edges is not at the same position as the corresponding edge of R: e.g., the right edge of r is located to the left of the right edge of R. Let r' be the rectangle obtained by moving the right edge of r right by one step. We will show that replacing r by r' increases the log-likelihood of the model, which consists of the background tile and the single tile r.

In the limit of infinite data, we may assume that the proportion of ones inside R is p and the proportion outside is q. When the log-likelihood is written as a sum over all cells of the data matrix, the terms corresponding to the cells inside r do not change, since the proportion of ones inside r is the same as inside r'. The terms corresponding to the strip $r' \setminus r$ improve, since the proportion of ones inside the strip is p, and the prediction of the new model for them is p. The terms corresponding to the outside of r' also improve slightly, since the prediction of the new model is closer to q than the prediction of the old model. □

GEOMETRIC-LOCAL-SEARCH(D)
Input: A data matrix D of size $n \times m$
Output: A geometric rectangle (X, Y)

$D^+ \leftarrow$ CUMULATIVE-SUMS(D)
$(X_1, X_2) \leftarrow$ random numbers such that $X_1 \leq X_2$
$(Y_1, Y_2) \leftarrow$ random numbers such that $Y_1 \leq Y_2$
best $\leftarrow -\infty$
repeat:
  $L \leftarrow$ GEOMETRIC-LOG-LIKELIHOOD$(X_1, X_2, Y_1, Y_2, D^+)$
  if $L \leq$ best:
    return B
  best $\leftarrow L$
  $B \leftarrow (\{ X_1, \ldots, X_2 \}, \{ Y_1, \ldots, Y_2 \})$
  $C \leftarrow \{ (X_1 - 1, X_2, Y_1, Y_2),\ (X_1 + 1, X_2, Y_1, Y_2),$
    $(X_1, X_2 - 1, Y_1, Y_2),\ (X_1, X_2 + 1, Y_1, Y_2),$
    $(X_1, X_2, Y_1 - 1, Y_2),\ (X_1, X_2, Y_1 + 1, Y_2),$
    $(X_1, X_2, Y_1, Y_2 - 1),\ (X_1, X_2, Y_1, Y_2 + 1) \}$
  $(X_1, X_2, Y_1, Y_2) \leftarrow$
    $\arg\max_{R \in C}$ GEOMETRIC-LOG-LIKELIHOOD(R, D^+)

Algorithm 5.3. Find a locally optimal geometric tile.

5.2 Combinatorial tiles

Mining geometric tiles may be useful if the rows and columns of the data matrix have a natural ordering. This is rarely the case in general data mining, and therefore we extend the definition to allow for arbitrary submatrices, which we call combinatorial tiles. The added generality comes at the price of much slower mining algorithms. Therefore, we present a method for finding a natural order for the data, allowing the use of the algorithms designed for geometric tiles.

Definition 5.6. *A* combinatorial rectangle *of D is a pair (X, Y), where X is a set of row numbers $1, \ldots, n$ and Y is a set of column numbers $1, \ldots, m$. The concepts of* proper subrectangle, disjointness, *and* falling within *are defined identically to Definition 5.1.*

We also define a *combinatorial tile* identically to Definition 5.2, with the obvious exception that the domain of a tile can be any combinatorial rectangle. A *hierarchical combinatorial tile model* is then a combinatorial tile whose domain is the whole data matrix.

COMBINATORIAL-EXHAUSTIVE-SEARCH(D)
Input: A data matrix D of size $n \times m$
Output: A combinatorial rectangle (X, Y)

```
best ← −∞, X_best ← ∅, Y_best ← ∅
for each nonempty Y ⊂ {1,...,m}:
    D' ← D[1...n, Y]
    r ← ROW-SUMS(D')
    SORT(r)
    for ν to 1...n:
        for m ∈ {minimal, maximal}:
            X ← any ν rows whose values in r are m
            S ← LOG-LIKELIHOOD(X, Y, D)
            if S > best:
                best ← S
                X_best ← X, Y_best ← Y
return (X_best, Y_best)
```

Algorithm 5.4. Find best combinatorial rectangle.

Finding good combinatorial tiles is much more difficult than finding good geometric tiles, since there is an exponential number of rectangles to consider. Thus the most obvious counterpart to Algorithm 5.2 examines $\Theta(2^m 2^n)$ tiles. However, a simple trick allows us to reduce the number to $\Theta(2^m n)$. Of course, even this is not practical for most data sets. The trick is shown in Algorithm 5.4. It examines every set Y of columns, and takes first the submatrix D' that has the columns Y and all the rows in D. It computes the row sums of this submatrix, and sorts these numbers. It then uses the sorted numbers to select $2n$ sets of rows: one set consisting of one row with maximal sum, another consisting of two rows whose sums are as large as possible, another consisting of three such rows, etc; and symmetrically one row with minimal sum, etc. The sorting can be done in time $\Theta(n \log n)$, and computing the log-likelihood takes time $\Theta(|X| \cdot |Y|) = \Theta(mn)$, since the preprocessing of D to D^+ that was used in the case of geometric tiles is not usable for combinatorial tiles. Thus the time complexity of the algorithm is $\Theta(2^m mn^2)$.

The idea is similar to that of the intersection counts of Section 4.1. In fact, the log-likelihood of a single tile with fixed columns is an intersection statistic.

Theorem 5.7. *Algorithm 5.4 finds the tile with optimal log-likelihood.*

Lemma 5.8. *For any given sets X of rows and Y of columns, if either of the two changes described below is possible, the change improves the log-likelihood of the tile based on $X \times Y$.*

- *If the proportion of ones within $X \times Y$ is at most $1/2$, remove from X a row and add to X a row that has fewer ones within the columns Y than the removed row.*
- *If the proportion of ones within $X \times Y$ is at most $1/2$, remove from X a row and add to X a row that has more ones within the columns Y than the removed row.*

Proof. The size of the rectangle does not change as a result of removing one row and adding another: thus, only the number of ones in the rectangle changes. Denote by p the proportion of ones before the change and by p' the proportion after the change. We can write the log-likelihood as $N \times H(p)$, where N is the size of the rectangle, and H is the negative entropy (as in the discussion after Proposition 5.4). But the derivative $H'(p) = \log p - \log(1-p)$ is negative for $p < 1/2$ and positive for $p > 1/2$, so for $p' < p \leq 1/2$ we have $H(p') > H(p)$ and for $p' > p > 1/2$ also $H(p') > H(p)$. □

Proof of Theorem 5.7. The column-sets considered by the algorithm are obviously of the kind where the changes of Lemma 5.8 are impossible. They are also essentially all such sets: if there are several rows with equally many ones in the column-set Y, the algorithm does not consider every combination of these rows. But such rows are obviously interchangeable without changing the likelihood. □

Algorithm 5.5 is a direct translation of Algorithm 5.3 for combinatorial tiles. It is considerably slower for two reasons: first, in each step, there are $m+n$ choices to consider, instead of $O(\log(m+n))$ as in the geometric case; second, computing the likelihood of a geometric tile takes longer, since the preprocessing trick of partial sums $D^+[i,j]$ is not usable.

