To my Parents

Acknowledgments

The work presented in this thesis would not be accomplished without the support of many people and institutions that I hereby thank.

First of all, I want to thank my supervisors, Professor João Gama, Associated Professor at the Faculty of Economics of the University of Porto (FEP), and Professor Vítor Santos Costa, Associated Professor at the Faculty of Sciences of the University of Porto, for accepting the supervision of this thesis.

Professor João Gama introduce me to this amazing area of machine learning and was the main propeller of the work presented here. Beyond the guidance and technical collaboration, the many informal meetings and discussions we had were a source of motivation and knowledge. Even the discussions about our football (rival) teams contributed to good working environment and to overcome the most difficult phases of this work.

Professor Vítor Santos Costa was my undergraduate teacher and since then I see him as an outstanding human being and researcher that is always involved and available. In this work I confirmed that and gain (I hope!) with his remarkable expertise in Inductive Logic Programming and scientific writing.

I deeply thank Professor Pavel Brazdil and Professor Alípio Jorge, the two coordinators of LIAAD-INESC TEC, in the period that I worked in this thesis, and Professor Fernando Silva, the coordinator of CRACS-INESC TEC. Without their support this work would be much more difficult to finish. A special thank to Professor Pavel Brazdil, he was the undergraduate teacher that introduce me to the Artificial Intelligence field and was the main source of inspiration that made me decide for this area.

I thank all researches and collaborators of LIAAD-INESC TEC for the pleasant discussions and with whom I learn something. To Rui Leite, João Moreira, Luís Torgo, Carlos Soares, Petr Kosina, Raquel Sebastião, Rita Ribeiro, Pedro Rodrigues, Márcia Oliveira, Ezilda Almeida, Luís Matias and Hadi Tork a great thanks. Moreover, I thank all people that present their work in the LIAAD-INESC TEC seminars. Without them I would not have evolved so much.

I also thank the YAP Prolog compiler team for the outstanding support. A special thank to Professor Vítor Santos Costa and Ricardo Rocha.

Thanks to my niece for the smile and the many fun times.

To my parents I do not thank... Agradeço aos meus pais por TUDO!

I thank the Faculty of Sciences of the University of Porto for allowing me to endure the doctoral program. I have been studying in this faculty since the beginning of my undergraduate course and it is a pleasure to be part of this institution.

The work presented in this thesis was partially supported by the *Portuguese Foundation for Science and Technology* (FCT) and the *Polytechnic Institute of Porto* (ISEP/IPP) with the PhD grant SFRH/PROTEC/49634/2009.

Two FCT projects also provide financial support and equipment to the work developed in this thesis, the *Knowledge Discovery from Ubiquitous Data Streams* (PTDC/EIA-EIA/098355/2008) project and the *HORUS: Horn Representations of Uncertainty in Systems* (PTDC/EIA-EIA/100897/2008) project financed many of the work developed in this thesis.

This work was also part-funded by the ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by the Portuguese Funds through the FCT within project FCOMP-01-0124-FEDER-022701 the FEDER funds through COMPETE program.

Thanks!

Abstract

Multi-relational databases are widely used to represent and store data. Often, a multi-relational database is composed by tables recording static data, that do not change over time, and by tables recording dynamic data, that is being accumulated over time. Finding temporal patterns in such temporal databases is an important challenge in domains as diverse as video processing, computational biology and elderly monitoring.

The main goal of this thesis is to study methods and techniques to explore the temporal information available in such multi-relational databases. Mainly, to find rich patterns and learn highly expressive classification theories. Therefore, this thesis is aimed at developing methodologies that can explore both static and dynamic data and the logico-relational information available in rich structured databases. Furthermore, as expressiveness of theories comes at expense of higher computational costs, in this thesis we investigated time and space efficient methodologies.

To achieve these goals we analyze the state-of-the-art methods and present a set of frameworks and algorithms that combine methods and techniques from propositional and relational learning fields. In particular, we explore temporal information using either propositional or first-order logic (FOL) sequence miners. Moreover, we explore propositionalization and predicate invention techniques to learn either propositional or FOL theories.

Our proposals include four frameworks and an ILP based algorithm. We introduce the *propositional MuSer* framework to explore temporal information using well-know and efficient propositional methodologies. At the core of this framework we have a new encoding methodology and a dual search strategy. The encoding methodology translates the temporal information available in a multi-relational database into a propositional sequence database suitable to be explored by a propositional sequence miner. With the dual search strategy frequent sequences are found by mining each dataset partition individually. Moreover, we make use of a new propositionalization

technique to induce a propositional classifier. The second framework, the *XMuSer*, combines propositional and relational techniques to learn a FOL classification theory. In this framework we use the encoding methodology introduced in the *MuSer* system and run a propositional sequence miner that can find a constrained set of temporal patterns. We find a constrained set of patterns and use a predicate invention technique to learn a FOL theory from static and temporal data. The third system that we introduce here does not use any propositional technique. Both the mining of sequence patterns and the theory learning procedures rely on FOL algorithms. The main goal of this framework is to explore the discriminative power of contrasting patterns found by the *BeamSouL* algorithm, also developed in this work. The *BeamSouL* algorithm is an ILP based algorithm that uses SeqLog representation formalism to represent and explore sequences of logical atoms. *BeamSouL* algorithm was designed to find all or a subset of long contrasting patterns. The efficiency of this algorithm relies on the θ-subsumption operator, that eliminates the need to run the sequence miner with low support values, and in a hybrid search strategy that combines exhaustive and beam search methods. By using a beam search procedure in a second phase the algorithm can find long size patterns that would be impossible to find with existing methods.

All these proposals were evaluated empirically by using four real databases. Moreover, we compare our proposals with a set of algorithms. In this set we include the *RUSE-WARMR* framework that we develop under this thesis. The *RUSE-WARMR* system uses arithmetic aggregation operations and a FOL association rule miner to explore temporal data. This way the *RUSE-WARMR* do not conveniently explore temporal data and exhibits the worst results among the evaluated architectures.

Moreover, as all our proposals generate at some point a large number of patterns, we analyze feature selection techniques and introduced a new technique that selects features in a rank order space induced by an ensemble of metrics.

Resumo

Bases de dados relacionais são muito utilizadas para representar e armazenar dados. Geralmente, uma base de dados relacional é composta por um conjunto de tabelas, algumas destas registando informação estática que não muda ao longo do tempo e outras tabelas registando informação dinâmica, que é acumulada ao longo do tempo. Encontrar padrões temporais nestas bases de dados temporais é um desafio importante em domínios tão diversos como o processamento de vídeo, a biologia computacional e a monitorização de idosos.

O principal objetivo desta tese é estudar métodos e técnicas que permitam explorar a informação temporal disponível nestas bases de dados relacionais. Mais precisamente, esta tese visa o desenvolvimento de metodologias que permitam encontrar padrões e aprender teorias de classificação expressivas. Assim, esta tese tem por objetivo o desenvolvimento de métodos que possam explorar tanto a informação estática e dinâmica como a informação lógico relacional disponível em base de dados relacionais. Além disso, como a expressividade das teorias é sempre obtida com elevados custos computacionais, também procuramos desenvolver metodologias eficientes em termos de tempo e espaço de procura.

