Sašo Džeroski
Bart Goethals
Panče Panov *Editors*

Inductive Databases and Constraint-Based Data Mining

Springer

Inductive Databases and Constraint-Based Data Mining

Sašo Džeroski • Bart Goethals • Panče Panov
Editors

Inductive Databases and Constraint-Based Data Mining

Springer

Editors
Sašo Džeroski
Jožef Stefan Institute
Dept. of Knowledge Technologies
Jamova cesta 39
SI-1000 Ljubljana
Slovenia
Saso.Dzeroski@ijs.si

Panče Panov
Jožef Stefan Institute
Dept. of Knowledge Technologies
Jamova cesta 39
SI-1000 Ljubljana
Slovenia
Pance.Panov@ijs.si

Bart Goethals
University of Antwerp
Mathematics and Computer Science Dept.
Middelheimlaan 1
B-2020 Antwerpen
Belgium
Bart.Goethals@ua.ac.be

ISBN 978-1-4419-7737-3 e-ISBN 978-1-4419-7738-0
DOI 10.1007/978-1-4419-7738-0
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2010938297

Preface

This book is about inductive databases and constraint-based data mining, emerging research topics lying at the intersection of data mining and database research. The aim of the book as to provide an overview of the state-of- the art in this novel and exciting research area. Of special interest are the recent methods for constraint-based mining of global models for prediction and clustering, the unification of pattern mining approaches through constraint programming, the clarification of the relationship between mining local patterns and global models, and the proposed integrative frameworks and approaches for inducive databases. On the application side, applications to practically relevant problems from bioinformatics are presented.

Inductive databases (IDBs) represent a database view on data mining and knowledge discovery. IDBs contain not only data, but also generalizations (patterns and models) valid in the data. In an IDB, ordinary queries can be used to access and manipulate data, while inductive queries can be used to generate (mine), manipulate, and apply patterns and models. In the IDB framework, patterns and models become ”first-class citizens” and KDD becomes an extended querying process in which both the data and the patterns/models that hold in the data are queried.

The IDB framework is appealing as a general framework for data mining, because it employs declarative queries instead of ad-hoc procedural constructs. As declarative queries are often formulated using constraints, inductive querying is closely related to constraint-based data mining. The IDB framework is also appealing for data mining applications, as it supports the entire KDD process, i.e., nontrivial multi-step KDD scenarios, rather than just individual data mining operations.

The interconnected ideas of inductive databases and constraint-based mining have the potential to radically change the theory and practice of data mining and knowledge discovery. The book provides a broad and unifying perspective on the field of data mining in general and inductive databases in particular. The 18 chapters in this state-of-the-art survey volume were selected to present a broad overview of the latest results in the field.

Unique content presented in the book includes constraint-based mining of global models for prediction and clustering, including predictive models for structured out-

puts and methods for bi-clustering; integration of mining local (frequent) patterns and global models (for prediction and clustering); constraint-based mining through constraint programming; integrative IDB approaches at the system and framework level; and applications to relevant problems that attract strong interest in the bioinformatics area. We hope that the volume will increase in relevance with time, as we witness the increasing trends to store patterns and models (produced by humans or learned from data) in addition to data, as well as retrieve, manipulate, and combine them with data.

This book contains sixteen chapters presenting recent research on the topics of inductive databases and queries, as well as constraint-based data, conducted within the project IQ (Inductive Queries for mining patterns and models), funded by the EU under contract number IST-2004-516169. It also contains two chapters on related topics by researchers coming from outside the project (Siebes and Puspitaningrum; Wicker et al.)

This book is divided into four parts. The first part describes the foundations of and frameworks for inductive databases and constraint-based data mining. The second part presents a variety of techniques for constraint-based data mining or inductive querying. The third part presents integration approaches to inductive databases. Finally, the fourth part is devoted to applications of inductive querying and constraint-based mining techniques in the area of bioinformatics.

The first, introductory, part of the book contains four chapters. Džeroski first introduces the topics of inductive databases and constraint-based data mining and gives a brief overview of the area, with a focus on the recent developments within the IQ project. Panov et al. then present a deep ontology of data mining. Blockeel et al. next present a practical comparative study of existing data-mining/inductive query languages. Finally, De Raedt et al. are concerned with mining under composite constraints, i.e., answering inductive queries that are Boolean combinations of primitive constraints.

The second part contains six chapters presenting constraint-based mining techniques. Besson et al. present a unified view on itemset mining under constraints within the context of constraint programming. Bringmann et al. then present a number of techniques for integrating the mining of (frequent) patterns and classification models. Struyf and Džeroski next discuss constrained induction of predictive clustering trees. Bingham then gives an overview of techniques for finding segmentations of sequences, some of these being able to handle constraints. Cerf et al. discuss constrained mining of cross-graph cliques in dynamic networks. Finally, De Raedt et al. introduce ProbLog, a probabilistic relational formalism, and discuss inductive querying in this formalism.

The third part contains four chapters discussing integration approaches to inductive databases. In the Mining Views approach (Blockeel et al.), the user can query the collection of all possible patterns as if they were stored in traditional relational tables. Wicker et al. present SINDBAD, a prototype of an inductive database system that aims to support the complete knowledge discovery process. Siebes and Puspitaningrum discuss the integration of inductive and ordinary queries (relational algebra). Finally, Vanschoren and Blockeel present experiment databases.

The fourth part of the book, contains four chapters dealing with applications in the area of bioinformatics (and chemoinformatics). Vens et al. describe the use of predictive clustering trees for predicting gene function. Slavkov and Džeroski describe several applications of predictive clustering trees for the analysis of gene expression data. Rigotti et al. describe how to use mining of frequent patterns on strings to discover putative transcription factor binding sites in gene promoter sequences. Finally, King et al. discuss a very ambitious application scenario for inductive querying in the context of a robot scientist for drug design.

The content of the book is described in more detail in the last two sections of the introductory chapter by Džeroski.

We would like to conclude with a word of thanks to those that helped bring this volume to life: This includes (but is not limited to) the contributing authors, the referees who reviewed the contributions, the members of the IQ project and the various funding agencies. A more complete listing of acknowledgements is given in the Acknowledgements section of the book.

September 2010

Sašo Džeroski
Bart Goethals
Panče Panov

Acknowledgements

Heartfelt thanks to all the people and institutions that made this volume possible and helped bring it to life.

First and foremost, we would like to thank the contributing authors. They did a great job, some of them at short notice. Also, most of them showed extraordinary patience with the editors.

We would then like to thank the reviewers of the contributed chapters, whose names are listed in a separate section. Each chapter was reviewed by at least two (on average three) referees. The comments they provided greatly helped in improving the quality of the contributions.

Most of the research presented in this volume was conducted within the project IQ (Inductive Queries for mining patterns and models). We would like to thank everybody that contributed to the success of the project: This includes the members of the project, both the contributing authors and the broader research teams at each of the six participating institutions, the project reviewers and the EU officials handling the project. The IQ project was funded by the European Comission of the EU within FP6-IST, FET branch, under contract number FP6-IST-2004-516169.

In addition, we want to acknowledge the following funding agencies:

- Sašo Džeroski is currently supported by the Slovenian Research Agency (through the research program *Knowledge Technologies* under grant P2-0103 and the research projects *Advanced machine learning methods for automated modelling of dynamic systems* under grant J2-0734 and *Data Mining for Integrative Data Analysis in Systems Biology* under grant J2-2285) and the European Commission (through the FP7 project PHAGOSYS *Systems biology of phagosome formation and maturation - modulation by intracellular pathogens* under grant number HEALTH-F4-2008-223451). He is also supported by the Centre of Excellence for Integrated Approaches in Chemistry and Biology of Proteins (operation no. OP13.1.1.2.02.0005 financed by the European Regional Development Fund (85%) and the Slovenian Ministry of Higher Education, Science and Technology (15%)), as well as the Jozef Stefan International Postgraduate School in Ljubljana.

- Bart Goethals wishes to acknowledge the support of FWO-Flanders through the project ”Foundations for inductive databases”.
- Panče Panov is supported by the Slovenian Research Agency through the research projects *Advanced machine learning methods for automated modelling of dynamic systems* (under grant J2-0734) and *Data Mining for Integrative Data Analysis in Systems Biology* (under grant J2-2285).

Finally, many thanks to our Springer editors, Jennifer Maurer and Melissa Fearon, for all the support and encouragement.