To speed up combinatorial tile mining, we next investigate an approach that reduces the task to geometric tile mining by reordering the database. The rows and columns are reordered separately. To reorder the rows, we first compute a similarity matrix S, whose entry s_{ij} is the similarity between rows i and j. The similarity measure can be any symmetric function, which should take high values for similar rows. We use dot-product and the Hamming overlap function as similarity measures.

COMBINATORIAL-LOCAL-SEARCH(D)
Input: input
Output: output

```
(X, Y) ← random rectangle
repeat:
    (X_best, Y_best) ← (X, Y)
    S ← LOG-LIKELIHOOD(X, Y, D)
    for r ← 0 to n − 1:
        X′ ← X ∪ { r } or X′ ← X \ { r }
        S′ ← LOG-LIKELIHOOD(X′, Y, D)
        if S′ > S:
            S ← S′, (X_best, Y_best) ← (X′, Y)
    for c ← 0 to m − 1:
        Y′ ← Y ∪ { c } or Y′ ← Y \ { c }
        S′ ← LOG-LIKELIHOOD(X, Y′, D)
        if S′ > S:
            S ← S′, (X_best, Y_best) ← (X, Y′)
    if (X_best, Y_best) = (X, Y):
        return (X, Y)
    else:
        (X, Y) ← (X_best, Y_best)
```

Algorithm 5.5. Search for a locally optimal combinatorial tile.

From the similarity matrix S, we first compute the Laplacian matrix $L_S = R - S$, where R is a diagonal matrix whose entries are the row sums of S, $r_{ii} = \sum_j s_{ij}$.

Lemma 5.9. [Fie73] *For any symmetric matrix S with nonnegative values, the Laplacian L_S is a symmetric positive semidefinite matrix.*

Proof. Let $\vec{x}$ be a vector; we will prove that $\vec{x}^T L_S \vec{x} \geq 0$. We have

$$\vec{x}^T R \vec{x} = \sum_i r_{ii} x_i^2 = \sum_i \Big(\sum_j s_{ij} \Big) x_i^2 = \sum_{i,j} s_{ij} x_i^2$$

and

$$\vec{x}^T S \vec{x} = \sum_{i,j} s_{ij} x_i x_j,$$

whence

$$\vec{x}^T L_S \vec{x} = \vec{x}^T R \vec{x} - \vec{x}^T S \vec{x} = \sum_{i,j} (s_{ij} x_i^2 - s_{ij} x_i x_j).$$

By using the symmetry assumption $s_{ij} = s_{ji}$, we can pair up the symmetric summands:

$$\vec{x}^T L_S \vec{x} = \sum_{i,j} [i < j] s_{ij}(x_i^2 + x_j^2 - 2x_i x_j) + \sum_i s_{ii} x_i^2$$
$$= \sum_{i,j} [i < j] s_{ij}(x_i - x_j)^2 + \sum_i s_{ii} x_i^2.$$

Thus the claim is proved. □

Lemma 5.9 implies that all eigenvalues of L_S are nonnegative. Since the row sums are zero, the all-ones vector $\vec{1}$ is an eigenvector with eigenvalue zero. The second smallest eigenvalue is more interesting, and is called the *Fiedler eigenvalue*. The corresponding eigenvector is the *Fiedler eigenvector*.

Lemma 5.10. [Fie73] *The Fiedler eigenvector $\vec{f}$ minimizes the expression*

$$\sum_{i,j} s_{ij}(f_i - f_j)^2$$

subject to the constraints $\sum_i f_i = 0$, $\sum_i f_i^2 = 1$.

Proof. Since $\vec{f}$ is an eigenvector of L_S, we know that $L_S \vec{f} = \lambda \vec{f}$, where λ is the Fiedler eigenvalue. Multiplying from the left by $\vec{f}^T$ yields $\vec{f}^T L_S \vec{f} = \lambda \vec{f}^T \vec{f}$. Since the scaling of eigenvectors is fixed so that $\|\vec{f}\| = \vec{f}^T \vec{f} = \sum_i f_i^2 = 1$, we have in fact $\lambda = \vec{f}^T L_S \vec{f}$. We know that λ is the second smallest eigenvalue, and that the smallest eigenvalue has $\vec{1}$ as the associated eigenvector, and thus that $\vec{f}$ must be orthogonal to $\vec{1}$. We can express these facts as a minimization task: the Fiedler eigenvalue λ minimizes $\vec{f}^T L_S \vec{f}$ subject to the constraints $\|\vec{f}\| = 1$ and $\vec{f}^T \vec{1} = \sum_i f_i = 0$. Under these constraints, minimizing the expression

$$\vec{f}^T L_S \vec{f} = \sum_i r_i f_i^2 - \sum_{i,j} f_i s_{ij} f_j = \text{constant} - \sum_{i,j} s_{ij} f_i f_j$$

is equivalent to minimizing

$$\sum_{i,j} s_{ij}(f_i - f_j)^2 = \sum_{i,j} s_{ij}(f_i^2 + f_j^2 - 2f_i f_j)$$
$$= \sum_{i,j} s_{ij}(f_i^2 + f_j^2) - 2\sum_{i,j} s_{ij} f_i f_j = \text{constant} - 2\sum_{i,j} s_{ij} f_i f_j,$$

which proves the result. □

We can exploit this result by using the Fiedler eigenvector coordinates to order the rows: row i appears before row j if $f_i < f_j$. We next show that in some simple cases the algorithm does indeed find a good ordering. We start with the statement of the result, then explain the assumptions and prove some lemmas, and finally prove the result.

Theorem 5.11. *For a "simple" tile model, with two row (column) blocks, as defined below, the spectral method will with high probability (over instantiations of the model) find an ordering in which the rows (columns) of all combinatorial tiles in the model become contiguous.*

A tile model is simply a tile whose domain is the whole data matrix, as explained after Definition 5.2. Now we are using the model as a *generative* one: if we generate a data matrix from the model, there is a high probability (i.e., a probability bounded away from zero) that we can order the matrix so that the tiles become geometric. Of course, we have to exclude models for which no ordering has this property. We also exclude models where the ordering is "hidden in the noise": the information about which tiles occur on the same rows and columns must be present in the pairwise similarities. To express this condition, we first define row and column blocks.

Definition 5.12. *A* row block *in a combinatorial tile model is a maximal set of rows that all intersect exactly the same tiles.* Column blocks *are defined symmetrically.*

For the blocks to be visible in the pairwise similarities, the similarity function must be able to differentiate between the blocks. Also, the blocks must not be too small. The reason for the restriction to two blocks is explained later, after Lemma 5.16.