Para atingir estes objetivos, analisamos o estado-da-arte e apresentamos um conjunto de arquiteturas e algoritmos que combinam métodos e técnicas de aprendizagem proposicional e relacional. Em particular, exploramos informação temporal usando algoritmos proposicionais ou de lógica de primeira ordem (LOP) que descobrem padrões sequenciais. Além disso, exploramos técnicas de proposicionalização e de criação de predicados para aprender teorias proposicionais ou de LPO.

Assim, nesta tese desenvolvemos quatro novas arquiteturas e um algoritmo baseado em ILP. A arquitetura proposicional *Muser* explora informação temporal usando algoritmos proposicionais, que são conhecidos pela sua eficiência. Fundamental para o desempenho desta metodologia é a técnica de codificação e a estratégia de pesquisa

dual. A técnica de codificação transforma a informação temporal existente na base de dados para uma outra base de dados contendo sequências proposicionais, sendo que esta última é adequada para ser usada como entrada para um algoritmo proposicional de descoberta de padrões sequenciais. Com a estratégia de pesquisa dual os padrões sequenciais são encontrados explorando cada uma das partições individualmente. Nesta arquitetura também fazemos uso da técnica proposicionalização de modo a incluir os padrões sequenciais na fase de aprendizagem de uma teoria proposicional. A segunda arquitetura desenvolvida é denominada *XMuSer* e combina algoritmos proposicionais de descoberta de padrões sequenciais e algoritmos baseados em ILP que aprendem uma teoria usando LPO. Nesta metodologia utilizamos a técnica de codificação introduzida na arquitetura *MuSer* e um algoritmo de descoberta de padrões sequenciais que encontra um conjunto restrito de padrões. Depois de encontrar um conjunto limitado de padrões, usamos a técnica de invenção de predicados para aprender uma teoria LPO que pode incluir informação estática e/ou temporal. A terceira metodologia que desenvolvemos não usa qualquer técnica proposicional. Tanto a descoberta de padrões sequenciais como a fase de aprendizagem usam a LPO como formalismo de representação. O desenvolvimento desta metodologia tem por objetivo explorar o poder de discriminativo dos padrões contrastantes encontrados pelo algoritmo *BeamSouL*, também desenvolvido neste trabalho. O algoritmo *BeamSouL* é baseado em ILP e utiliza o formalismo SeqLog para representar e descobrir padrões sequenciais de átomos lógicos. O algoritmo *BeamSouL* foi concebido para encontrar todos ou um subconjunto de padrões contrastantes de grande comprimento. A eficiência deste algoritmo baseia-se num procedimento, que usa a subordinação-θ, que elimina a necessidade de executar o algoritmo de descoberta de padrões sequenciais usando um valor de suporte baixo e também no uso de uma estratégia de pesquisa híbrida que combina pesquisa exaustiva com pesquisa de feixe. Ao usar a pesquisa de feixe na segunda fase, o algoritmo consegue encontrar padrões de grande comprimento que seriam impossíveis de encontrar recorrendo aos algoritmos existentes.

Todas estas propostas foram avaliadas de forma empírica e utilizando quatro conjuntos de dados reais. Na avaliação das nossas propostas consideramos um conjunto de algoritmos concorrentes. Neste conjunto incluímos a arquitetura *RUSE-WARMR* que também desenvolvemos nesta tese. Esta arquitetura usa técnicas de agregação dos dados temporais e um algoritmo que encontra regras de associação representadas em LPO. Com esta estratégia, o *RUSE-WARMR* não explora convenientemente dados temporais e apresenta piores resultados do que qualquer das metodologias especialmente desenvolvidas para explorar a informação temporal.

Além disso, e como todos os algoritmos de descoberta de padrões sequenciais geram um grande número de padrões, analisamos um conjunto de técnicas de seleção de atributos e introduzimos uma nova técnica que seleciona atributos num espaço cuja ordenação é induzido por um conjunto de métricas.

Contents

Abstract **VII**

Resumo **IX**

List of Tables **XXI**

List of Figures **XXIV**

1 Introduction **1**

1.1 Motivation 2

1.2 Main Contributions 2

1.3 Outline of the Thesis 4

2 Multi-Relational Data Mining **7**

2.1 Introduction 7

2.2 Relational Databases 9

2.3 Inductive Logic Programming 10

2.3.1 Clausal Theory 11

2.3.1.1 Syntax of Logical Clauses 11

2.3.1.2 Semantic of Logical Clauses 12

2.3.2 Learning Tasks 13

2.3.3 Searching for a Clausal Theory 14
2.3.3.1 Structuring the Search Space 15
2.3.3.2 Searching the Hypothesis Space 15
2.4 Propositionalization 18
2.4.1 Database-oriented Approach 19
2.4.2 Logic-oriented Approach 20
2.4.3 Propositional Classifiers 23
2.5 Learning Issues 24
2.5.1 Extending Language Bias 24
2.5.1.1 Constructive Induction 25
2.5.1.2 Predicate Invention 26
2.5.2 Feature Selection 27
2.5.3 Classifier Evaluation 32
2.5.3.1 Evaluation Criteria 32
2.5.3.2 Estimating the Performance of a Predictive Model . . 34
2.5.3.3 Learning and Evaluation of Multi-class Classifiers . . . 36
2.6 Pattern Mining: Itemsets and Sequences 37
2.6.1 Frequent Itemset Discovery 38
2.6.1.1 Finding Propositional Patterns 38
2.6.1.2 Finding First-order Logic Patterns 42
2.6.2 Frequent Sequence Discovery 42
2.6.2.1 Finding Propositional Patterns 42
2.6.2.2 Finding First-order Logic Patterns 45

3 Constructive Induction with Ensemble Feature Selection 51

3.1 Introduction 51

3.2 The rankCI Framework 52
3.2.1 Constructive Induction - Phase 1 53
3.2.2 Feature Selection - Phase 2 54
3.2.3 Decision Tree Induction - Phase 3 54
3.2.4 Illustrative Example 55
3.3 Experimental Evaluation 57
3.3.1 Configuration of the Experiments 57
3.3.1.1 Datasets and Development Environment 57
3.3.1.2 Organization of the Experiments 58
3.3.2 Results 60
3.3.3 Analysis of the Results 62
3.4 Conclusions 63

4 Propositionalization with First-order Logic Features 65

4.1 Introduction 65
4.2 Preliminaries 67
4.3 The RUSE-WARMR Framework 68
4.3.1 Finding First Order Descriptors - Phase 1 69
4.3.2 Feature Selection and Propositionalization - Phase 2 69
4.3.3 Learning a Classification Model - Phase 3 70
4.3.4 Illustrative Example 70
4.4 Experimental Evaluation 72
4.4.1 Configuration of the Experiments 75
4.4.2 Results 76
4.4.3 Analysis of the Results 77
4.5 Conclusions 79