September 2010

Sašo Džeroski
Bart Goethals
Panče Panov

List of Reviewers

Hendrik Blockeel	Katholieke Universiteit Leuven, Belgium
Marko Bohanec	Jožef Stefan Institute, Slovenia
Jean-Francois Boulicaut	University of Lyon, INSA Lyon, France
Mario Boley	University of Bonn and Fraunhofer IAIS, Germany
Toon Calders	Eindhoven Technical University, Netherlands
Vineet Chaoji	Yahoo! Labs, Bangalore, India
Amanda Clare	Aberystwyth University, United Kingdom
James Cussens	University of York, United Kingdom
Tomaž Curk	University of Ljubljana, Ljubljana, Slovenia
Ian Davidson	University of California - Davis, USA
Luc Dehaspe	Katholieke Universiteit Leuven, Belgium
Luc De Raedt	Katholieke Universiteit Leuven, Belgium
Jeroen De Knijf	University of Antwerp, Belgium
Tijl De Bie	University of Bristol, United Kingdom
Sašo Džeroski	Jožef Stefan Institute, Slovenia
Elisa Fromont	University of Jean Monnet, France
Gemma C. Garriga	University of Paris VI, France
Christophe Giraud-Carrier	Brigham Young University, USA
Jiawei Han	University of Illinois at Urbana-Champaign, USA
Hannes Heikinheimo	Aalto Universit, Finland
Cristoph Hema	In Silico Toxicology, Switzerland
Andreas Karwath	Albert-Ludwigs-Universitat, Germany
Jörg-Uwe Kietz	University of Zurich, Switzerland
Arno Knobbe	University of Leiden, Netherlands
Petra Kralj Novak	Jožef Stefan Institute, Slovenia
Stefan Kramer	Technische Universität München, Germany
Rosa Meo	University of Torino, Italy
Pauli Miettinen	Max-Planck-Institut für Informatik, Germany
Siegfried Nijssen	Katholieke Universiteit Leuven, Belgium
Markus Ojala	Aalto University, Finland
Themis Palpanas	University of Trento, Italy

Panče Panov	Jožef Stefan Institute, Ljubljana, Slovenia
Juho Rousu	University of Helsinki, Finland
Nikolaj Tatti	University of Antwerp, Belgium
Grigorios Tsoumakas	Aristotle University of Thessaloniki, Greece
Giorgio Valentini	University of Milano, Italy
Jan Van den Bussche	Universiteit Hasselt, Belgium
Jilles Vreeken	University of Utrecht, Netherlands
Kiri Wagstaff	California Institute of Technology, USA
Joerg Wicker	Technische Universität München, Germany
Gerson Zaverucha	Federal University of Rio de Janeiro, Brazil
Albrecht Zimmermann	Katholieke Universiteit Leuven, Belgium
Bernard Ženko	Jožef Stefan Institute, Slovenia

Contents

Chapter 13
Patterns on Queries

Arno Siebes and Diyah Puspitaningrum

Abstract One of the most important features of any database system is that it supports *queries*. For example, in relational databases one can construct new tables from the stored tables using relational algebra. For an inductive database, it is reasonable to assume that the stored tables have been modelled. The problem we study in this chapter is: do the models available on the stored tables help to model the table constructed by a query? To focus the discussion, we concentrate on one type of modelling, i.e., computing frequent item sets. This chapter is based on results reported in two earlier papers [12, 13]. Unifying the approaches advocated by those papers as well as comparing them is the main contribution of this chapter.

13.1 Introduction

By far the most successful type of DBMS is relational. In a relational database, the data is stored in tables and a query constructs a new table from these stored tables using, e.g., relational algebra [5]. While querying an inductive relational database, the user will, in general, not only be interested in the table that the query yields, but also -if not more- in particular models induced from that result-table. Since inductive databases have models as first-class citizens -meaning they can be stored and queried- it is reasonable to assume that the original, stored, tables are already modelled. Hence, a natural question is: does knowing a model on the original tables help in inducing a model on the result of a query?

Slightly more formally, let M_{DB} be the model we induced from database DB and let Q be a query on DB. Does knowing M_{DB} help in inducing a model M_Q on $Q(DB)$, i.e., on the result of Q when applied to DB. For example, if M_{DB} is a classifier and

Arno Siebes · Diyah Puspitaningrum
Department Of Information and Computing Sciences, Universiteit Utrecht, The Netherlands
e-mail: {arno,diyah}@cs.uu.nl

S. Džeroski, *Inductive Databases and Constraint-Based Data Mining,*
DOI 10.1007/978-1-4419-7738-0_13,

Q selects a subset of DB, does knowing M_{DB} help the induction of a new classifier M_Q on the subset $Q(DB)$?

This formulation is only slightly more formal as the term "help" is a non-technical and, thus, ill-defined concept. In this chapter, we will formalise "help" in two different ways. Firstly, in the sense that we can compute M_Q directly from M_{DB} *without* consulting either DB or $Q(DB)$. While this is clearly the most elegant way to formalise "help", it puts such stringent requirements on the class of models we consider that the answer to our question becomes *no* for many interesting model-classes; we'll exhibit one in this chapter.

Hence, secondly, we interpret "help", far less ambitiously, as meaning "speeding-up" the computation of M_Q. That is, let $\mathscr{A}lg$ be the algorithm used to induce M_{DB} from DB, i.e., $\mathscr{A}lg(DB) = M_{DB}$. We want to transform $\mathscr{A}lg$ into an algorithm $\mathscr{A}lg^*$, which takes M_{DB} as extra input such that

$$\mathscr{A}lg^*(Q(DB), M_{DB}) \approx \mathscr{A}lg(Q(DB))$$

Note that we do not ask for exactly the same model, approximately the same answer is acceptable if the speed-up is considerable. In fact, for many application areas, such as marketing, a *good enough model* rather than the *best model* is all that is required.

The problem as stated is not only relevant in the context of inductive databases, but also in existing data mining practice. In the data mining literature, the usual assumption is that we are given some database that has to be mined. In practice, however, this assumption is usually not met. Rather, the construction of the mining database is often one of the hardest parts of the KDD process [9]. The data often resides in a data warehouse or in multiple databases, and the mining database is constructed from these underlying databases.

From most perspectives, it is not very interesting to know whether one mines a specially constructed database or an original database. For example, if the goal is to build the best possible classifier on that data set, the origins of the database are of no importance whatsoever.

It makes a difference, however, if the underlying databases have already been modelled. Then, like with inductive databases, one would hope that knowing such models would help in modelling the specially constructed mining database. For example, if we have constructed a classifier on a database of customers, one would hope that this would help in developing a classifier for the female customers only.

In other words, the problem occurs both in the context of inductive databases and in the everyday practice of data miners. Hence, it is a relevant problem, but isn't it trivial? After all, if M_{DB} is a good model on DB, it is almost always also a good model on a random subset of DB; almost always, because a random subset may be highly untypical. The problem is, however, *not* trivial because queries in general do *not* compute a random subset. Rather, queries construct a very specific result.

For the usual "project-select-join" queries, there is not even a natural way in which the query-result can be seen as subset of the original database. Even if Q is just a "select"-query, the result is usually not random and M_{DB} can even be highly

misleading on $Q(DB)$. This is nicely illustrated by the well-known example of *Simpson's Paradox* on Berkeley's admission data [3]. Overall, 44% of the male applicants were admitted, while only 35% of the females were admitted. Four of the six departments, however, have a bias that is in favour of female applicants. While the overall model may be adequate for certain purposes, it is woefully inadequate for a query that selects a single department.

In other words, we do address a relevant and non-trivial problem. Addressing the problem, in either sense of "help", for all possible model classes and/or algorithms is, unfortunately, too daunting a task for this chapter. In the sense of "direct construction" it would require a discussion of all possible model classes, which is too large a set to consider (and would result in a rather boring discussion). In the "speed-up and approximation" sense it would require either a transformation of all possible induction algorithms or a generic transformation that would transform any such algorithm to one with the required properties. The former would, again, be far too long, while a generic transformation is unlikely to exist.

Therefore we restrict ourselves to one type of model, i.e., frequent item sets [1] and one induction algorithm, i.e., our own KRIMP algorithm [14]. The structure of this chapter is as follows. In the next section, Section 13.2, we introduce our data, models -that is code-tables-, and the KRIMP algorithm. Next, in Section 13.3 we investigate the "direct computation" interpretation of "help" in the context of frequent item set mining. This is followed in Section 13.4 by the introduction of a transformed variant of KRIMP for the "speed-up" interpretation of "help". In Section 13.5, we discuss and compare these two approaches. The chapter ends with conclusions and prospects for further research.