Definition 5.13. *A combinatorial tile model is* simple *if it meets the following conditions:*

- *there exists an ordering of the data matrix in which all tiles in the model become geometric;*
- *each row block is at least of size αm;*
- *there exists a constant σ_k for each row block k, and a constant σ^*, such that the similarity s_{ij} between rows i and j fulfills $E[s_{ij}] \leq \sigma_k$, if both rows belong to block k, and $E[s_{ij}] \geq \sigma^*$, if the rows belong to different blocks, and $\sigma_k - \sigma^* \geq \sigma^\#$ for all k, with a constant $\sigma^\# > 0$;*
- *symmetric conditions hold for column blocks.*

The next tool we need is a characterization of corresponding blocks in similarity matrices.

Definition 5.14. *A similarity matrix $S = (s_{ij})_{1\le i\le n, 1\le j\le n}$ has* (k, d, s_1, s_2)-block structure *if the indices $1, \dots, n$ of the matrix can be partitioned in k blocks $B_1, \dots, B_k$ such that*

- *the size of each block is at least d;*
- *for all ℓ, for all $i, j \in B_\ell$ we have $s_{ij} = \tilde{s}_\ell \ge s_1$;*
- *for all $t \ne \ell$, for $i \in B_\ell$, $j \in B_t$ we have $s_{ij} = \tilde{s}_{\ell t} \le s_2$.*

A vector $\vec{x}$ respects *the block structure if there is no triple (i, j, h) with $x_i \le x_j \le x_h$ such that $i, h \in B_\ell$, $j \in B_t$, and $t \ne \ell$.*

First we prove a technical lemma, then use it to show that with some assumptions, block-structured matrices have Fiedler eigenvectors that respect the block structure. We denote by $F_S(x)$ the Fiedler function $x^T S x$.

Lemma 5.15. *Let S be any $n \times n$ similarity matrix for which $s_{1i} = s_{2i} = \cdots = s_{mi} = s_i$ for all $i = m+1, \dots, n$, and let $x = (x_1, x_2, \dots, x_n)$ be a vector satisfying $x^T e = 0$ and $x^T x = 1$. Consider the vector $x' = \alpha(y, \dots, y, x_{m+1}, \dots, x_n)$ obtained from x, where the first m coordinates have been replaced by their average $y = (x_1 + \cdots + x_m)/m$, and the vector has been scaled by α so that it satisfies $x'^T x' = 1$. Then x' satisfies also the constraint $x'^T e = 0$, and we have $F_S(x) \ge F_S(x')$.*

Proof. Notice that the constraint $x'^T e = 0$ is automatically satisfied. The scaling factor α must satisfy $m(\alpha y)^2 + \sum_{i=m+1}^{n} (\alpha x_i)^2 = 1$. Therefore

$$\begin{aligned}
\frac{1}{\alpha^2} &= m\Big(\frac{x_1 + \cdots + x_m}{m}\Big)^2 + \sum_{i=m+1}^{n} x_i^2 \\
&= \frac{1}{m}\sum_{i=1}^{m} x_i^2 + \frac{2}{m}\sum_{1\le i<j\le m} x_i x_j + \sum_{i=m+1}^{n} x_i^2 \\
&= 1 - \frac{m-1}{m}\sum_{i=1}^{m} x_i^2 + \frac{2}{m}\sum_{1\le i<j\le m} x_i x_j \\
&= 1 + \frac{1}{m}\Bigg[m\sum_{i=1}^{m} x_i^2 - \Big(\sum_{i=1}^{m} x_i\Big)^2\Bigg].
\end{aligned}$$

The expression in square brackets is nonnegative because of the Cauchy–Schwartz inequality:

$$\sum_{i=1}^{m} (1 \times x_i) \le \sqrt{\sum_{i=1}^{m} 1^2} \times \sqrt{\sum_{i=1}^{m} x_i^2}.$$

Therefore, $\alpha^2 \le 1$.

Now we have

$$F_S(x') = \sum_{1\le i,j\le m} s_{ij}(\alpha y - \alpha y)^2 + m\sum_{i=m+1}^{n} s_i(\alpha x_i - \alpha y)^2 + \sum_{m+1\le i,j\le n} s_{ij}(\alpha x_i - \alpha x_j)^2.$$

The first sum is of course zero. To bound the second sum, we use the inequality

$$\Big(w - \frac{1}{m}\sum_{i=1}^{m} x_i\Big)^2 \le \frac{1}{m}\sum_{i=1}^{m}(w - x_i)^2, \tag{5.4}$$

which we will prove below. We thus have

$$F_S(x') \le \alpha^2\Big(\sum_{i=m+1}^{n} s_i \sum_{j=1}^{m}(x_i - x_j)^2\Big) + \alpha^2 \sum_{m+1\le i,j\le n} s_{ij}(x_i - x_j)^2 \le \alpha^2 F_S(x).$$

Because $\alpha^2 \le 1$, we have proven the theorem.

It remains to prove inequality (5.4). Both sides contain the terms w^2 and $-(2w/m)\sum_i x_i$, so it suffices to prove that

$$\frac{1}{m^2}\Big(\sum_{i=1}^{m} x_i\Big)^2 \le \frac{1}{m}\sum_{i=1}^{m} x_i^2.$$

But this is the same inequality that we proved already. □

Lemma 5.16. *Let S be an $n \times n$ similarity matrix that has $(2, d, s_1, s_2)$-block structure. Consider $S' = S + E$, where E is a symmetric matrix. Then the Fiedler vector of $L_{S'}$ respects the structure of S, provided that $\|L_E\| = o(d(s_1 - s_2))$, where $\|L_E\|$ is the norm of the matrix L_E.*

Proof. We denote by x_i the coordinates of vector x that correspond to objects in the first block ($i \in B_1$) and by x_j the coordinates that correspond to the second block ($j \in B_2$). Without loss of generality we assume that $\tilde{s}_1 = \tilde{s}_2 = s_1 = s$ and $\tilde{s}_{12} = s_2 = 0$. In other words, objects within blocks have the same similarity value s and objects across blocks have similarity 0. We treat s as the difference $s_1 - s_2$.

The proof is structured as follows. First we find the Fiedler vector v of L_S, for which $F_S(v) = 0$ because of the assumptions made above. Then we show that if a vector b does not respect the block structure of S,

the Fiedler function is $F_S(b) \geq \Theta(ds)$, i.e., significantly higher than the one of v. Finally we show that perturbing S by E does not affect the Fiedler function too much, i.e., that any vector b not respecting the block structure of S has higher $F_{S'}(b)$ than the Fiedler vector v' of S'.

Consider the vector v with all coordinates v_i equal to a constant a and all coordinates v_j equal to another constant b. Constants a and b can be specified uniquely (up to renaming) so that $v^T e = 0$ and $v^T v = 1$. It is easy to see that v satisfies $F_S(v) = 0$, therefore v is a Fiedler vector of S. It is also clear that v respects the block structure of S since $a < b$ (or $b < a$).