5 Exploring Multi-Relational Temporal Data using Propositional Techniques 81

5.1 Introduction . . . 81

5.2 The *MuSer* Framework . . . 83

5.2.1 Encoding Methodology - Phase 0 . . . 84

5.2.2 Finding Frequent Sequence Patterns - Phase 1 . . . 86

5.2.3 Feature Selection - Phase 2 . . . 86

5.2.4 Mapping Features - Phase 3 . . . 87

5.2.5 Classifier Induction - Phase 4 . . . 87

5.3 Experimental Evaluation . . . 88

5.3.1 Configuration of the Experiments . . . 88

5.3.2 Results . . . 89

5.3.3 Analysis of the Results . . . 92

5.4 Conclusions . . . 94

6 Discriminative Sequence Miner in Multi-Relational Learning 97

6.1 Introduction . . . 97

6.2 Preliminaries . . . 99

6.3 The XMuSer framework . . . 102

6.3.1 Data Encoding - Phase 0 . . . 102

6.3.2 Finding Frequent Sequential Patterns - Phase 1 . . . 103

6.3.3 Sort Sequential Patterns - Phase 2 . . . 105

6.3.4 Mapping Back Interesting Sequences - Phase 3 . . . 105

6.3.5 Learning a Theory - Phase 4 . . . 106

6.4 Experimental evaluation . . . 107

6.4.1 Configuration of the Experiments . . . 107

6.4.2 Results . . . 108
6.4.3 Analysis of the Results . . . 111
6.5 Conclusions and Future Work . . . 113

7 Contrasting Logical Sequences in Multi-Relational Learning 115

7.1 Introduction . . . 115
7.2 The BeamSouL Algorithm . . . 117
7.2.1 BeamSouL: A FOL Sequence Miner . . . 117
7.2.1.1 Initialize Parameters and Find Patterns of Size One . . 117
7.2.1.2 Hybrid Search and Candidate Generation-and-Test Strategies . . . 119
7.2.1.3 Return Contrasting Patterns . . . 119
7.2.2 Contrasting Patterns in Relational Learning . . . 120
7.2.2.1 Find Contrasting Patterns with *BeamSouL* . . . 120
7.2.2.2 Select Interesting Patterns using the Confidence Metric 121
7.2.2.3 Mapping into a New Predicate . . . 121
7.2.2.4 Learn a Theory . . . 121
7.2.3 Illustrative Example . . . 122
7.3 Experimental Evaluation . . . 125
7.3.1 Configuration of the Experiments . . . 125
7.3.2 Results . . . 128
7.3.3 Analysis of the Results . . . 132
7.4 Conclusions . . . 135

8 Conclusions 137

8.1 Summary . . . 137
8.2 Findings . . . 138

8.3 Limitations 141
8.4 Future Work 141

References **143**

Appendix A DNF Formulae **155**

Appendix B Datasets **157**

B.1 Hepatitis 157
B.2 Financial 158
B.3 FinancialB 158
B.4 Protein 159

Appendix C RUSE-WARMR Results **161**

Appendix D MuSer Results **165**

Appendix E XMuSer Results **169**

Appendix F ConFOLding Results **175**

List of Tables

2.1 Relational database. Illustrative example inspired in the PKDD 2005 Hepatitis dataset 8

2.2 Contingency table 28

2.3 Selection metrics for association rules 29

3.1 Four instances of the *yellow-small+adult-stretch* dataset 55

3.2 Total order over the itemsets using the value of each heuristic 55

3.3 Database enlarged with ten new attributes 56

3.4 Description of the datasets and the number of itemsets found by the Apriori algorithm and selected using the *xor-filter* 58

3.5 Misclassification error using the six heuristics *ensemble* 59

3.6 Theory size: number of tree nodes using the six heuristics *ensemble* . . 60

3.7 Misclassification error using a single heuristic 61

3.8 Theory size: number of tree nodes using a single heuristic 62

4.1 Enlarged target table 72

4.2 Number of patterns found in different phases of the *RUSE-WARMR* framework 73

4.3 Generalization accuracy of the *RUSE-WARMR* framework 74

5.1 Relational database used to illustrate the encoding technique 84

5.2 Sequence Database 85

5.3 Enlarged target table 87

5.4 Number of patterns found in each step of the *MuSer* framework 91

5.5 Generalization accuracy of the *MuSer* framework 92

6.1 Sequence Database 103

6.2 Predictive sequential patterns found 104

6.3 Ordered list of sequential patterns 104

6.4 Sequence Relation 105

6.5 Number of patterns found in each step of the XMuSer framework . . . 109

6.6 Generalization accuracy of the XMuSer framework 110

7.1 Relational database used to clarify some details of the *BeamSouL* algorithm and of the *ConFOLding* framework 122

7.2 Number of patterns found in each run of the *BeamSouL* algorithm . . . 129

7.3 Generalization accuracy of the *ConFOLding* framework 130

B.1 Number of relations and class distribution of each database 159

C.1 Peak memory usage of the *RUSE-WARMR* framework (in kilobytes) . . 162

C.2 Run-time of the *RUSE-WARMR* framework (in seconds) 163

C.3 Generalization accuracy of the *RUSE-WARMR* framework selecting the top ten and the top twenty most interesting patterns 164

D.1 Peak memory usage in each step of the *MuSer* framework (in kilobytes) 166

D.2 Run-time of each step of the *MuSer* framework (in seconds) 167

E.1 Peak memory usage in each step of the *XMuSer* framework (in kilobytes) 170

E.2 Run-time of each step of the *XMuSer* framework (in seconds) 171

E.3 Generalization accuracy of the *XMuSer* framework using a propositional learner in the last step 172

E.4 Theory size: number of nodes 173

F.1 Peak memory usage of the *ConFOLding* framework (in kilobytes) . . . 176

F.2 Run-time of the *ConFOLding* framework (in seconds) 179

F.3 Theory size: number of rules 182

F.4 Generalization accuracy of the propositional *ConFOLding* framework . 183

F.5 Theory size: number of tree nodes 184

List of Figures

2.1 Partial order over the itemsets 40

3.1 *Rank ensemble* selection 56

3.2 Stand-alone C4.5 56

4.1 *RUSE-WARMR* induced decision tree for the Hepatitis subtype problem 71

5.1 Patient one event sequence 85

5.2 *MuSer*'s best induced decision tree for the Hepatitis subtype problem . 90

5.3 Example of a decision tree induced when the support value of *MuSer*'s sequence miner is set to 90% 93

7.1 Number of patterns found by the *BeamSouL* algorithm in the majority partition of the Hepatitis dataset. We present the number of frequent, closed and maximal patterns found for a range of maximum refinement levels 127

7.2 Run-time needed by the *BeamSouL* and the MineSeqLog algorithms to find patterns in the majority partition of the Hepatitis dataset. We present the run-time of these algorithms for a range of maximum refinement levels 128

7.3 Number of patterns found by the *BeamSouL* algorithm in the majority partition of the Hepatitis dataset. We present the number of frequent, closed and maximal patterns found for a range of beam sizes 132

7.4 Run-time needed by the *BeamSouL* algorithm to find patterns in the majority class of the Hepatitis dataset. We present the run-time of the *BeamSouL* algorithm for different beam sizes 133

F.1 Run-time needed by the *BeamSouL* and the MineSeqLog algorithms to find patterns in minority partition of the Hepatitis dataset. We present the run-time of these algorithms for different refinement levels 177

F.2 Number of patterns found by the *BeamSouL* algorithm in the minority partition of the Hepatitis dataset. We present the number of frequent, closed and maximal patterns found for a range of beam sizes 178

F.3 Run-time needed by the *BeamSouL* algorithm to find patterns in the Hepatitis dataset. We present the run-time of the *BeamSouL* algorithm for different beam sizes and for the negative-run of the *ConFOLding* framework 180

F.4 Run-time needed by the *BeamSouL* algorithm to find patterns in the Hepatitis dataset. We present the run-time of the *BeamSouL* algorithm for different beam sizes and for the positive-run of the *ConFOLding* framework 181

Chapter 1

Introduction

Multi-relational databases are widely used to represent and store data. Often, a multi-relational database is composed by a set of tables, some recording *static* characteristics that do not change over time and other tables recording evolving, *dynamic*, data that is accumulated over time. For instance, in the medical domain a set of tables can record static information about the patient, like the patient's date-of-birth and name, and other tables may record an always increasing number of sequences of medical examinations, such as blood analysis or urinalysis. Finding sequence patterns in such temporal databases is an important challenge in domains as diverse as video processing, computational biology and elderly monitoring.