13.2 Preliminaries

In this section we give a brief introduction to the data, models, and algorithms as used in this chapter.

13.2.1 Data

In this chapter we restrict ourselves to databases with categorical data only, the biggest impact being that we do not consider real-valued attributes. Moreover, rather than using the standard representation for relational databases, we represent them as transaction databases familiar from item set mining. After briefly introducing such databases, we will briefly discuss how a (categorical) relational database can be transformed into such a transaction database. Moreover, for each relational algebra operator, we will briefly discuss how they should be interpreted in the transaction setting.

13.2.1.1 Transaction Databases

The problem of frequent item set mining [1] can be described as follows. The basis is a set of items $\mathscr{I}$, e.g., the items for sale in a store; $|\mathscr{I}| = n$. A transaction t is a set of items, i.e., $t \in \mathscr{P}(\mathscr{I})$ in which $P(X)$ denotes the power set of X. For example, t represents the set of items a client bought at the store. A table (normally called a database) over $\mathscr{I}$ is simply a bag of transactions, e.g., the different sale transactions in the store on a given day.

A transaction database is a set of transaction tables that is related through the familiar key-foreign key mechanism known from the relational model [5]. Without loss of generality we assume that there is at most one key-foreign key relation between any two tables. That is, we assume that the *join* between two tables is unambiguous without explicit key-foreign key identification.

An item set $I \subset \mathscr{I}$ occurs in a transaction $t \in T$ iff $I \subseteq t$. The *support* of I in T, denoted by $sup_T(I)$ is the number of transactions in the table in which t occurs. The problem of frequent item set mining is: given a threshold *min-sup*, determine all item sets I such that $sup_T(I) \geq$ *min-sup*. These *frequent item sets* represent, e.g., sets of items customers buy together often enough.

Based on the A Priori property,

$$I \subseteq J \Rightarrow sup_T(I) \geq sup_T(J),$$

reasonably efficient frequent item set miners exist.

13.2.1.2 Relational Databases as Transaction Databases

Transforming a relational database into a transaction database is straight-forward. Let T be a table in the relational database DB, having (non-key) attributes $A_1, \ldots, A_k$. Let the (finite!) domain of A_i be $D_i = \{d_{i,1}, \ldots d_{i,m_i}\}$. Then we define the set of items $\mathscr{I}_{T,i} = \{A_i = d_{i,1}, \ldots, A_i = d_{i,m_i}\}$. Moreover, define $\mathscr{I}_T = \bigcup_{i \in \{1,\ldots,k\}} \mathscr{I}_{T,i}$ and, obviously, $\mathscr{I} = \bigcup_{T \in DB} \mathscr{I}_T$.

The "transactified" table T' is then defined over the items in $\mathscr{I}_T$. The "transactified" version $t' \in T'$ of a $t \in T$ is given by:

$$"A_i = d''_{i,j} \in t' \Leftrightarrow t.A_i = d_{i,j}$$

The keys and foreign keys of T are simply copied in T'.

Note that this is not the most efficient way to encode a relational database as a transaction database. However, the efficiency of this encoding is irrelevant in this chapter. Moreover, while being inefficient, it is the most intuitive encoding; which is far more important for the purposes of this chapter.

From now on, we assume that all our databases are transaction databases.

13.2.1.3 Relational Algebra on Transaction Databases

To investigate models on the results of queries, we have to make our query language precise. Since we focus on relational databases -albeit in their "transactified" form- a relational query language is the obvious choice. Of these query languages, relational algebra is the most suited. More precisely, we focus on the usual "select-project-join" queries. That is, on the selection operator σ, the projection operator π, and the (equi-)join operator $\bowtie$; see [5].

We interpret these operators on transactions in the intuitive way. That is, σ selects those transactions that satisfy the selection predicate. The projection π returns that part of each transaction that is specified by the projection predicate. That is, we do not take the original relational representation into account. More in particular, this means that we, e.g., project on $A_i = d_{i,j}$ rather than on A_i. The former is more natural in the transaction context and the latter can easily be simulated by the former.

Finally the join is computed using key-foreign key relations only. That is, $\bowtie$ itself does not have items -attribute-value pairs- in its predicate. The reason is that such further selections can easily be accomplished using σ

Two final remarks on the queries in this chapter are the following, Firstly, as usual in the database literature, we use *bag* semantics. That is, we do allow duplicates tuples in tables and query results.

Secondly, as mentioned in the introduction, the mining database is constructed from *DB* using queries. Given the compositionality of the relational algebra, we may assume, again without loss of generality, that the analysis database is constructed using one query Q. That is, the analysis database is $Q(DB)$, for some relational algebra expression Q. Since *DB* is fixed, we will often simply write Q for $Q(DB)$; that is, we will use Q to denote both the query and its result.

13.2.2 Models

In this paper we consider two different types of models. The first is simply the set of all frequent item sets. The second are the models as computed by our KRIMP algorithm [14]. Since this later kind of model is less well-known, we provide a brief review of thses models.

The models computed by KRIMP consist of two components. First a constant -the same for all possible models- component, the COVER algorithm. Second a variable -database dependent- component, a *code table.*

Given a prefix code $\mathscr{C}$ a code table CT over $\mathscr{I}$ and $\mathscr{C}$ is a two-column table containing item sets and codes such that:

- each $I \in \mathscr{P}(\mathscr{I})$ and each $C \in \mathscr{C}$ occurs at most once in CT
- all the singleton item sets occur in CT
- The item sets in the code table are ordered descending on 1) item set length and 2) support size and 3) lexicographically.

Slightly abusing notation we say $I \in CT$ and $C \in CT$.

To encode a database with a code table, each transaction is partitioned into item sets in the code table:

COVER(CT,t)
If there exists $I \in CT$ such that $I \subseteq t$
  Then $Res := \{I\}$ where I is the first such element
    If $t \setminus I \neq \emptyset$
      Then $Res := Res \cup$ COVER$(CT, t \setminus I)$
  Else Fail
Return Res

D can be encoded by CT using COVER in the obvious way:

- compute the cover of each transaction $t \in D$
- replace each $I \in$ COVER(CT,t) by its code and concatenate these codes

Decoding is similarly easy because C is a prefix code:

- determine the codes in the code string
- take the union of the item sets that belong to these codes

Defined in this way, not all code tables are equally satisfying as a model of a given database DB. For, CT may assign very long codes to things that occur very often in DB, while it may assign very short codes to rare things. This is clearly unsatisfactory. We want the encoding to be optimal given the item sets in the code table.

The *usage* of an $I \in CT$ while coding DB is defined by:

$$usage(I) = |\{t \in DB | I \in \text{ COVER}(CT,t)\}|$$

Usage yields a probability distribution on the $I \in CT$:

$$\mathbb{P}(I) = \frac{usage(I)}{\sum_{J \in CT} usage(J)}$$

A Shannon code, which always exists [7], for CT is a prefix code with:

$$length(code(I)) = -\log(\mathbb{P}(I))$$

Such a code is optimal in the sense that the more often a code is used, the shorter its length is. From now on we assume that the code tables we consider have such Shannon-codes for database DB.

13.2.3 Algorithms

To induce the frequent item sets used in Section 13.3 we simply use one of the well-known frequent item set miners. For the code tables used in Section 13.4 we use our

KRIMP algorithm, since this is not as well-known, we provide a brief introduction here. For a more detailed description please refer to [14].

13.2.3.1 MDL for Code Tables

Even if all code tables we consider have Shannon optimal codes, not all such code tables are equally good models for *DB*. For example, there is one that contains the singleton item sets only. This is a model that specifies nothing about the correlation between the various items. To determine the best code table, we use the Minimum Description Length principle (MDL).

MDL [10] embraces the slogan *Induction by Compression.* It can be roughly described as follows.

Given a set of models[1] $\mathscr{H}$, the best model $H \in \mathscr{H}$ is the one that minimises

$$L(H) + L(D|H)$$

in which

- $L(H)$ is the length, in bits, of the description of H, and
- $L(D|H)$ is the length, in bits, of the description of the data when encoded with H.

One can paraphrase this by: the smaller $L(H) + L(D|H)$, the better H models D. In our terminology we want the code table that compresses *DB* best.