Next consider a vector b with $b^T e = 0$ and $b^T b = 1$ that does not respect the block structure of S; we will call such vectors *bad*. A bad vector contains at least one violation of the block structure, e.g., $v_{i_1} \leq v_j \leq v_{i_2}$. We call a bad vector b_m *minimally bad*, if (*i*) in each of the two blocks, only one coordinate is involved in a violation, and (*ii*) in each block, all coordinates except the one involved in the violation are assigned the same value. From any bad vector b, we can reach a minimally bad vector b_m by performing the local merge moves of Lemma 5.15 that replace some coordinates by their average. The local merge moves do not increase the value of the Fiedler function, i.e., $F_S(b) \geq F_S(b_m)$. Furthermore, in a minimally bad vector, in at least one block, the coordinate involved in a violation is at least distance $\Theta(1)$ away from the other coordinates. This implies that $F_S(b_m) \geq \Theta(ds)$, and therefore $F_S(b) \geq \Theta(ds)$ for any bad vector b.

Finally, consider the Fiedler vector v' of $L_{S'}$, the Fiedler vector v of L_S, and any bad vector b. Notice that $F_{S'}(x) = x^T L_{S'} x = x^T L_S x + x^T L_E x$ for all vectors x. For any vector x with $x^T x = 1$ we have $0 \leq x^T L_E x \leq \|L_E\|$ by the definition of matrix norm. By the definition of v', we have $F_{S'}(v') \leq F_{S'}(v) = v^T L_E v \leq \|L_E\|$, while $F_{S'}(b) = F_S(b) + b^T L_E b \geq \Theta(ds)$. Therefore, assuming $\|L_E\| = o(ds)$, no bad vector can be a Fiedler vector of S'. □

This lemma concerns only matrices with two blocks. With a larger number of blocks, the setting becomes more complicated, since the Fiedler vectors form a subspace of dimension higher than one. A practical way to handle this problem is to apply spectral clustering recursively: use the Fiedler vector to separate the rows into two clusters only, and then subcluster the two clusters separately. The details of the recursion are explained in the description of Algorithm 5.6. We conjecture that this method can be used to generalize the lemma to an arbitrary number of blocks.

Proof of Theorem 5.11. We write the row-row similarity matrix S as $S = E[S] + E$, i.e., as the sum of its expectation and an error term. For each pair of rows i and j the expected similarity $E[s_{ij}]$ depends only on the blocks that i and j belong to, so the matrix $E[S]$ has block structure. From the assumptions of the model it follows that blocks have size $\Theta(m)$ and that the similarities between rows in the same block are $\Theta(n)$ larger than similarities between rows in different blocks. Thus, the quantity $d(s_2 - s_1)$ of Lemma 5.16 is $\Theta(mn)$.

To complete the proof, we next show that $\|L_E\| = o(mn)$. Recall that each entry of the similarity matrix is the sum of n independent Bernoulli trials. By a standard application of the Chernoff bound we obtain that with high probability the deviation from the expected similarity $E[s_{ij}]$ is $O(\sqrt{n})$, that is, the entries of E are bounded by $O(\sqrt{n})$. Consider now the Laplacian L_E, whose diagonal entries are clearly bounded by $O(m\sqrt{n})$. The eigenvalues of E are at most $O(m)$, by a standard result of Boppana [Bop87] on the eigenvalues of random matrices. The eigenvalues of L_E can in turn be bounded by the eigenvalues of E plus the magnitude of the diagonal entries of L_E. Thus, $|L_E| = O(m\sqrt{n} + m) = o(mn)$. Lemma 5.16 now implies that the Fiedler vector respects the structure of S, and therefore the spectral algorithm discovers the correct order of rows. The case of columns is identical. □

Algorithm 5.6 implements a practical version of the spectral sort. As mentioned after the proof of Lemma 5.15, the simplest possible spectral sorting method has problems when the Fiedler subspace is of high dimension. In particular, in many practical cases the vector has two clear components such that the rows belonging to each component are grouped together, but the values within the components are almost equal, and the within-component ordering is highly affected by numerical artefacts. Therefore Algorithm 5.6 checks whether the Fiedler eigenvector can be split into two parts whose variances are much smaller than that of the whole vector. If so, it splits the data according to the best such cutoff f_c, and calls itself recursively to sort the two pieces. In the base case where no such split exists it uses the vector to sort the matrix.

5.3 Empirical results

The algorithms presented this far only find single tiles, not complete hierarchical models. On top of these algorithms, one can use greedy searching to find several tiles. The algorithms need to be modified to disallow partially overlapping tiles, and to take into account not only

SPECTRAL-SORT(D, S)
Input: Data matrix $D_{n\times m}$,
symmetric row-similarity matrix $S_{n\times n}$.
Output: Reordered data matrix $D'_{n\times m}$.

$L_S \leftarrow \text{diag}(\sum_{\text{rows}} S) - S$
$\vec{f} \leftarrow$ second smallest eigenvector of L_S
c $\leftarrow \arg\min_c(\text{Var}(\sum_i [f_i \le f_c]\, f_i) + \text{Var}(\sum_i [f_i > f_c]\, f_i))$
if $\text{Var}(\sum_i [f_i \le f_c]\, f_i) + \text{Var}(\sum_i [f_i > f_c]\, f_i) > \frac{1}{2}\text{Var}(\vec{f})$:
    $D' \leftarrow D$ with rows sorted using $\vec{f}$
else:
    $D_1 \leftarrow$ rows i of D with $f_i \le f_c$
    $D_2 \leftarrow$ rows i of D with $f_i > f_c$
    $S_1, S_2 \leftarrow$ corresponding submatrices of S
    $D'_1 \leftarrow$ SPECTRAL-SORT(D_1, S_1)
    $D'_2 \leftarrow$ SPECTRAL-SORT(D_2, S_2)
    $D' \leftarrow$ concatenation of D'_1 and D'_2
return D'

Algorithm 5.6. Rearrange the rows of a matrix. The similarity matrix S may be obtained using e.g. dot-product or Hamming measures.

the log-likelihood due to one tile but the change in log-likelihood of the surrounding tile. Since these modifications merely add bookkeeping but do not change anything fundamental in the algorithms, the details are not presented here.

In addition to rejecting partial overlaps between tiles, it may be useful to direct the greedy search by rejecting tiles that are inside previously found tiles[1] (bottom-up strategy), or rejecting tiles that contain previously found tiles (top-down strategy), or rejecting both (single-level strategy). If neither kind of tiles are rejected, we call the search strategy "mixed".

The spectral sorting (Algorithm 5.6) could only be completed for the two smallest data matrices, *Course* and *Chess*. The similarity measures used were dot product (i.e., shared number of ones) and Hamming (i.e., number of positions where the values are equal). For these four reordered matrices, local search for geometric tiles (Algorithm 5.3) was run with the four search strategies outlined previously. At each stage of the greedy search, the algorithm was run 100 times with random restarts, and the best tile was used. This was repeated until 50 tiles were found. Results

[1] Excluding, of course, the tile that covers the complete data matrix.