The classical approach to explore such data is to join tables and then run a propositional miner to find temporal patterns [KLF00, KRZ+03b]. The main issue with joining before mining is the large size of the generated table and the loss of valuable logico-relational information. An alternative approach to *join before mining* is to *mine before join*. Among the methodologies used in this last approach is Inductive Logic Programming (ILP) [MR94, Rae08]. Moreover, most of the existing techniques, including ILP techniques, fail to correctly explore the temporal information. Typically temporal information is aggregated [DT99] and hence important logic and temporal information is lost.

In this work we study and introduce algorithms and techniques to explore temporal information available in multi-relational databases. Namely, we explore temporal information using both propositional [AS95, Zak01, PHMA+01] and first-order logic (FOL) sequence miners [LR04, dAF07]. We use these techniques to develop novel methodologies for supervised learning.

1.1 Motivation

The methodologies available to explore temporal databases are based on the exclusive use of propositional or multi-relational techniques [LR04]. The former set of methods are widely available and are well-known to be time efficient. The main issue with these propositional methods is that they are unable to explore valuable relational information recorded in multi-relational databases. For instance, if we join the tables and then use a sequence miner to find frequent sequences we are unable to explore conveniently static information or include other valuable domain knowledge. In contrast to this, the relational techniques, especially ILP techniques, use high order representational formalisms to learn and represent a theory but pay a high cost in terms of scalability and time efficiency [BS03]. For instance, ILP based methods fail to find long patterns available in multi-relational databases that record dynamic information that is accumulated over a long period of time. Moreover, it is well-known that pattern mining methodologies, both propositional and relational, find a large set of patterns [PHMA+01], and most of them are redundant and uninteresting. Typically, the number of findings is so large that it is almost impossible for a classification algorithm or a human to select the most interesting patterns [YL04, YL03].

Thus, the goal of this thesis is to study methodologies that can benefit from the strengths of propositional and relational methodologies and eliminate, or reduce, the weaknesses of the two approaches. We should answer the following research questions:

RQ1: How can we find temporal patterns available in multi-relational databases that accumulate data over a long period of time?

RQ2: How do we combine propositional sequence miners with ILP based techniques to gain with the efficiency of propositional miners and representational power of first-order logic formalism?

RQ3: Can we develop feature selection techniques that select the most discriminative features found by a pattern mining algorithm?

1.2 Main Contributions

The research questions presented in the previous section can be answered with a wide range of methodologies. In this thesis we propose a set of frameworks and algorithms

that explore temporal data using either propositional or relational sequence miners. Moreover, the methodologies are targeted to solve classification problems.

Thus, the main contributions of this work are:

C1: Develop the *BeamSouL* algorithm [FGC14] that can be tuned to find all or a subset of the long and contrasting FOL sequences. This algorithm is based in the SeqLog representation formalism and uses a hybrid search strategy to explore the lattice induced by the subsumption operator. In the first phase the algorithm runs an exhaustive search to find all *short* patterns. In the second phase the algorithm uses a beam search strategy and the confidence metric to find interesting long sequences. Moreover, by using this algorithm we can find contrasting patterns available in the database without running the sequence miner to find low support patterns in any partition.

C2: Develop the *ConFOLding* framework [FGC14] that both explores temporal patterns and learns a classification theory using first-order logic methods. This architecture uses predicate invention technique to include the most interesting FOL contrasting patterns in the ILP learning task, where a FOL classification theory is learned.

C3: Develop the *MuSer* framework [FGC10] that explores temporal databases using a set of propositional methodologies. This architecture uses an encoding strategy and a classical propositional sequence miner to find temporal patterns. Then, two filters are used to select the most interesting patterns. A propositional classifier is used to learn a propositional classification model.

C4: Introduce a strategy that converts the temporal data available in a multi-relational database into a sequence database [FGC10, FGC11, FGC12a]. The obtained database can contain sequence data from different source tables and is suitable to be explored by propositional sequence miners.

C5: A strategy that finds discriminative patterns by mining each class partition individually [FGC10, FGC11, FGC12a, FGC14]. By using this strategy we can find a large number of patterns that can only be found if we run the sequence miner with low support values on the full dataset. With our strategy we can run different instances of a sequence miner in parallel, one instance to explore each dataset partition, using approximately the same running time and equal or less memory usage as if we run a sequence miner on the full dataset. Moreover, with our strategy we can find a larger set of patterns than we can find by mining the

full dataset using the same support value. To find the same set of patterns using the classical approach we need to use a small support threshold value.

C6: Develop *XMuSer* architecture [FGC12a], that explores temporal databases by combining a propositional discriminative sequence miner with an ILP learner. By using this framework we can explore and include both non-temporal, temporal, frequent and infrequent data in a FOL classification model. Moreover, we include the feature selection task in the discriminative sequence miner and do not need to use the post-processing feature selection technique introduced in *MuSer*. In the last step we use predicate invention technique and an ILP algorithm to learn the FOL classification model.

C7: A feature selection filter [FG07] that selects features in a rank-order space induced by a set of heterogeneous heuristics.

1.3 Outline of the Thesis

The structure of this thesis has five main parts. First, in part one we give an overview of the work developed in this thesis and present fundamental concepts and techniques that are needed to develop and understand the work presented in parts two and three of this document. Second, in part two we study a set of methods and techniques to explore static and relational information recorded in a single table or multi-relational databases. Next, in part three we develop propositional and relational methodologies to explore temporal information available in multi-relational databases. In part four we conclude and present future research directions. In part five we present three appendices that present a detailed description of the results summarized in part three.

Part I - Concepts and Techniques
In Chapter 2 we introduce fundamental concepts and algorithms that are needed to develop and understand this thesis. We introduce concepts such as Classification, Multi-relational Data Mining, Pattern Mining and ILP.

Part II - Exploring Static Data
In Chapter 3 we present preliminary work that we develop to explore feature selection and constructive induction techniques. We develop the *rankCI* framework to explore

data registered in a single table. In this architecture we find patterns using a propositional association rule miner and, then, use an ensemble of metrics to rank and sort the findings. In the last phase, the most interesting features are projected and used to learn a highly descriptive propositional classification model. Chapter 4 presents the *RUSE-WARMR* framework. In this system we search for patterns available in multi-relational databases using the WARMR FOL association rule miner [DT99]. Moreover, we use the propositionalizon technique and a propositional classifier to learn a classification theory. In this chapter we use temporal databases to evaluate the *RUSE-WARMR* architecture but we use aggregation techniques to convert temporal information into static information.