We already know how to compute the size of the compressed database. Simply encode *DB* and add the lengths of all the codes, which are Shannon optimal. That is,

$$L(DB|CT) = -\sum_{I \in CT : freq(I) \neq 0} usage(I) \log(\mathbb{P}(I))$$

Note that the stipulation $freq(I) \neq 0$ is only there because we require that all singleton item sets are present in CT. All other item sets are only present in CT if they are actually used.

Similarly, we already know the size in bits of the second column of CT, it is simply the sum of the sizes of all codes in *DB*. So, we only have to determine the size in bits of the first column, i.e,. of all the item sets in CT.

To determine that size we encode those item sets with the code table for *DB* that consists of the singleton item sets only.

- this means we can reconstruct D up to the actual label of the $i \in \mathscr{I}$.

This is actually a good feature. It means, among other things, that the model we find does not depend on the actual language used to describe the data.

The size of the left-hand column is the sum of these encoded sizes, The size of CT, denoted by $L(CT)$ is simply the sum of the sizes of the two columns. Hence, for a given database *DB* we have:

[1] MDL-theorists tend to talk about *hypothesis* in this context, hence the $\mathscr{H}$; see [10] for the details.

$$\mathscr{L}(CT, DB) = L(CT) + L(DB|CT)$$

Note that we omit the size of COVER as it is the same for all databases and code tables. That is, it is just an additive constant, which does not influence the search for the optimal model.

13.2.3.2 KRIMP

Unfortunately, finding the best code table is too expensive. Therefore we use a heuristic algorithm called KRIMP. KRIMP starts with a valid code table (only the collection of singletons) and a sorted list of candidates (frequent item sets). These candidates are assumed to be sorted descending on 1) support size, 2) item set length and 3) lexicographically. Each candidate item set is considered by inserting it at the right position in CT and calculating the new total compressed size. A candidate is only kept in the code table iff the resulting total size is smaller than it was before adding the candidate. If it is kept, all other elements of CT are reconsidered to see if they still positively contribute to compression. The whole process is illustrated in Figure 13.1; see [14]. If we assume a fixed minimum support threshold for a database, KRIMP has only one essential parameter: the database. For, given the database and the (fixed) minimum support threshold, the candidate list is also specified. Hence, we will simply write CT_{DB} and KRIMP(DB), to denote the code table induced by KRIMP from DB. Similarly CT_Q and KRIMP(Q) denote the code table induced by KRIMP from the result of applying query Q to DB.

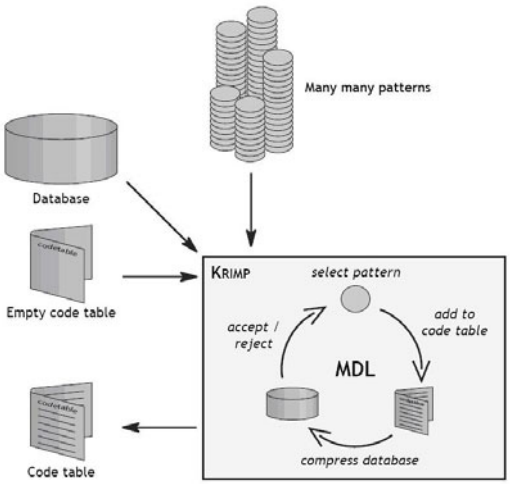


Fig. 13.1 KRIMP in action

13.3 Frequent Item Set Mining

The goal of this section is to investigate whether we can determine the set of frequent item sets on $Q(DB)$ without consulting $Q(DB)$. Rather, we are given the frequent item sets on DB and the query Q and from that only we should determine the frequent item sets on $Q(DB)$. That is, we want to *lift* the relational operators to sets of frequent item sets.

13.3.1 Selection

The relational algebra operator σ (select) is a mapping:

$$\sigma : \mathscr{B}(D) \rightarrow \mathscr{B}(D)$$

in which $\mathscr{B}(D)$ denotes all possible bags over domain D.

Lifting means that we are looking for an operator $\sigma_{(D,\mathscr{A}lg)}$ that makes the diagram in Figure 13.2 commute: Such diagrams are well-known in , e.g., category theory [2] and the standard interpretation is:

$$\mathscr{A}lg \circ \sigma = \sigma_{(D,\mathscr{A}lg)} \circ \mathscr{A}lg$$

In other words, first inducing the model using algorithm $\mathscr{A}lg$ followed by the application of the *lifted* selection operator $\sigma_{(D,\mathscr{A}lg)}$ yields the same result as first applying the *standard* selection operator σ followed by induction with algorithm $\mathscr{A}lg$.

In fact, we are willing to settle for commutation of the diagram in a loose sense: That is, if we are able to give reasonable support bounds for those item sets whose support we can not determine exactly, we are satisfied.

For frequent item sets the three basic selections are $\sigma_{I=0}$, $\sigma_{I=1}$, and $\sigma_{I_1=I_2}$. More complicated selections can be made by conjunctions of these basic comparisons. We look at the different basic selections in turn.

First consider $\sigma_{I=0}$. If it is applied to a table, all transactions in which I occurs are removed from that table. Hence, all item sets that contain I get a support of zero in the resulting table. For those item sets in which I doesn't occur, we have to compute which part of their support consists of transactions in which I does occur and subtract that number. Hence, for support for item sets J, we have:

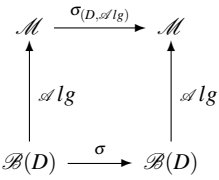


Fig. 13.2 Lifting the selection operator

$$sup_{\sigma_{I=0}(T)}(J) = \begin{cases} 0 & \text{if } I \in J, \\ sup_T(J) - sup_T(J \cup \{I\}) & \text{otherwise.} \end{cases}$$

If we apply $\sigma_{I=1}$ to the table, all transactions in which I doesn't occur are removed from the table. In other words, the support of item sets that contain I doesn't change. For those item sets that do not contain I, the support is given by those transactions that *also* contained I. Hence, we have:

$$sup_{\sigma_{I=1}(T)}(J) = \begin{cases} sup_T(J) & \text{if } I \in J, \\ sup_T(J \cup \{I\}) & \text{otherwise.} \end{cases}$$

If we apply $\sigma_{I_1=I_2}$ to the table, the only transactions that remain are those that either contain both I_1 and I_2 or neither. In other words, for frequent item sets that contain both, the support remains the same. For all others, the support changes. For those item sets J that contain just one of the I_i the support will be the support of $J \cup \{I_1, I_2\}$. For those that contain neither of the I_i, we have to correct for those transactions that contain one of the I_i in their support. If we denote this by $sup_T(J \neg I_1 \neg I_2)$ (a support that can be easily computed) We have:

$$sup_{\sigma_{I_1=I_2}(T)}(J) = \begin{cases} sup_T(J \cup \{I_1, I_2\}) & \text{if } \{I_1, I_2\} \cap J \neq \emptyset, \\ sup_T(J \neg I_1 \neg I_2) & \text{otherwise.} \end{cases}$$

Clearly, we can also "lift" conjunctions of the basic selections, simply processing one at the time. So, in principle, we can lift all selections for frequent item sets. But only in principle, because we need the support of item sets that are *not necessarily frequent*. Frequent item sets are a lossy model (not all aspects of the data distribution are modelled) and that can have its repercussions: in general the lifting will *not* be commutative. In our loose sense of "commutativity", the situation is slightly better. For, we can give reasonable bounds for the resulting supports; for those supports we do not know are bounded (from above) by *min-sup*.

We haven't mentioned constraints [11] so far. Constraints in frequent item set mining are the pre-dominant way to select a subset of the frequent item sets. In general the constraints studied do not correspond to selections on the database. The exception is the class of *succinct anti-monotone constraints* introduced in [11]. For these constraints there is such a selection (that is what succinct means) and the constraint can be pushed into the algorithm. This means we get the commutative diagram in Figure 13.3. Note that in this case we know that the diagonal arrow

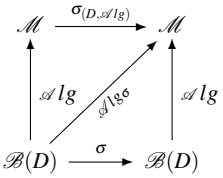


Fig. 13.3 Lifting selections for succinct constraints

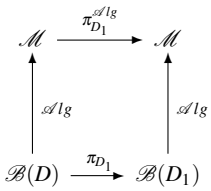


Fig. 13.4 Lifting projections

makes the bottom right triangle commute in the strict sense of the word. For the upper left triangle, as well as the square, our previous analysis remains true.