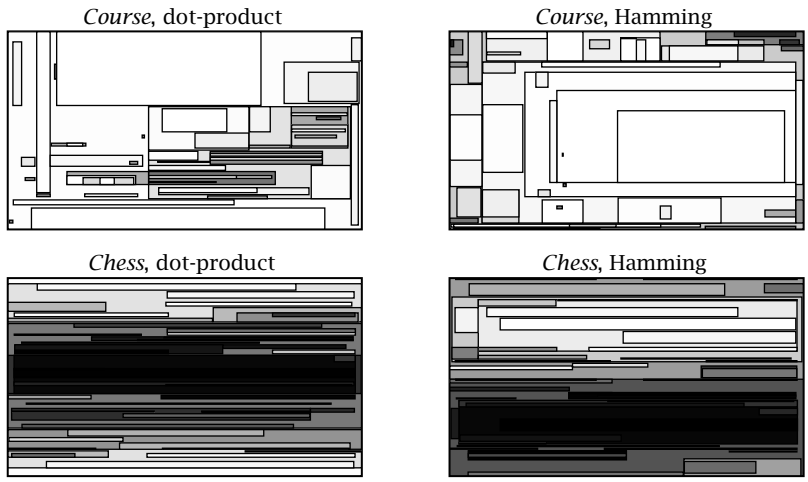


Figure 5.2. Tiles found using mixed strategy in the two data sets that could be sorted. The grey level of a tile indicates the parameter p, i.e., darker tiles include more ones.

for the mixed strategy for both orderings of both data sets are shown in Figure 5.2. Figure 5.3 shows the results of all four strategies in *Course* reordered using dot product similarity. The results for the top-down and mixed strategies are similar, and likewise for bottom-up and single-level. The results are similar for *Chess* and Hamming-ordered *Course*. It seems that the bottom-up and single-level strategies are more likely to "paint themselves into a corner" by selecting large tiles early and not being able to find details within the tiles. The mixed strategy has the greatest freedom to find more tiles.

5.4 Discussion

Tile models. The tile models of this chapter can depict features of the data in a considerably versatile manner compared to the dense itemsets of the previous chapter. There are important differences: tiles can also reflect lack of co-occurrence, and they are not restricted in the number of attributes they can include, like dense itemsets are because of the levelwise mining algorithm. However, tiles come at the price of more time-consuming algorithms. Especially the spectral sorting can take

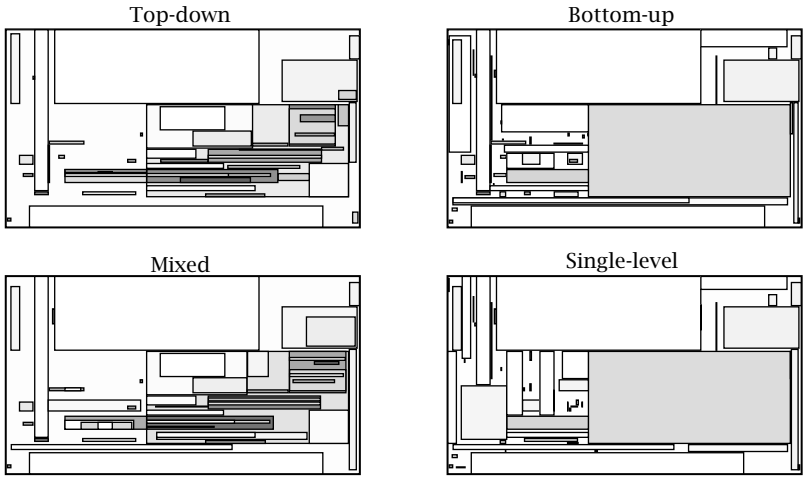


Figure 5.3. Tiles found using greedy search with all four strategies in *Course* reordered using dot product similarity. The grey level of a tile indicates the parameter p, i.e., darker tiles include more ones.

much time, and needs to be improved for the models to be practically discoverable. One possibility is to perform the sorting on only a sample of the data and then extend the order, as in [BPM01].

A generalization. Consider a supermarket scenario where almost every customer buys milk but very few buy mineral water, but some customers are different in that they buy less milk and more mineral water. Such customers do not form ordinary tiles, but they can be characterized by a slightly modified tile model that takes into account the background probabilities of each attribute (i.e., column). Each tile simply needs a vector of numbers $(p_j)_{j \in Y}$ instead of a single number p. We can use the same likelihood function as before. The maximum-likelihood value for $\vec{p}$ is obtained by averaging in each column. This formulation is close to modeling by mixtures of multivariate Bernoulli distributions. There are two crucial differences: first, each tile determines a multivariate Bernoulli distribution on a different set of dimensions; second, the data points (matrix rows) can be explained by any number of tiles, and in the case of overlapping tiles, the smaller tile (which is necessarily contained by the other) takes precedence. We also assume that there is top-level tile that explains all data not explained by smaller tiles.

In this chapter, we have concentrated on finding maximum-likelihood tiles, and have not tried to guard against overfitting. With column-specific parameters, overfitting is a much larger problem. For almost any data matrix and any set of rows, we can maximize the likelihood by including almost every column in the tile: the fraction of 1s within the rows is unlikely to be exactly the same as the overall fraction, and the likelihood is improved by any small difference. Thus some kind of a penalization scheme must be used.

For any penalization term that can be written in the form

$$p(\tau) = f(X) + \sum_{j \in Y} g(X, j) + h(|Y|),$$

where $f(X)$ is constant with respect to Y, $g(X, j)$ depends only on the column j and not the other columns, and $h(|Y|)$ depends only on the number of columns, we can for any row-set X find the optimal column-set Y easily: compute $g(X, j)$ for every $j = 1, \ldots, m$, sort the columns in order of increasing g, and consider each of the sets

$$Y_\delta = \{\, j \mid g(X, j) \leq \delta \,\}.$$

The number of these sets is $\Theta(m)$, so a linear search is feasible. If $h(|Y|)$ is increasing, it suffices to examine $\Theta(\log m)$ sets in a binary search, but this is not a large improvement, since the sorting already takes $\Theta(m \log m)$ time.

This observation suggests an iterative algorithm for finding column-parameterized tiles: starting from a random set X_0 of rows, find the optimal column set Y_0, then somehow find a good row set X_1, and iterate until convergence. As we have seen, finding the columns given the rows is easy; the opposite direction is not so simple. One possibility is to use an EM-like algorithm: given the set Y of columns (dimensions), consider only these dimensions and find the best division of all rows (data points) into two clusters, then let X be one of the row sets thus defined.

Related work. There are of course numerous different approaches for modeling and clustering discrete data, such as topic models [CC99, BMS02, SBM03] and latent-variable models [Gir01]. Closer to our method is that of [CPMF04], where the data is also rearranged and divided into clusters. However, in this approach the clusters are formed by dividing both rows and columns into k clusters, and the final clusters are restricted to consist of the k^2 intersections of the row- and column-wise clusters. The tiles of [GGM04] and the holes and empty spaces of [LKH97, EGLM03] differ from ours by being full of ones or zeros, like frequent itemsets.