Part III - Exploring Static and Temporal Relations

In Chapter 5 we explore temporal data available in multi-relational databases using a propositional sequence miner. We introduce the *MuSer* framework that, first, selects the most interesting patterns using two selection filters. Then, a propositionalization technique is used to map the patterns into the target table. In Chapter 7 we update and extend the *MuSer* architecture. We introduce the *XMuSer* framework that runs cSpade propositional sequence miner [Zak00] to find a constrained subset of sequence patterns. By using the cSpade algorithm we eliminate the filter selection procedure introduced in *MuSer* framework. Moreover, we use predicate invention technique to represent the information available in the discriminative patterns. Furthermore, we use an ILP algorithm to learn a FOL theory. In Chapter 7 we present the *BeamSouL* algorithm to find FOL contrasting sequences available in temporal databases. This algorithm uses a hybrid search strategy to find all or a subset of long sequences. Moreover, we introduce a FOL architecture that gains with the strengths of the *BeamSouL* algorithm.

Part IV - Conclusions

In Chapter 8 we analyze the work developed in this thesis. We summarize the main findings, present the limitations of this work and present future research directions.

Part V - Addenda

In the appendices that appear at the end of this thesis we present a detailed description of the datasets used to evaluate the frameworks and algorithms introduced in Part II and Part III. Moreover, we present additional results that we obtain when we experimentally evaluate the methodologies developed in this thesis.

In the next chapter we introduce multi-relational data mining and overview the state-of-the-art in mining multi-relational temporal patterns.

Chapter 2

Multi-Relational Data Mining

In this chapter we briefly introduce some key concepts that were needed to develop the work presented in this thesis. We present concepts such as Multi-Relational Data Mining (MRDM), Inductive Logic Programming (ILP), Propositionalization and Pattern Mining. Moreover, we introduce some techniques and methodologies suitable to explore information stored in multi-relational databases.

2.1 Introduction

Data mining can be stated as the process of extracting knowledge from large amounts of raw data [HKP11]. Most of the existing work put the focus on developing methodologies to explore data stored in a single table. Such methodologies, known as propositional or attribute-value learning approaches, search for patterns that can be valuable in data mining tasks like classification or clustering [Dŏ3]. These methodologies explore propositional data and are known to efficiently generate effective models like association rules or decision trees [FG07]. The main issue with these propositional representation and propositional learners is that the majority of the relational real databases of interest have multiple relations and the most interesting patterns can involve several relations.

A relational database typically consists of several tables registering logic related information. The relations in the database can be defined extensional or intensionally. In an extensional relation each tuple is defined explicitly using values from the domain of each attribute. In Table 2.1 we present an example database with three relations. This example database is inspired in the PKDD 2005 Hepatitis dataset and registers

Table 2.1: Relational database. Illustrative example inspired in the PKDD 2005 Hepatitis dataset

Patient Info
(Target Table)

Id	*Sex*	*BornDate*	*Class*
1	m	19520109	*c*
2	f	19750123	*b*

Blood Analysis

Id	*Date*	*RBC*	*WBC*
1	19750102	7.2	5.1
1	19780203	6.5	10.0
2	19770107	6.9	3.1

Urinalysis

Id	*Date*	*Exam*	*Value*
1	19741201	plt	400
1	19750102	alb	4.2
1	19750102	ttp	7.3
1	19760204	alb	7.9
2	19800403	alb	2.3

information about a set of medical examinations taken by patients with hepatitis subtype *b* and *c* (see Appendix B for additional information). Defining an intensional relation means that we are using information from inter-related relations to define a new relation. These intensional relations are also known as views and usually are build to represent domain or background knowledge about the domain of discourse [Dŏ3]. Multi-relational data mining is the process of extracting knowledge from such multi-relational databases.

To explore spatio-temporal information stored in multi-relational databases two main strategies can be used. One strategy, known as propositionalization [KLF00, KRZ+03a, KRH01], requires a pre-processing stage where the data is converted into a single table. Typically, database joins or aggregations operations are used [Kno05] to transform data. Then, a propositional algorithm is used to learn models from the single table. The other strategy is to use a relational algorithm that uses the structure of the database and a relational language to explore the richness of the multi-relational data representation. Most of these relational algorithms came from the field of logic programming, more precisely, from the field of Inductive Logic Programming (ILP). Moreover, a wide number of propositional algorithms that are known by their efficiency have been extended to deal with relational data [LR00, FL04].

A set of techniques were developed to include domain knowledge, both non-observable and observable, in the MRDM process. [Mic93] distinguishes between selective induction and constructive induction saying:

Empirical learning uses little domain knowledge, while constructive induction uses more domain knowledge. A more precise way to characterize this distinction is that in empirical induction the description space for examples and for the hypotheses is the same, while in constructive induction these spaces are different

For instance, using ideas from propositional constructive induction [Mic93] and first-order logic (FOL) features, that can be extracted in a pattern mining process, Predicate Invention [Kra95] technique was developed. Using this technique, that can be combined with almost any relational algorithm, in a relational data mining process proved to generate high accurate and less complex relational models [Kra95, BM98].

In the following sections we introduce some of the above mentioned fundamental concepts. Firstly, we introduce relational databases. In Section 2.3 we introduce ILP. Then, in Section 2.4, we overview the Propositionalization technique. Next, in Section 2.5, we discuss some learning issues. Last, in Section 2.6, we present and analyze pattern mining algorithms that can find frequent itemsets and sequences. We discuss algorithms that can find propositional and FOL patterns.

2.2 Relational Databases

Most of today's structured data is stored in relational databases. A relational database [GMUW09] is a set of relations/tables $T_1, T_2, \ldots, T_n$ and a set of relationships $R_1, R_2, \ldots, R_m$. Each relation/table T_i in the database is composed by a set of records (also known as tuples) and each record in the table is characterized by a fixed set of fields (also known as attributes) $A_1, A_2, \ldots, A_k$. Each tuple in a relation T_i is an ordered list of values $t =< v_1, v_2, \ldots, v_n >$, where each $v_i \in Dom(A_i)$ or equals the Null value. Each attribute A_i has a domain $Dom(A_i)$ consisting of values that the attribute can take.

To model the relationships between relations in a database we introduce the concepts of primary and foreign key and describe three types of relationships: one-to-one, one-to-many or many-to-many.

Definition 1. *The* primary key *of a relational table, denoted as $T_i.A^{PK}$, is a combination of attributes or an attribute A_k(not null) that uniquely identifies each record in the table T_i.*

Definition 2. *The* foreign key *of a table T_i, denoted $T_i.A^{FK}$, is an attribute A_m(not*

null) that refers to a primary key of another table, $T_j.A_k$. Moreover, the $Dom(T_i.A_m) \subseteq Dom(T_j.A_k)$. This key ensures that the data stored in a database is consistent.