13.3.2 Project

For the projection operator π, we have a new domain D_1 such that $D = D_1 \times D_2$. Projection on D_1 has thus as signature:

$$\pi_{D_1} : \mathscr{B}(D) \to \mathscr{B}(D_1)$$

Hence, we try to find an operator $\pi_{D_1}^{\mathscr{A}lg}$ that makes the diagram in Figure 13.4 commute. Note that D_1 is spanned by the set of variables (or items) we project on.

We project on a set of items $\mathscr{J} \subseteq \mathscr{I}$, let $J \subseteq \mathscr{I}$ be a frequent item set. There are three cases to consider:

1. if $J \subseteq \mathscr{J}$, then all transactions in the support of J will simply remain in the table, hence J will remain frequent.
2. if $J \cap \mathscr{J} \neq \emptyset$, then $J \cap \mathscr{J}$ is also frequent and will remain in the set of frequent item sets.
3. if $J \cap \mathscr{J} = \emptyset$, then its support will vanish.

In other words, if $\mathscr{F}$ denotes the set of all frequent item sets, then:

$$\pi_{\mathscr{J}}(\mathscr{F}) = \{J \in \mathscr{F} | J \subseteq \mathscr{J}\}$$

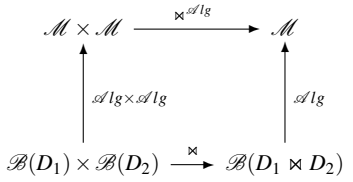
Clearly, this method of lifting will make the diagram commute in the strict sense if we use absolute minimal frequency. In other words, for projections, frequent item sets do capture enough of the underlying data distribution to allow lifting.

13.3.3 EquiJoin

The equijoin has as signature:

$$\bowtie : \mathscr{B}(D_1) \times \mathscr{B}(D_2) \to \mathscr{B}(D_1 \bowtie D_2)$$

Fig. 13.5 Lifting the equijoin



Hence, the diagram we want to make commute is given in Figure 13.5. The join can be computed, though not very efficiently, starting with the Cartesian product of the two tables. Since in extreme cases, the equi-join equals the Cartesian product, we discuss that operator.

Let J_1 be a frequent item set for the first table and J_2 for the second. The frequency of the pair on the Cartesian product of the two tables is simply given by:

$$sup_{T_1 \times T_2}(J_1, J_2) = sup_{T_1}(J_1) \times sup_{T_2}(J_2)$$

While this is easy to compute, it means again that in general we will not be able to compute all frequent item sets on the Cartesian product without consulting the database. Even if we set the minimal frequency to the product of the two minimal frequencies, the combination of an infrequent item set on one database with a frequent one on the other may turn out to be frequent.

In other words, we cannot even make the diagram commute in the approximate sense of the word. For, the bound is given by $\max\{|T_1| \times (\text{min-sup} - 1), |T_2| \times (\text{min-sup} - 1)\}$, which is hardly a reasonable bound.

Given that the number of joins possible in a database is limited and known beforehand, we may make our lives slightly easier. That is, we may allow ourselves to do some pre-computations.

Assume that we compute the tables $T_1^2 = \pi_{T_1}(T_1 \ltimes T_2)$ and $T_2^1 = \pi_{T_2}(T_1 \ltimes T_2)$ and their frequent item sets, say $\mathscr{F}_1^2$ and $\mathscr{F}_2^1$, off-line. Are those sets enough to lift the join? For the extreme case, the Cartesian product, the answer is clearly: *yes*. By "blowing" up the original tables we add enough information to compute the support of any item set in the join *iff* that item set exceeds the minimal support.

Unfortunately, the same is not true for the join in general. Since we cannot see from either $\mathscr{F}_1^2$ or $\mathscr{F}_2^1$ which combinations of frequent item sets will actually occur in $(T_1 \ltimes T_2)$. That is, we can only compute a superset of the frequent item sets on the join.

Hence, the only way to lift the join is to compute and store the frequent item sets on all possible joins. While this is doable given the limited number of possible joins, this can hardly count as lifting.

13.3.4 Discussion

The fact that lifting the relational algebra operators to sets of frequent item sets is only partially possible should hardly come as a surprise: the *min-sup* constraint makes this into an inherently lossy model. For models that do try to capture the complete distribution, such as Baysian networks, one would expect far better results; see [12] for a discussion of lifting for such networks.

13.4 Transforming KRIMP

Recall from the Introduction that the problem we investigate in this Section is that we want to transform an induction algorithm $\mathscr{A}lg$ into an algorithm $\mathscr{A}lg^*$ that takes at least two inputs, i.e, both Q and $\mathscr{M}_{DB}$, such that:

1. $\mathscr{A}lg^*$ gives a reasonable approximation of $\mathscr{A}lg$ when applied to Q, i.e.,

$$\mathscr{A}lg^*(Q, \mathscr{M}_{DB}) \approx \mathscr{M}_Q$$

2. $\mathscr{A}lg^*(Q, \mathscr{M}_{DB})$ is simpler to compute than $\mathscr{M}_Q$.

The second criterion is easy to formalise: the runtime of $\mathscr{A}lg^*$ should be shorter than that of $\mathscr{A}lg$. The first one is harder. What do we mean that one model is an approximation of another? Moreover, what does it mean that it is a *reasonable* approximation?

Before we discuss how KRIMP can be transformed and provide experimental evidence that our approach works, we first formalise this notion of approximation.

13.4.1 Model Approximation

The answer to the question of how to formalise that one model approximates another depends very much on the goal. If $\mathscr{A}lg$ induces classifiers, approximation should probably be defined in terms of prediction accuracy, e.g., on the Area Under the ROC-curve (AUC).

KRIMP computes code tables. Hence, the quick approximating algorithm we are looking for, KRIMP* in the notation used above, also has to compute code tables. So, one way to define the notion of approximation is by comparing the resulting code tables. Let CT_{KRIMP} be the code table computed by KRIMP and similarly, let CT_{KRIMP^*} denote the code table computed by KRIMP* on the same data set. The more similar CT_{KRIMP^*} is to CT_{KRIMP}, the better KRIMP* approximates KRIMP.

While this is intuitively a good way to proceed, it is far from obvious how to compare two code tables. Fortunately, we do not need to compare code tables directly. KRIMP is based on MDL and MDL offers another way to compare models,

i.e., by their *compression-rate*. Note that using MDL to define "approximation" has the advantage that we can formalise our problem for a larger class of algorithms than just KRIMP. It is formalised for all algorithms that are based on MDL. MDL is quickly becoming a popular formalism in data mining research, see, e.g., [8] for an overview of other applications of MDL in data mining.

What we are interested in is comparing two algorithms on the same data set, i.e., on $Q(DB)$. Slightly abusing notation, we will write $\mathscr{L}(\mathscr{A}lg(Q))$ for $L(\mathscr{A}lg(Q)) + L(Q(DB)|\mathscr{A}lg(Q))$, similarly, we will write $\mathscr{L}(\mathscr{A}lg^*(Q,\mathscr{M}_{DB}))$. Then, we are interested in comparing $\mathscr{L}(\mathscr{A}lg^*(Q,\mathscr{M}_{DB}))$ to $\mathscr{L}(\mathscr{A}lg(Q))$. The closer the former is to the latter, the better the approximation is.

Just taking the difference of the two, however, can be quite misleading. Take, e.g., two databases db_1 and db_2 sampled from the same underlying distribution, such that db_1 is far bigger than db_2. Moreover, fix a model H. Then necessarily $L(db_1|H)$ is bigger than $L(db_2|H)$. In other words, big absolute numbers do not necessarily mean very much. We have to *normalise* the difference to get a feeling for how good the approximation is. Therefore we define the asymmetric dissimilarity measure (ADM) as follows [15].

Definition 13.1. Let H_1 and H_2 be two models for a dataset D. The asymmetric dissimilarity measure $ADM(H_1,H_2)$ is defined by:

$$ADM(H_1,H_2) = \frac{|\mathscr{L}(H_1) - \mathscr{L}(H_2)|}{\mathscr{L}(H_2)}$$

Note that this dissimilarity measure is related to the Normalised Compression Distance [4]. The reason why we use this asymmetric version is that we have a "gold standard". We want to know how far our approximate result $\mathscr{A}lg^*(Q,\mathscr{M}_{DB})$ deviates from the optimal result $\mathscr{A}lg(Q)$.