An interesting new approach is that of α/β concepts, which allow a fixed number of exceptions on each row and column [BRB04]. They are thus quite close to fault-tolerant itemsets [PTH01].

Some inspiration for the hierarchical models came from the work on learning decision lists [Riv87] and ripple-down rules [GC95]. A PAC-learning algorithm for finding hierarchical concepts was given in [KMU92]. Spectral algorithms are widely used for problems such as linear systems [PSW92], ordering [ABH99, KH02], and data clustering [NJW01, ZHD+01, Dhi01].

Chapter 6

Conclusion

We have explored two data mining questions: how to approximate Boolean queries using frequent itemsets, and how to extend the frequent itemset concept to allow for patterns that do not fit the mould of perfect co-occurrence. To answer the first question, we have concentrated on the truncated inclusion-exclusion algorithm, for which we proved a large but tight upper bound in terms of the negative border, and which we generalized to arbitrary Boolean formulas from disjunctions. Experiments show that the large upper bound is pessimistic, although with some dense data sets the algorithm does not give useful results. The probabilistic approach of maximum entropy and its special case the independence assumption are usually more accurate, but cannot be applied in all the cases where the truncated inclusion-exclusion algorithm can.

For the second question we have two answers: dense itemsets, which are a true generalization of frequent itemsets, and tile models, which are a different approach. Dense itemsets can be found using the same algorithm as frequent itemsets, but they also share the limitations of the algorithm; namely, to discover a set of k items, all its $2^k - 1$ subsets must be discovered first. Our algorithm for finding combinatorial tile models is not thus limited, but it is usable in its current form for fairly small data sets only. Experiments on both concepts show that within their limitations, the algorithms are usable and can find interesting patterns in the data.

Much remains to be done. Some detailed ideas for further work are given in the preceding chapters, but one general theme that has not been touched on by this work is that of predictive modeling. A descriptive model may allow for exploration of the data and may reveal interesting things, but in many applications the interest is in predicting future data. The question of overfitting arises immediately: a descriptive model that

is too closely tied to the data it describes will not generalize well to other data. The usual solutions of cross-validation and penalization of complexity can be applied to combat overfitting.

An underlying theme connecting the topics discussed in this dissertation is the interplay of two data mining objectives, local patterns and descriptive models. A pattern is something like a frequent itemset: an interesting phenomenon occurring in some part of the data. In contrast, a descriptive model tells us something interesting about the whole data. The query approximation problem of Chapter 3 is motivated by a desire to convert a frequent itemset collection into a model that can be used to answer queries about the data. The dense itemsets of Chapter 4 were used to create a description of the data using a greedy algorithm inspired by the set cover problem. While frequent itemsets could be used similarly, the requirement of complete co-occurrence hinders the effort, and the relaxed requirement of dense itemsets admits interesting sets that frequent itemset mining would not have found, as was shown by the experiments on the advanced course data. The tile models of Chapter 5 are, of course, descriptive models built from local patterns. The greedy algorithm adds detail to the model by always seeking out the most interesting pattern yet unexplained by the model.

Bibliography

[ABH99] Jonathan Atkins, Erik Boman, and Bruce Hendrickson. A spectral algorithm for seriation and the consecutive ones problem. *SIAM Journal on Computing*, 28(1), 1999.

[AGM04] Foto Afrati, Aristides Gionis, and Heikki Mannila. Approximating a collection of frequent sets. In Ronny Kohavi, Johannes Gehrke, William DuMouchel, and Joydeep Ghosh, editors, *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004)*, pages 12–19. ACM Press, 2004.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *SIGMOD Conference*, pages 207–216. ACM Press, 1993.

[AMS+96] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 12, pages 307–328. AAAI Press, 1996.

[BBR00] Jean-François Boulicaut, Artur Bykowski, and Christophe Rigotti. Approximation of frequency queris by means of free-sets. In Djamel A. Zighed, Henryk Jan Komorowski, and Jan M. Zytkow, editors, *PKDD*, volume 1910 of *Lecture Notes in Computer Science*, pages 75–85. Springer, 2000.

[BGKM02] Endre Boros, Vladimir Gurvich, Leonid Khachiyan, and Kazuhisa Makino. On the complexity of generating maximal frequent and minimal infrequent sets. In Helmut Alt and Afonso Ferreira, editors, *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2285 of

Lecture Notes in Computer Science, pages 133–141. Springer, 2002.

[BGZ04] Roberto Bayardo, Bart Goethals, and Mohammed J. Zaki, editors. *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI–04)*, volume 126 of *CEUR-WS*, Brighton, UK, November 2004. `http://CEUR-WS.org/Vol-126/`.

[BMS02] Ella Bingham, Heikki Mannila, and Jouni K. Seppänen. Topics in 0–1 data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pages 450–455. ACM, 2002.

[Bol88] Béla Bollobás. *Combinatorics: set systems, hypergraphs, families of vectors and combinatorial probability*. Cambridge University Press, 1988.

[Bop87] Ravi B. Boppana. Eigenvalues and graph bisection: An average-case analysis. In *28th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 280–285. IEEE Computer Society, October 1987.

[Bou04] Jean-François Boulicaut. Inductive databases and multiple uses of frequent itemsets: the cInQ approach. In Rosa Meo, Pier Luca Lanzi, and Mika Klemettinen, editors, *Database Support for Data Mining Applications: Discovering Knowledge with Inductive Queries*, volume 2682 of *Lecture Notes in Artificial Intelligence*, pages 1–23. Springer-Verlag, 2004.

[BPM01] Alina Beygelzimer, Chang-Shing Perng, and Sheng Ma. Fast ordering of large categorical datasets for better visualization. In Foster Provost, Ramakrishnan Srikant, Mario Schkolnick, and Doheon Lee, editors, *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2001)*, pages 239–244, New York, NY, USA, 2001. ACM Press.

[BRB04] Jérémy Besson, Céline Robardet, and Jean-François Boulicaut. Mining formal concepts with a bounded number of exceptions from transactional data. In Bart Goethals and Arno Siebes, editors, *Knowledge Discovery in Inductive Databases, Third International Workshop (KDID 2004)*, volume 3377 of *Lecture Notes in Computer Science*, pages 33–45. Springer-Verlag, 2004.

[BSH02] Artur Bykowski, Jouni K. Seppänen, and Jaakko Hollmén. Model-independent bounding of Boolean formulae in binary

data. In Mika Klemettinen, Rosa Meo, Fosca Giannotti, and Luc De Raedt, editors, *Knowledge Discovery in Inductive Databases, First International Workshop (KDID 2002)*, pages 20–31, Helsinki, Finland, August 2002. University of Helsinki Department of Computer Science Report B–2002–7.