Definition 3. *A table T_i is said to have* one-to-one relationship *with table T_j, if for every tuple in table T_i we have at most one associated tuple in table T_j.*

Definition 4. *A table T_i is said to have a* one-to-many relationship *with table T_j, if for every tuple in T_i we have zero, one or more tuples associated with table T_j. And, for each tuple in table T_j we have an associated tuple in table T_i.*

Definition 5. *A* many-to-many relationship *between two tables T_i and T_j exists when for each tuple in table T_i there exist many tuples associated in table T_j, and vice versa.*

In a multi-relational classification task it is required to have a table labeling the individuals registered in the database.

Definition 6. *A* target table *is a table that consists of labeled n-ary tuples describing a unique individual/example. Each target tuple is associated with a class label. The individuals registered in this type of tables are characterized by non-target records related with the target table.*

Additionally, we can use the *relational algebra* formal system to manipulate relations. Among the operators available in relational algebra we highlight the natural join operation.

Definition 7. *The* natural join *of two database relations R_1 and R_2, denoted $R_1 \bowtie R_2$, is the set of all combinations of tuples in R_1 and R_2 having a common subset of attributes and matching attribute values.*

2.3 Inductive Logic Programming

Inductive Logic Programming is a research area that studies tools and techniques to learn FOL clausal theories from examples and background knowledge [MR94, Dž03, LDN99]. Different from logical program systems that use deductive inference, ILP systems use inductive inference to explore relational data. In this section we introduce the clausal theory, the ILP predictive setting and some ILP algorithms that can find a predictive theory.

2.3.1 Clausal Theory

In this section we introduce some syntactic and semantic logical concepts that will be useful along this thesis.

2.3.1.1 Syntax of Logical Clauses

ILP uses the representational formalism of Logic Programming to learn a theory that consists of a set of FOL clauses. Each clause in the learning theory as a conclusion part and a condition part, and can be defined in the following way:

Definition 8. *A clause is an expression* $p_1 \vee \ldots \vee p_n \leftarrow q_1 \wedge \ldots \wedge q_m$*, where the* $p_i s$ *and the* $q_j s$ *are FOL atoms. The* $p_1 \vee \ldots \vee p_n$ *disjunction is known as the head of the clause, or the conclusion part, and* $q_1 \wedge \ldots \wedge q_m$ *conjunction is known as the body of the clause, or the condition part. Moreover, the variables appearing in each logical atom are universally quantified.*

A FOL *atom* is a formula of the form $\mathrm{p}(t_1, \ldots, t_n)$, of arity n, where each t_i is a term. A logical fact is a logical atom having no variables. A *term* t_i can be a constant, a variable or a k-ary functor f applied to k terms.

The clause $p_1 \vee \ldots \vee p_n \leftarrow q_1 \wedge \ldots \wedge q_m$ can also be represented as a disjunction of literals $p_1 \vee \ldots \vee p_n \vee \neg q_1 \vee \ldots \vee \neg q_m$, where a *literal* is a logical atom or its negation.

Most of the existing ILP algorithms learn a theory that consists of a set of *Horn clauses*, i.e., clauses whose conclusion part has at most one positive literal. The clause

$$patient_Info(A, B, C, D) \leftarrow blood_Analysis(E, F, low, H), urinanalysis(A, J, plt, high)$$

is an example of a Horn clause. In this clause $patient_Info(A, B, C, D)$ is the head of the clause and $blood_Analysis(E, F, low, H), urinanalysis(A, J, plt, high)$ is the body of the clause. This clause was found by an ILP system that learns a theory from the relational data presented in Table 2.1.

A k-ary predicate in logic programming corresponds to a k-tuple relation in a relational database. The main difference is that in relational databases the attributes are associated with a domain whereas in Logic Programming they are not.

Moreover, in an attempt to apply principles of logic programming to explore relational databases the Datalog representation formalism was invented. *Datalog* is a function-free subset of the *Prolog* logic language that is related with the relational algebra.

Any non-recursive Datalog logic program can be translated into database relational algebra [GMUW09].

2.3.1.2 Semantic of Logical Clauses

Definition 9. *A* substitution θ *is a finite set* $\{X_1/t_1, X_2/t_2, \ldots, X_n/t_n\}$ *where the* X_i *are variables and the* t_j *are terms. Each pair* X_i/t_j *represents a replacement of variable* X_i *by the term* t_j*. Moreover, we say that the pair* X_i/t_j *is a binding for* X_i*.*

For instance, if we consider the clause

$$c : patient_Info(A, B, C, D) \leftarrow urinanalysis(A, E, F, G)$$

and define the substitution $\theta = \{C/m, F/plt, G/high\}$, the instantiated formula $c\theta$ is the formula $patient_Info(A, B, m, D) \leftarrow urinanalysis(A, E, plt, high)$. This formula was obtained by replacing the variables C, F and G by the constants m, plt and $high$, respectively.

Definition 10. *A* Herbrand interpretation *of a set of clauses,* C*, is the set of ground atoms occurring in* C*.*

All ground atoms appearing in a Herbrand interpretation are assumed to be true. All other ground atoms, that do not appear in the Herbrand interpretation, are assumed to be false.

Definition 11. *The* Herbrand model *of a set of clauses* C *is a Herbrand interpretation in which all clauses in* C *are true.*

Considering the Herbrand interpretation I and a clause $h \leftarrow b$, we say that I is a Herbrand model for clause $h \leftarrow b$ if all substitutions θ that make $b\theta$ true in I, make $h\theta$ true in I.

Moreover, we can define a Herbrand model for a set of clauses C as a Herbrand interpretation that contains all consequences of the set of clauses C. Using this remark we define the notion of Least Herbrand model.

Definition 12. *The* Least Herbrand model *of a set of clauses,* C*, is the minimum set of all ground atoms that are a consequence of a set of clauses* C*.*

In the following definition we present a formal definition of logical consequence of a clause set C. We introduce the notion of logical entailment.

Definition 13. *Let C be a set of clauses and c be a clause. We say that C (semantically) entails c, and write $C \models c$, if every Herbrand model of C is a Herbrand model of c.*

In this definition we use Herbrand models to define entailment. This is not a requirement and entailment can be proved with any model. We use Herbrand models as they play a central role in ILP and because of theorem 1 [NCdW96].

Theorem 1. *Let H be a set of clauses. Then H has a model if and only if H has a Herbrand model.*

In practical applications where we want to prove that a ground atom, g, is entailed by a set of clauses C, the semantic entailment $C \models g$ is replaced by *resolution* deductive inference rule $C \vdash_{res} g$. In the majority of the cases, SLD-resolution [MR94] is used.

Given a clause set C and a clause c, we say that a *Prolog* system is sound if c is (Prolog) provable from C implies that c is semantically entailed by C. Formally we write:

$$C \vdash_{res} c \Rightarrow C \models c.$$

2.3.2 Learning Tasks

In general, the set of tasks that ILP can perform is formally defined [LDN99]:

Given:

- a set of examples E
- a background theory B
- a language bias L that define the clauses allowed in the hypothesis
- an explanation criterion

Find:

- a hypothesis $H \subset L$ which explains the examples E with respect to the theory B

Here we put the focus on ILP systems that find classification and descriptive logic rules. We restrict the presentation to these two tasks since these are the tasks that

we address in this thesis and are among the most popular tasks addressed by ILP systems. There are systems such as FORS [KB97] that can be used to solve regression problems and Cola [Emd94] that can be used to solve clustering problems.