The remaining question is, of course, what ADM scores indicate a good approximation? In a previous paper [15], we took two random samples from data sets, say D_1 and D_2. Code tables CT_1 and CT_2 were induced from D_1 and D_2 respectively. Next we tested how well CT_i compressed D_j. For the four data sets also used in this paper, *Iris, Led7, Pima* and, *PageBlocks*, the "other" code table compressed 16% to 18% worse than the "own" code table; the figures for other data sets are in the same ball-park. In other words, an ADM score of 0.2 is in-line with the "natural variation" in a data set. If it gets much higher, it shows that the two code tables are rather different.

Clearly, $ADM(\mathscr{A}lg^*(Q,\mathscr{M}_{DB}),\mathscr{A}lg(Q))$ does not only depend on $\mathscr{A}lg^*$ and on $\mathscr{A}lg$, but also very much on Q. We do not seek a low *ADM* on one particular Q, rather we want to have a reasonable approximation on all possible queries. Requiring that the *ADM* is equally small on all possible queries seems too strong a requirement. Some queries might result in a very untypical subset of *DB*, the ADM is probably higher on the result of such queries than it is on queries that result in more typical subsets. Hence, it is more reasonable to require that the *ADM* is small most of the time. This is formalised through the notion of an (ε,δ)-approximation

Definition 13.2. Let DB be a database and let Q be a random query on DB. Moreover, let $\mathscr{A}lg_1$ and $\mathscr{A}lg_2$ be two data mining algorithms on DB. Let $\varepsilon \in \mathbb{R}$ be the threshold for the maximal acceptable ADM score and $\delta \in \mathbb{R}$ be the error tolerance for this maximum. $\mathscr{A}lg_1$ is an (ε,δ)-approximation of $\mathscr{A}lg_2$ iff

$$\mathbb{P}(ADM(\mathscr{A}lg_1(Q),\mathscr{A}lg_2(Q)) > \varepsilon) < \delta$$

13.4.2 Transforming KRIMP

Given that KRIMP results in a code table, there is only one sensible way in which KRIMP(DB) can be re-used to compute KRIMP(Q): provide KRIMP only with the item sets in CT_{DB} as candidates. While we change nothing to the algorithm, we'll use the notation KRIMP* to indicate that KRIMP got only code table elements as candidates. So, e.g., KRIMP$^*(Q)$ is the code table that KRIMP induces from $Q(DB)$ using the item sets in CT_{DB} only.

Given our general problem statement, we now have to show that KRIMP* satisfies our two requirements for a transformed algorithm. That is, we have to show for a random database DB:

- For reasonable values for ε and δ, KRIMP* is an (ε,δ)-approximation of KRIMP, i.e, for a random query Q on DB:

  $$\mathbb{P}(ADM(\text{KRIMP}^*(Q),\text{KRIMP}(Q)) > \varepsilon) < \delta$$

  Or in MDL-terminology:

  $$\mathbb{P}\left(\frac{|\mathscr{L}(\text{KRIMP}^*(Q)) - \mathscr{L}(\text{KRIMP}(Q))|}{\mathscr{L}(\text{KRIMP}(Q))} > \varepsilon\right) < \delta$$

- Moreover, we have to show that it is faster to compute KRIMP$^*(Q)$ than it is to compute KRIMP(Q).

Neither of these two properties can be formally proven, if only because KRIMP and thus KRIMP* are both heuristic algorithms. Rather, we report on extensive tests of these two requirements.

13.4.3 The Experiments

In this subsection, we describe our experimental set-up. First we briefly describe the data sets we used. Next we discuss the queries used for testing. Finally we describe how the tests were performed.

13.4.3.1 The Data Sets

To test our hypothesis that KRIMP* is a good and fast approximation of KRIMP, we have performed extensive tests mostly on 6 well-known UCI [6] data sets and one data set from the KDDcup 2004.

In particular, we have used the data sets *connect, adult, chessBig, letRecog, PenDigits* and *mushroom* from UCI. These data sets were chosen because they are well suited for KRIMP. Some of the other data sets in the UCI repository are simply too small for KRIMP to perform well. MDL needs a reasonable amount of data to be able to function. Some other data sets are very dense. While KRIMP performs well on these very dense data sets, choosing them would have turned our extensive testing prohibitively time-consuming.

Since all these data sets are single table data sets, they do not allow testing with queries involving joins. To test such queries, we used tables from the "Hepatitis Medical Analysis"[2] of the KDDcup 2004. From this relational database we selected the tables *bio* and *hemat*. The former contains biopsy results, while the latter contains results on hematological analysis. The original tables have been converted to item set data and rows with missing data have been removed.

13.4.3.2 The Queries

To test our hypothesis, we need to consider randomly generated queries. On first sight this appears a daunting task. Firstly, because the set of all possible queries is very large. How do we determine a representative set of queries? Secondly, many of the generated queries will have no or very few results. If the query has no results, the hypothesis is vacuously true. If the result is very small, MDL (and thus KRIMP) doesn't perform very well.

To overcome these problems, we restrict ourselves to queries that are built by using selections (σ), projections (π), and joins ($\bowtie$) only. The rationale for this choice is twofold. Firstly, simple queries will have, in general, larger results than more complex queries. Secondly, we have seen in Section 13.3 that lifting these operators is already a problem.

13.4.3.3 The Experiments

The experiments preformed for each of the queries on each of the data sets were generated as follows.

Projection: The projection queries were generated by randomly choosing a set X of n items, for $n \in \{1,3,5,7,9\}$. The generated query is then $\pi_{\overline{X}}$. That is, the elements of X are projected out of each of the transactions. For example, $\pi_{\overline{\{I_1,I_3\}}}(\{I_1,I_2,I_3\}) = \{I_2\}$. For this case, the code table elements generated on

[2] http://lisp.vse.cz/challenge/

the complete data set were projected in the same way. For each value of n, 10 random sets X were generated on each data set.

As an aside, note that the rationale for limiting X to maximally 9 elements is that for larger values too many result sets became too small for meaningful results.

Selection: The random selection queries were again generated by randomly choosing a set X of n items, with $n \in \{1,2,3,4\}$. Next for each random item I_i a random value v_i (0 or 1) in its domain D_i was chosen. Finally, for each I_i in X a random $\theta_i \in \{=,\neq\}$ was chosen. The generated query is thus $\sigma(\bigwedge_{I_i \in X} I_i \theta_i v_i)$. As in the previous case, we performed 10 random experiments on each of the data sets for each of the values of n.

Project-Select: The random project-select queries generated are essentially combinations of the simple projection and selection queries as explained above. The only difference is that we used $n \in \{1,3\}$ for the projection and $n \in \{1,2\}$ for the selections. That is we select on 1 or 2 items and we project away either 1 or 3 items. The size of the results is, of course, again the rationale for this choice. For each of the four combinations, we performed 100 random experiments on each of the data sets: first we chose randomly the selection (10 times for each selection), for each such selection we performed 10 random projections.

Project-Select-Join: Since we only use one "multi-relational" data set and there is only one possible way to join the *bio* and *hemat* tables, we could not do random tests for the join operator. However, in combination with projections and selections, we can perform random tests. These tests consist of randomly generated project-select queries on the join of *bio* and *hemat*. In this two-table case, KRIMP* got as input all pairs $(\mathscr{I}_1, \mathscr{I}_2)$ in which $\mathscr{I}_1$ is an item set in the code table of the "blown-up" version of *bio*, and $\mathscr{I}_2$ is an item set in the code table of the "blown-up" version of *hemat*. Again we select on 1 or 2 items and we project away either 1 or 3 items. And, again, we performed again 100 random experiments on the database for each of the four combinations; as above.

13.4.4 The Results

In this subsection we give an overview of the results of the experiments described in the previous section. Each test query is briefly discussed in its own subsubsection.

13.4.4.1 Projection Queries

In Figure 13.6 the results of the random projection queries on the *letRecog* data set are visualised. The marks in the picture denote the averages over the 10 experiments, while the error bars denote the standard deviation. Note that, while not statistically significant, the average ADM grows with the number of attributes projected away. This makes sense, since the more attributes are projected away, the smaller the result

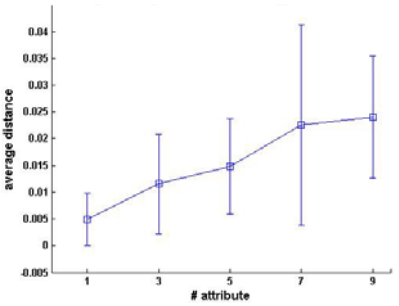


Fig. 13.6 Projection results on *letRecog*

set becomes. On the other data sets, KRIMP* performs similarly. Since this is also clear from the project-select query results, we do not provide all details here.