[BSH04] Artur Bykowski, Jouni K. Seppänen, and Jaakko Hollmén. Model-independent bounding of the supports of Boolean formulae in binary data. In Rosa Meo, Pier Luca Lanzi, and Mika Klemettinen, editors, *Database Support for Data Mining Applications: Discovering Knowledge with Inductive Queries*, volume 2682 of *Lecture Notes in Artificial Intelligence*, pages 234–249. Springer-Verlag, 2004.

[Cal04] Toon Calders. Deducing bounds on the support of frequent itemsets. In *Database Support for Data Mining Applications: Discovering Knowledge with Inductive Queries*, pages 214–233. Springer-Verlag, 2004.

[CC99] Chris Clifton and Robert Cooley. TopCat: Data mining for topic identification in a text corpus. In Jan M. Zytkow and Jan Rauch, editors, *Principles of Data Mining and Knowledge Discovery, Third European Conference (PKDD 1999)*, volume 1704 of *Lecture Notes in Computer Science*, pages 174–183. Springer, 1999.

[CG02] Toon Calders and Bart Goethals. Mining all non-derivable frequent itemsets. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Principles of Data Mining and Knowledge Discovery, 6th European Conference (PKDD 2002)*, volume 2431 of *Lecture Notes in Computer Science*, pages 74–85. Springer, 2002.

[CG03] Toon Calders and Bart Goethals. Minimal k-free representations of frequent sets. In Nada Lavrač, Dragan Gamberger, Ljupčo Todorovski, and Hendrik Blockeel, editors, *Knowledge Discovery in Databases: PKDD 2003: 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, volume 2838 of *Lecture Notes in Artificial Intelligence*, pages 71–82, Cavtat–Dubrovnik, Croatia, September 2003. Springer-Verlag.

[CPMF04] Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra S. Modha, and Christos Faloutsos. Fully automatic cross-associations. In Ronny Kohavi, Johannes Gehrke, William DuMouchel, and Joydeep Ghosh, editors, *Proceedings of the*

Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004), pages 79–88. ACM Press, 2004.

[DH73] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[Dhi01] Inderjit S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Knowledge Discovery and Data Mining*, pages 269–274, 2001.

[DS40] W. Edwards Deming and Frederick F. Stephan. On a least squares adjustment of a sampled frequency table when the expected marginal totals are known. *The Annals of Mathematical Statistics*, 11(4):427–444, December 1940.

[EGLM03] Jeff Edmonds, Jarek Gryz, Dongming Liang, and Renée J. Miller. Mining for empty spaces in large data sets. *Theor. Comput. Sci.*, 3(296):435–452, 2003.

[Fie73] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[GC95] Brian R. Gaines and Paul Compton. Induction of ripple-down rules applied to modeling large databases. *Journal of Intelligent Information Systems*, 5(3):211–228, 1995.

[GGM04] Floris Geerts, Bart Goethals, and Taneli Mielikäinen. Tiling databases. In Einoshin Suzuki and Setsuo Arikawa, editors, *Discovery Science*, volume 3245 of *Lecture Notes in Computer Science*, pages 278–289. Springer, 2004.

[Gir01] Mark Girolami. The topographic organization and visualization of binary data using multivariate-Bernoulli latent variable models. *IEEE Transactions on Neural Networks*, 12(6):1367–1374, 2001.

[GKM+03] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen, and Ram Sewak Sharma. Discovering all most specific sentences. *ACM Trans. Database Syst.*, 28(2):140–174, 2003.

[GMS04] Aristides Gionis, Heikki Mannila, and Jouni K. Seppänen. Geometric and combinatorial tiles in 0-1 data. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Knowledge Discovery in Databases: PKDD 2004: 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, volume 3202 of *Lecture*

Notes in Artificial Intelligence, pages 173–184. Springer-Verlag, 2004.

[GP02] Linchun Gao and András Prékopa. Lower and upper bounds for the probability that at least r and exactly r out of n events occur. *Mathematical Inequalities & Applications*, 5(2):315–333, 2002.

[GS96] Janos Galambos and Italo Simonelli. *Bonferroni-type Inequalities with Applications*. Probability and its Applications. Springer-Verlag, 1996.

[GZ03] Bart Goethals and Mohammed J. Zaki, editors. *Proceedings of the Workshop on Frequent Itemset Mining Implementations (FIMI–03)*, volume 90 of *CEUR-WS*, Melbourne, Florida, 2003. `http://CEUR-WS.org/Vol-90/`.

[Hai65] Theodore Hailperin. Best possible inequalities for the probability of a logical function of events. *The American Mathematical Monthly*, 72(4):343–359, April 1965.

[HBM98] S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[HMS01] David Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Massachusetts, 2001.

[HSM03] Jaakko Hollmén, Jouni K. Seppänen, and Heikki Mannila. Mixture models and frequent sets: Combining global and local methods for 0–1 data. In Daniel Barbará and Chandrika Kamath, editors, *Proceedings of the Third SIAM International Conference on Data Mining, San Francisco, CA, USA, May 1-3, 2003*. SIAM, 2003.

[HWLT02] Jiawei Han, Jianyong Wang, Ying Lu, and Petre Tzvetkov. Mining top-K frequent closed patterns without minimum support. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)*, pages 211–218. IEEE Computer Society, 2002.

[IK68] C. T. Ireland and S. Kullback. Contingency tables with given marginals. *Biometrika*, 55(1):179–188, March 1968.

[IM96] Tomasz Imielinski and Heikki Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, November 1996.

[JSR04] Szymon Jaroszewicz, Dan A. Simovici, and Ivo Rosenberg. Measures on Boolean polynomials and their applications in data mining. *Discrete Applied Mathematics*, 144(1-2):123–139, 2004.

[KH02] Y. Koren and D. Harel. Multi-scale algorithm for the linear arrangement problem. Technical Report MCS02-04, The Weizmann Institute of Science, 2002.

[KK01] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In Nick Cercone, Tsau Young Lin, and Xindong Wu, editors, *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 313–320. IEEE Computer Society, 2001.

[KM76] Stratis Kounias and Jacqueline Marin. Best linear Bonferroni bounds. *SIAM Journal on Applied Mathematics*, 30(2):307–323, 1976.

[KMN99] Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, April 1999.

[KMU92] Jyrki Kivinen, Heikki Mannila, and Esko Ukkonen. Learning hierarchical rule sets. In *Computational Learning Theory*, pages 37–44, 1992.

[Knu92] Donald E. Knuth. Two notes on notation. *The American Mathematical Monthly*, 99(5):403–422, May 1992.

[KPR04] Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan. Segmentation problems. *J. ACM*, 51(2):263–280, 2004.

[KS02] David Kessler and Jeremy Schiff. Inclusion-exclusion *redux*. *Electronic Communications in Probability*, 7:85–96, 2002.

[Kwe75] Seymour M. Kwerel. Most stringent bounds on aggregated probabilities of partially specified dependent probability systems. *Journal of the American Statistical Association*, 70(350):472–479, 1975.

[LKH97] Bing Liu, Liang-Ping Ku, and Wynne Hsu. Discovering interesting holes in data. In *Proceedings of Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 930–935, 1997.