In an ILP classification problem the goal is to learn a set of definite classification rules. Given: a set of positive examples E^+ and negative examples E^-, i.e, examples that belong to the target table; a set of definite clauses B representing background relations and domain knowledge; a definite hypothesis language L specifying syntactical and semantic restrictions. Find a classification theory that explains all examples in the target table and is complete and consistent.

In this setting, the notion of explanation used is coverage. Given the example set $E = E^+ \cup E^-$, the background knowledge B and a hypothesis H, we say that an example $e \in E$ is *covered* by H if $B \cup H \models e$. A hypothesis H is complete if $\forall e \in E^+ : \ H \cup B \models e$. A hypothesis H is consistent if $\forall e \in E^- : \ H \cup B \not\models e$. This is equivalent to finding a theory H where all examples in E^+ are true in the least Herbrand model of $B \cup H$ and all examples in E^- are false.

In descriptive tasks the goal is to find a set of general rules/patterns available in the data. Typically, we consider that the set of examples is restricted to positive examples, E^+, the background knowledge B is restricted to definite clauses and the hypothesis language, L, is a set of general clauses. The goal is to find a hypothesis H such that each clause $c \in H$ is true in the least Herbrand model of $B \cup E$.

In practical applications the data is noisy and it is usual to relax these notions. Typically, we relax the coverage criterion and learn incomplete and inconsistent theories, i.e., theories that do not cover some positive examples or/and cover some negative examples.

Foil [QCJ95], Golem [MF90], Progol [Mug95] and Aleph [Sri03] are examples of predictive ILP systems that find FOL classification rules. Claudien [RD97], Warmr [DT99] and FARMER [NK01] are examples of descriptive ILP systems that find FOL association rules.

2.3.3 Searching for a Clausal Theory

Any ILP task can be seen as a search problem. In the following two sections we introduce an order relation that structures the search space and present strategies to explore the search space.

2.3.3.1 Structuring the Search Space

To explore the space of clauses determined by the language bias, L, we need to introduce an ordering in the ILP search space. One natural way to introduce ordering was to use semantic entailment. The problem with this approach is that semantic entailment is undecidable. In [Plo70] Plotkin introduced the syntactic relation $\theta-$*subsumption*, a *more general than* relation, that is used by the majority of the existing ILP systems.

Definition 14. *Let c_1 and c_2 be two clauses. If there exists a substitution θ such that $c_1\theta \subset c_2$, we say that c_1* θ-subsumes *c_2.*

When we say that a clause c_1 *θ-subsumes* c_2, we are saying that c_1 is more general than c_2. If this is the case we write $c_1 \preceq c_2$.

One important property of θ-subsumption is that given two clauses c_1 and c_2, if $c_1 \preceq c_2$ then $c_1 \models c_2$.

Moreover, Plotkin proved in [Plo70] that this relation induces a lattice on the set of all clauses, up to equivalent classes and variable renaming. This guarantees that any two clauses have a least upper bound and a greatest lower bound that are unique up to renaming of variables. The clause that represents the least upper bound of two clauses is also known as the least general generalization (lgg). This lgg is a key clause that is explored for some of the most efficient ILP algorithms [Sri03].

Furthermore, we can use $\theta-$subsumption to prune and bound the search.

2.3.3.2 Searching the Hypothesis Space

To explore the lattice space induced by θ-subsumption on the space of clauses, three main strategies are used by the existing algorithms: top-down search, bottom-up search, and a hybrid strategy. Here we present an overview of each search strategy. Moreover, we present a generic ILP algorithm and a detailed description of the Aleph algorithm [Sri03], that uses a hybrid search strategy. We do so because in this thesis we present a set of ILP based frameworks that are grounded in the Aleph algorithm.

In Algorithm 1 we present a generic top-down ILP algorithm. In this algorithm, $E = E^+ \cup E^-$ is the set of positive and negative examples, L is the language bias, B is the background knowledge and H is the hypothesis theory. The function $covers(B, H, e)$ is a boolean function that represents the notion of coverage. For each example $e \in E$, we say that hypothesis H covers an example e with respect to background knowledge

B if $B \cup H \models e$ and write $covers(B, H, e) = true$. The clause $\top \leftarrow$ represent a clause with no body and whose head is the most general clause, typically the target predicate with only free variables. This type of algorithms starts from the most general clause and search for the *best clause* through repeated θ-subsumption specializations. The specialization operator, that is also known as refinement operator, maps a clause c to a set of specializations $\rho(c) = \{c' \in L | c \preceq c'\}$. The *best clause* is a specific clause with the best score according to some metric like number of covered examples, accuracy, compression and minimum description length, etc. If the best specific clause does not cover any negative example and compresses the data, the clause is added to the current hypothesis theory. Next, all the positive examples covered by the current hypothesis theory are removed. If positive examples remain uncovered by the current hypothesis or the theory, we start again the covering loop. In the end the algorithm returns the hypothesis theory.

Algorithm 1: Generic Top-down ILP Algorithm

input : the set of examples, E; the language bias, L; the background knowledge, B
output: Hypothesis H

```
1  // Initialize the hypothesis space
2  H := ∅;
3  repeat
4      // Covering Loop
5      H_current = H;
6      c := T ←;
7      repeat
8          // Clause Specialization
9          Find the best refinement c_best ∈ ρ(c);
10         c = c_best;
11     until c cover examples E⁻ and c compresses the examples;
12     // Update the Hypothesis set of clauses
13     H := H_current ∪ {c};
14     // Remove positive examples covered by the Hypothesis
15     E := E − {e ∈ E⁺ | covers(B, H', e) = true};
16 until H = H_current or E⁺ = ∅
17 return H;
```

One instance of this generic algorithm is FOIL [QCJ95]. In FOIL algorithm, the best

specific clause is selected using an entropy based metric.

The other strategy is the bottom-up search. In this strategy the search starts from the examples (ground definite clauses) and uses a generalization operator (similar to lgg) to find the most general clauses. Typically, a generalization of two clauses is obtained through the use of the lgg. One example of such an ILP algorithm is GOLEM [MF90].

A hybrid strategy that combines ideas from the previous strategies is also available. In Algorithm 2 we present the pseudo-code of such an algorithm, the Aleph algorithm [Sri03]. The algorithm starts by randomly selecting an example $e \in E^+$. Then, using the language bias L, the most specific clause that entails example e is built. This most specific clause is known as the bottom clause and is represented by the symbol $\perp$. Next, and using the bottom clause as a bound, top-down search is used to find the *best clause* that is more general than the bottom clause. The top-down search uses refinements over the predicates and terms available in the bottom clause. When we arrive at a point where the bag of positive examples is empty or the theory does not compress, the algorithm returns the learned theory.

By restricting the top-down search to the predicates in the bottom clause, the algorithm substantially reduces the search space. Progol [Mug95] is another example of an ILP algorithm that uses this hybrid search strategy.