13.4.4.2 Selection Queries

The results of the random selection queries on the *penDigits* data set are visualised in Figure 13.7. For the same reason as above, it makes sense that the average ADM grows with the number of attributes selected on. Note, however, that the ADM averages for selection queries seem much larger than those for projection queries. These numbers are, however, not representative for the results on the other data sets. It turned out that *penDigits* is actually too small and sparse to test KRIMP* seriously. In the remainder of our results section, we do not report further results on *penDigits*. The reason why we report on it here is to illustrate that even on rather small and sparse data sets KRIMP* still performs reasonably well. On all other data sets KRIMP* performs far better, as will become clear next.

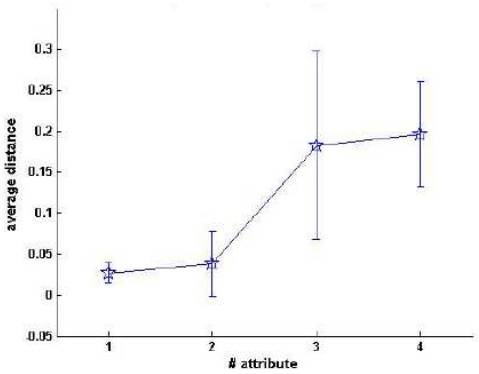


Fig. 13.7 Selection results on *penDigits*

Table 13.1 The results of Project-Select Queries

ADM ± STD		connect	adult	chessBig	letRecog	mushroom
Select 1	**Project out 1**	0.1 ± 0.01	0.1 ± 0.01	0.04 ± 0.01	0.1 ± 0.01	0.3 ± 0.02
	Project out 3	0.1 ± 0.02	0.1 ± 0.01	0.04 ± 0.03	0.1 ± 0.01	0.3 ± 0.16
Select 2	**Project out 1**	0.2 ± 0.01	0.1 ± 0.01	0.1 ± 0.03	0.04 ± 0.01	0.2 ± 0.04
	Project out 3	0.2 ± 0.02	0.1 ± 0.01	0.1 ± 0.03	0.04 ± 0.01	0.2 ± 0.05

13.4.4.3 Project-Select Queries

The results of the projection-select queries are given in Table 13.1. All numbers are the average ADM score ± the standard deviation for the 100 random experiments. All the ADM numbers are rather small, only for mushroom do they get above 0.2.

Two important observations can be made from this table. Firstly, as for the projection and selection queries reported on above, the ADM scores get only slightly worse when the query results get smaller: "Select 2, Project out 3" has slightly worse ADM scores than "Select 1, Project out 1". Secondly, even more importantly, combining algebra operators only degrades the ADM scores slightly. This can be seen if we compare the results for "Project out 3" on *letRecog* in Figure 13.6 with the "Select 1, Project out 3" and "Select 2, Project out 3" queries in Table 13.1 on the same data set. These results are very comparable, the combination effect is small and mostly due to the smaller result sets. While not shown here, the same observation holds for the other data sets.

To give insight in the distribution of the ADM scores of the "Select 2, Project out 3" queries on the *connect* data set are given in Figure 13.8. From this figure we see that if we choose $\varepsilon = 0.2$, $\delta = 0.08$. In other words, KRIMP* is a pretty good approximation of KRIMP. Almost always the approximation is less than 20% worse than the optimal result. The remaining question is, of course, how much faster is KRIMP*? This is illustrated in Table 13.2.

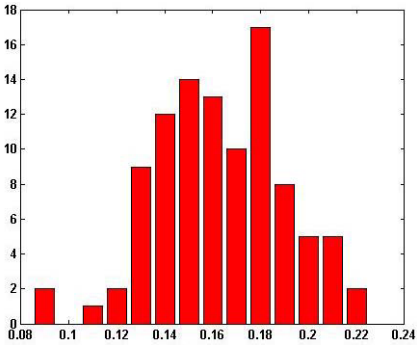


Fig. 13.8 Histogram of 100 Project-Select Queries on *connect*

Table 13.2 Relative number of candidates for KRIMP*

Relative #candidates		connect	adult	chessBig	letRecog	mushroom
Select 1	**Project out 1**	0.01 ± 0.001	0.01 ± 0.002	0.21 ± 0.012	0.01 ± 0.001	0.01 ± 0.001
	Project out 3	0.01 ± 0.001	0.01 ± 0.004	0.26 ± 0.031	0.02 ± 0.004	0.01 ± 0.001
Select 2	**Project out 1**	0.01 ± 0.001	0.03 ± 0.003	0.76 ± 0.056	0.02 ± 0.002	0.03 ± 0.002
	Project out 3	0.01 ± 0.002	0.03 ± 0.008	0.96 ± 0.125	0.02 ± 0.004	0.03 ± 0.003

Table 13.2 gives the average number of candidates KRIMP* has to consider relative to those that the full KRIMP run has to consider. Since, both KRIMP* and KRIMP are linear in the number of candidates, this table shows that the speed-up is considerable; a factor of 100 is often attained; except for *chessBig* were the query results get small and, thus, have few frequent item sets. The experiments are those that are reported on in Table 13.1.

13.4.4.4 Select-Project-Join Queries

The results for the select-project-join queries are very much in line with the results reported on above. In fact, they are even better. Since the join leads to rather large results, the ADM score is almost always zero: in only 15 of the 400 experiments the score is non-zero (average of non-zero values is 1%). The speed-up is also in line with the numbers reported above, a factor of 100 is again often attained.

13.4.5 Discussion

As noted in the previous section, the speed-up of KRIMP* is easily seen. The number of candidates that KRIMP* has to consider is often a factor 100 smaller than those that the full KRIMP run has to consider. Given that the algorithm is linear in the number of candidates, this means a speed-up by a factor 100. In fact, one should also note that for KRIMP*, we do not have to run a frequent item set miner. In other words, in practice, using KRIMP* is even faster than suggested by the Speed-up scores.

But, how about the other goal: how good is the approximation? That is, how should one interpret ADM scores? Except for some outliers, ADM scores are below 0.2. That is, a full-fledged KRIMP run compresses the data set 20% better than KRIMP*. As noted when we introduced the ADM score, this about as good as one can expect, such a percentage shows the natural variation in the data. Hence, given that the average ADM scores are often much lower we conclude that the approximation by KRIMP* is good.

In other words, the experiments verify our hypothesis: KRIMP* gives a fast and good approximation of KRIMP. The experiments show this for simple "project-select-join" queries, but as noticed with the results of the "project-select" queries,

the effect of combining algebra operators is small. If the result set is large enough, the approximation is good.

13.5 Comparing the two Approaches

In this chapter, we introduced two ways in which the models present in an inductive database DB help in computing the models on the results of a query Q on the data in that database. The first, if applicable, gives results without consulting $Q(DB)$. The result is computed directly from the models M_T induced on the tables used by Q. For the relational algebra we formalised this by lifting the relational algebra operators to the set of all models.

The second approach does allow access to $Q(DB)$. The induction algorithm $\mathscr{A}lg$ is transformed into an algorithm $\mathscr{A}lg^*$ that takes at least two inputs, i.e, both Q and $\mathscr{M}_{DB}$, such that:

1. $\mathscr{A}lg^*$ gives a reasonable approximation of $\mathscr{A}lg$ when applied to Q, i.e.,

$$\mathscr{A}lg^*(Q, \mathscr{M}_{DB}) \approx \mathscr{M}_Q$$

2. $\mathscr{A}lg^*(Q, \mathscr{M}_{DB})$ is simpler to compute than $\mathscr{M}_Q$.

The first requirement was formalised using MDL into the requirement:

$$\mathbb{P}\left(\frac{|\mathscr{L}(\mathscr{A}lg^*(Q)) - \mathscr{L}(\mathscr{A}lg(Q))|}{\mathscr{L}(\mathscr{A}lg(Q))} > \varepsilon\right) < \delta$$

for reasonably small ε and δ. The second requirement was simply interpreted as a significant speed-up in computation.

Clearly, when applicable, the first approach is to be preferred above the second approach. Firstly because it doesn't even require the computation of $Q(DB)$, and is, hence, likely to be much faster. Secondly, because an algebraic structure on the set of all models opens up many more possible applications.