[LN90] Nathan Linial and Noam Nisan. Approximate inclusion-exclusion. *Combinatorica*, 10(4):349–365, 1990.

[LS00] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In Todd K. Leen, Thomas G.

Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 556–562. MIT Press, 2000.

[Man02] Heikki Mannila. Local and global methods in data mining: Basic techniques and open problems. In P. Widmayer, F. Triguero, R. Morales, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *Automata, Languages and Programming*, volume 2380 of *LNCS*, pages 57–68. Springer-Verlag, 2002.

[MM03] Taneli Mielikäinen and Heikki Mannila. The pattern ordering problem. In Nada Lavrač, Dragan Gamberger, Ljupčo Todorovski, and Hendrik Blockeel, editors, *Knowledge Discovery in Databases: PKDD 2003: 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, volume 2838 of *Lecture Notes in Artificial Intelligence*, pages 327–338, Cavtat–Dubrovnik, Croatia, September 2003. Springer-Verlag.

[MT96] Heikki Mannila and Hannu Toivonen. Multiple uses of frequent sets and condensed representations (extended abstract). In *2nd International Conference on Knowledge Discovery and Data Mining (KDD–1996)*, pages 189–194, Portland, Oregon, August 1996. AAAI Press.

[MTV97] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, November 1997.

[NJW01] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 849–856. MIT Press, 2001.

[PBTL99] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In Catriel Beeri and Peter Buneman, editors, *Database Theory - ICDT '99, 7th International Conference*, volume 1540 of *Lecture Notes in Computer Science*, pages 398–416. Springer, 1999.

[PG05] András Prékopa and Linchun Gao. Bounding the probability of the union of events by aggregation and disaggregation in linear programs. *Discrete Applied Mathematics*, 145:444–454, 2005.

[PMS00] Dmitry Pavlov, Heikki Mannila, and Padhraic Smyth. Probabilistic models for query approximation with large sparse binary

data sets. In Craig Boutilier and Moisés Goldszmidt, editors, *UAI '00: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence*, pages 465–472. Morgan Kaufmann, 2000.

[Pré88] András Prékopa. Boole-Bonferroni inequalities and linear programming. *Operations Research*, 36(1):145–162, 1988.

[Pré90] András Prékopa. Sharp bounds on probabilities using linear programming. *Operations Research*, 38(2):227–239, 1990.

[PS01] Dmitry Pavlov and Padhraic Smyth. Probabilistic query models for transaction data. In Foster Provost, Ramakrishnan Srikant, Mario Schkolnick, and Doheon Lee, editors, *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2001)*, New York, NY, USA, 2001. ACM Press.

[PSW92] A. Pothen, H. Simon, and L. Wang. Spectral nested dissection. Technical Report CS-92-01, Pennsylvania State University, Department of Computer Science, 1992.

[PTH01] Jian Pei, Anthony K.H. Tung, and Jiawei Han. Fault-tolerant frequent pattern mining: Problems and challenges. In *Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001.

[Riv87] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[SBM03] Jouni K. Seppänen, Ella Bingham, and Heikki Mannila. A simple algorithm for topic identification in 0–1 data. In Nada Lavrač, Dragan Gamberger, Ljupčo Todorovski, and Hendrik Blockeel, editors, *Knowledge Discovery in Databases: PKDD 2003: 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, volume 2838 of *Lecture Notes in Artificial Intelligence*, pages 423–434, Cavtat–Dubrovnik, Croatia, September 2003. Springer-Verlag.

[SM04] Jouni K. Seppänen and Heikki Mannila. Dense itemsets. In Ronny Kohavi, Johannes Gehrke, William DuMouchel, and Joydeep Ghosh, editors, *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004)*, pages 683–688. ACM Press, 2004.

[SM05] Jouni K. Seppänen and Heikki Mannila. Boolean formulas and frequent sets. In Jean-François Boulicaut, Luc de Raedt, and Heikki Mannila, editors, *Constraint-based mining and inductive*

databases, Lecture Notes in Computer Science, pages 348–361. Springer-Verlag, 2005.

[Swe88] J. A. Swets. Measuring the accuracy of diagnostic systems. *Science*, 240(4857):1285–93, June 1988.

[Tat06] Nikolaj Tatti. Safe projections of binary data sets. *Acta Informatica*, 42(8-9):617–638, 2006.

[TM80] John Tydeman and Robert Mitchell. A note on the Kounias and Marin method of best linear Bonferroni bounds. *SIAM Journal on Applied Mathematics*, 39(1):173–177, 1980.

[TYH03] Petre Tzvetkov, Xifeng Yan, and Jiawei Han. TSP: Mining top-K closed sequential patterns. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003)*, pages 347–354. IEEE Computer Society, 2003.

[YFB01] Cheng Yang, Usama Fayyad, and Paul S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In Foster Provost, Ramakrishnan Srikant, Mario Schkolnick, and Doheon Lee, editors, *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2001)*, New York, NY, USA, 2001. ACM Press.

[ZHD+01] Hongyuan Zha, Xiaofeng He, Chris H. Q. Ding, Ming Gu, and Horst D. Simon. Bipartite graph partitioning and data clustering. In *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management*, pages 25–32. ACM, 2001.

[ZKM01] Zijian Zheng, Ron Kohavi, and Llew Mason. Real world performance of association rule algorithms. In Foster Provost, Ramakrishnan Srikant, Mario Schkolnick, and Doheon Lee, editors, *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2001)*, pages 401–406, New York, NY, USA, 2001. ACM Press.

HELSINKI UNIVERSITY OF TECHNOLOGY
DISSERTATIONS IN COMPUTER AND INFORMATION SCIENCE

Report D1 Koskela, M., Interactive Image Retrieval Using Self-Organizing Maps, 2003.

Report D2 Sinkkonen, J., Learning Metrics and Discriminative Clustering, 2003.

Report D3 Hurri, J., Computational Models Relating Properties of Visual Neurons to Natural Stimulus Statistics, 2003.

Report D4 Bingham, E., Advances in Independent Component Analysis with Applications to Data Mining, 2003.

Report D5 Himberg, J., From Insights to Innovations: Data Mining, Visualization and User Interfaces, 2004.

Report D6 Särelä, J., Exploratory Source Separation in Biomedical Systems, 2004.

Report D7 Peltonen, J., Data Exploration with Learning Metrics, 2004.

Report D8 Könönen, V., Multiagent Reinforcement Learning in Markov Games: Asymmetric and Symmetric Approaches, 2004.

Report D9 Inki, M., Extensions of Independent Component Analysis for Natural Image Data, 2004.

Report D10 Honkela, A., Advances in Variational Bayesian Nonlinear Blind Source Separation, 2005.

Report D11 Nikkilä, J., Exploratory Cluster Analysis of Genomic High-Throughput Data Sets and Their Dependencies, 2005.

ISBN 951-22-8201-1
ISSN 1459-7020