In this chapter, we investigated both approaches on item sets. More precisely, we investigated lifting the relational algebra operators to sets of frequent item sets. Moreover, we transformed our KRIMP algorithm to investigate the second approach.

As noted already in Section 13.3, lifting the relational algebra operators to sets of frequent item sets has its problems. Only for the projection it works well. For the selection operator we get a reasonable approximation. Reasonable in the sense that we can put a bound on the error of the approximated support; an upper bound that is determined by the minimal support threshold. Since this bound is an upperbound, this means that we may declare too many item sets to be frequent. If we declare an item set to be infrequent, it is infrequent on the result of the selection.

The join operator, unfortunately, can not be lifted at all. Not even if we provide extra information by giving access to the frequent item sets on the "blown-up" version of the underlying tables. In that case, we again only have an upperbound on

the support. That is, again, we declare too many item sets to be frequent. In the case of the join, however, there is no bound on the error. For, if I_1 has a high support on $T_1^2 = \pi_{T_1}(T_1 \bowtie T_2)$, say n_1, while I_2 has a high support on $T_2^1 = \pi_{T_2}(T_1 \bowtie T_2)$, say n_2, then the computed upperbound on the support of (I_1, I_2) on $T_1 \bowtie T_2$ will be $n_1 \times n_2$, while there may be no transaction in $T_1 \bowtie T_2$ which actually supports this pair! Again, if we declare an item set to be infrequent on the join, it is infrequent.

Again as noted before, the reason for this failure is that sets of frequent item sets are an inherently lossy model. As our analysis above shows, this loss of information makes us overestimate the support of item sets on $Q(DB)$, in the case of the join with an unbounded error.

The transformation of KRIMP proved to be far more successful. The algorithm KRIMP*, which is simply KRIMP with a restricted set of candidates proved in the experiments to be much faster and provide models which approximate the true model very well. Given the lack of success for frequent item sets, this is a surprising result.

For, from earlier research [15] we know that the code tables produced by KRIMP determine the support of all item sets rather accurately. More precisely, in that paper we showed that these code tables can be used to generate a new code table. The support of an arbitrary frequent item set in this generated database, say DB_{gen}, is almost always almost equal to the support of that item set in the original database, say DB_{orig}. As usual, this sentence is probably more clear in its mathematical formulation:

$$\mathbb{P}\left(|sup_{DB_{orig}}(I) - sup_{DB_{gen}}(I)| > \varepsilon\right) < \delta$$

This surprise raises two immediate questions:

1. Why does transforming KRIMP work and
2. Can we transform frequent item set mining?

The reason that transforming KRIMP work is firstly exactly the fact that it determines the support of all item sets so well. Given a code table, which KRIMP* produces, we know the support of these item sets. Clearly, as for the set of frequent item sets, this means that we will overestimate the support of item sets on the result of a query. However, different from the lifting approach, we do allow access to the query result and, hence, the overestimation can be corrected. This is the second reason why transforming KRIMP works.

This reasoning makes the question "Can we transform item set mining?" all the more relevant. Unfortunately, the answer to this question is *probably not*. This can be easily seen from the join. The input for the transformed item set miner would be the joined tables as well as the Cartesian product of the sets of frequent item sets on the "blown-up" individual tables. This set of candidate frequent item sets will be **prohibitively large**, far larger than the final set of item sets that is frequent on the join. Hence, checking all these candidates will be more expensive than computing only the frequent ones efficiently.

Pruning the set of candidates while searching for the frequent ones requires a data structure that stores all candidates. Whenever, we can prune, a set of candidates has to be physically deleted from this data structure. The normal item set miners do not even generate most of these pruned candidates. In this approach we would

first generate and then delete them. In other words, it is highly unlikely that this approach will have a performance similar to the best item set miners. Let alone that it will be significantly more efficient than these algorithms, as is required by the transformation approach.

In turn, this reasoning points to the third reason why transforming KRIMP works. The code tables KRIMP produces are small, far smaller than the set of frequent item sets. Hence, checking the support of all candidates suggested by KRIMP is not detrimental for the efficiency of KRIMP*.

From this discussion we can derive the following succinct all-encompassing reason why transforming KRIMP works. KRIMP produces, relatively, small code tables that capture the support of all item sets rather well, such that checking the set of all suggested candidates is rather cheap.

Note that the comparison of the two approaches for a single case, i.e., that of item sets does not imply at all that the second approach is inherently superior to the first one. In fact, we already argued at the start of this section that the first approach, if applicable, is to be preferred above the second one. Moreover, in [12] we argued that the first approach is applicable for the discovery of Bayesian networks from data. In other words, the first approach is a viable approach.

A conclusion we can, tentatively, draw from the discussion in this section is that for either approach to work, the models should capture the data distribution well.

13.6 Conclusions and Prospects for Further Research

In this chapter we introduced a problem that has received little attention in the literature on inductive databases or in the literature on data mining in general. This question is: does knowing models on the database help in inducing models on the result of a query on that database?

We gave two approaches to solve this problem, induced by two interpretations of "help". The first, more elegant, one produces results without access to the result of the query. The second one does allow access to this result.

We investigated both approaches for item set mining. It turned out that the first approach is not applicable to frequent item set mining, while the second one produced good experimental results for our KRIMP algorithm. In Section 13.5 we discussed this failure and success. The final tentative conclusion of this discussion is: for either approach to work, the models should capture the data distribution well.

This conclusion points directly to other classes of models that may be good candidates for either approach, i.e., those models that capture a detailed picture of the data distribution. One example are Bayesian networks already discussed in [12]. Just as interesting, if not even more, are models based on bagging or boosting or similar approaches. Such models do not concentrate all effort on the overall data distribution, but also take small selections with their own distribution into account. Hence, for such models one would expect that, e.g., lifting the selection operator should be relatively straight forward.

This is an example for a much broader research agenda: For which classes of models and algorithms do the approaches work? Clearly, we have only scratched the surface of this topic. Another, similarly broad, area for further research is: Are there other, better, ways to formalise "help"?

References

1. Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI, 1996.
2. Andrea Asperti and Giuseppe Longo. *Categories, Types, and Structures*. MIT Press, 1991.
3. P.J. Bickel, E.A. Hammel, and J.W. O'Connell. Sex bias in graduate admissions: Data from berkeley. *Science*, 187(4175):398–404, 1975.
4. Rudi Cilibrasi and Paul Vitanyi. Automatic meaning discovery using google. In *IEEE Transactions on Knowledge and Data Engineering*, volume 19, pages 370–383. 2007.
5. E.F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.
6. Frans Coenen. The LUCS-KDD discretised/normalised ARM and CARM data library: http://www.csc.liv.ac.uk/˜frans/ KDD/Software/LUCS_KDD_DN/. 2003.
7. T.M. Cover and J.A. Thomas. *Elements of Information Theory, 2nd ed.* John Wiley and Sons, 2006.
8. C. Faloutsos and V. Megalooikonomou. On data mining, compression and kolmogorov complexity. In *Data Mining and Knowledge Discovery*, volume 15, pages 3–20. Springer Verlag, 2007.
9. Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. 1996.
10. Peter D. Grünwald. Minimum description length tutorial. In P.D. Grünwald and I.J. Myung, editors, *Advances in Minimum Description Length*. MIT Press, 2005.
11. Raymond T. Ng, Laks V. S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. ACM SIGMOD conference*, 1998.
12. Arno Siebes. Data mining in inductive databases. In Francesco Bonchi and Jean-François Boulicaut, editors, *Knowledge Discovery in Inductive Databases, 4th International Workshop, KDID 2005, Revised Selected and Invited Papers*, volume 3933 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2005.
13. Arno Siebes and Diyah Puspitaningrum. Mining databases to mine queries faster. In Wray L. Buntine, Marko Grobelnik, Dunja Mladenic, and John Shawe-Taylor, editors, *Proceedings ECML PKDD 2009, Part II*, volume 5782 of *Lecture Notes in Computer Science*, pages 382–397. Springer, 2009.
14. Arno Siebes, Jilles Vreeken, and Matthijs van Leeuwen. Item sets that compress. In *Proceedings of the SIAM Conference on Data Mining*, pages 393–404, 2006.
15. Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. Preserving privacy through data generation. In *Proceedings of the IEEE International Conference on Data Mining*, pages 685–690, 2007.