Unlike the majority of ILP algorithms that use a covering approach, TILDE [BR98] uses a divide-and-conquer strategy to induce logical decision trees. Using the learning from interpretations paradigm, where each example is represented by a set of definite clauses, the algorithm induces C4.5 like decision trees by upgrading heuristics to deal with clauses. The major difference are the tests put in the internal nodes, here every test is a conjunction of literals. Search for the best test is done using a refinement operator under θ-subsumption by adding literals, one or more, with the highest gain ratio (or gain) based on language specifications introduced by the user. Beyond language specifications that restrict the mode, types and number of conjunctions to be put in a test, the user must also supply a set of examples and background knowledge.

Van Laer [LR00] describes a methodology to upgrade propositional learners to FOL preserving the structural methods and techniques of the propositional learner. The methodology is a generalization that extends heuristics and paradigms of propositional learners. Grounded on learning from interpretations paradigm, θ-subsumption and user biasing the author upgrades CN2 propositional learner to the ICL relational learner.

Algorithm 2: Aleph Algorithm

input : the set of examples, E; the language bias, L; the background knowledge, B

output: Hypothesis H

```
1  // Initialize the hypothesis space
2  H := ∅;
3  repeat
4  |  H_current = H;
5  |  Select a positive example e ∈ E+;
6  |  Build the bottom clause ⊥_e for example e;
7  |  Top-down search the subset of literals in ⊥_e to find the best clause c;
8  |  // Update the Hypothesis set of clauses
9  |  H := H_current ∪ {c};
10 |  // Remove positive examples covered by the Hypothesis
11 |  E := E − {e ∈ E+ | covers(B, H', e) = true};
12 until E+ = ∅ or H := H_current
13 return H;
```

2.4 Propositionalization

The complexity of multi-relational problems and the availability of efficient, despite less expressive [Fla99], propositional algorithms, like C4.5 [Qui93], motivated the development of a set of techniques that translate a multi-relational database into a single table. These techniques are known as *propositionalization* techniques [KRZ+03a].

To use any propositionalization technique it is required that in the multi-relational database there exists a clear notion of individual and the learning task is individual focused [KRZ+03a], e.g. are multi-relational classification tasks. In the obtained single-table, each individual is characterized by a set of features/attributes that summarize information that exists in non-target tables.

The existing propositionalization approaches can be characterized according to the data translation completeness and the methodologies used to summarize the data. Concerning translation completeness, we can have complete or partial conversion. In complete conversion no information is lost. The main issue of this type of conversion is, typically, the large size of the single table generated by the translation process. Using this lossless conversion we can easily run out of computer disk space. In the

partial conversion we search for a subset of features that summarize the most relevant information recorded in the database. Typically, a heuristic search and an aggregation strategy is used to summarize the data available in the non-target tables.

The methodologies used to propositionalize the multi-relational data can be grouped into database-oriented and logical-oriented techniques.

2.4.1 Database-oriented Approach

The propositionalization techniques presented in this section use relational algebra operations (mainly SQL, a query language for relational databases) and the aggregation functions implemented in a database system [GMUW09] to convert a multi-relational database into a single table.

Most existing database oriented approaches [KHS01, KW01] exploit the relational database scheme, mainly the primary key and foreign key relationships, the table join operation and aggregations operations to summarize the attribute values.

In [KHS01, KW01] the foreign keys of the examples registered in the target table are used to recursively join the tables reachable from the target table. The information of tables in the path from the target table will be included in the resulting join table. Using this procedure a set of join tables is built. Each one of these tables includes the target table.

In this set of join tables the attributes of non-target tables are summarized using aggregation operations. If the attribute is numeric, the *Average, Maximum, Minimum or Sum* values of the attribute can be computed to summarize the values. If the attribute is nominal, and if the cardinality of the attribute is small, in the join table we can add a number of new attributes that equals the cardinality of the nominal attribute. The value of each one of these attributes can be obtained using the *Count* function.

In the end all the join tables are combined into a single table. In the obtained table each row is associated with a unique individual/example and registers the individual features.

The main issue with this approach is the cost of joining tables. Some strategies like virtual joins, also known as tuple-id propagation [KRZ+03a, YHYY04], were proposed. This technique consists in propagating the primary key of each individual to all non-target tuples reachable from the target table. In [YH05] the authors combine virtual

join with the use of heuristics that select links in a costless way.

2.4.2 Logic-oriented Approach

In this section we present propositionalization techniques that summarize data available in non-target tables using FOL features. To learn such FOL features, techniques from ILP are used.

This propositionalization approach can be seen as an ILP based feature construction problem. The goal is to use an ILP algorithm to find a set of clauses

$f^1(X) : -l_1^1, l_2^1, \ldots, l_{n_1}^1$
$f^2(X) : -l_1^2, l_2^2, \ldots, l_{n_2}^2$
$\ldots$
$f^n(X) : -l_1^n, l_2^n, \ldots, l_{n_n}^1.$

The head of each clause consists of 1-ary predicate where X is the identifier of a database individual. The literals l^i in the body of each clause are part of the background knowledge and define each feature f^i. Each one of these f^i features is then mapped into a Boolean attribute in a relational target table. The attribute value is computed as follows. For each example X in the relational database that the clause f^i succeeds, the attribute takes value *true*. Otherwise, the attribute takes the value *false*.

The first propositionalization system of this type is LINUS [LDG91]. This system runs an exhaustive search procedure to find a constrained set of function-free non-recursive Horn clauses with negation. All variables that appear in the body of this clauses must also appear in the head of the clause. Moreover, the obtained clauses can include background knowledge. After learning a set of clauses, this system takes the literals in the body of the clauses as new features. For instance, if LINUS finds the clause $father(X, Y) \leftarrow male(X), parent(X, Y)$, then $male(X)$ and $parent(X, Y)$ are features to be used by a propositional learner. To extend the descriptive power of the clauses found by LINUS, Lavrac and Dzeroski present the DINUS [LD93] and SINUS [KRZ+03a] systems. DINUS can find determinate clauses, i.e, it can find clauses having literals with variables that all appear in previous literals or if a new variable appears in a literal this variable can only have a unique binding. For instance, in clause $grandfather(X, Y) \leftarrow father(Z, Y), father(X, Z)$ the variable Z can have only a single binding. Moreover, DINUS was extended to find recursive clauses.

With these extensions DINUS can find more expressive clauses than LINUS without sacrificing too much in performance. The main issue with DINUS is that many real problems do not have the determinacy property. The SINUS system, improved on DINUS by allowing unconstrained clauses, making use of language bias in the feature selection and verifying if it is possible to unify newly found literals with existing ones, while keeping consistency between pre-existing variable namings, thus reducing the nal number of features. The major issue is that SINUS can generate redundant decomposable features.

The RSD system presented in [ZL06] introduces a more elaborate hypothesis language declaration and uses a search strategy that eliminates redundant decomposable features. This system uses a three step strategy to find features. In the first step, a user-defined syntactic language bias (the modes) is used to built FOL features. This step can be completely independent of the input data. In the second step, the constants available in the input data are used to instantiate the variables in the FOL features. Using the constant instantiation operation, a set of instantiated features is generated from each feature. In the last step, this set of instantiated FOL features is used to generate a propositional representation of the multi-relational database. The features are mapped into binary attributes and the value of each attribute is computed for each individual in the target table. For instance, the predicate declaration : $-modeb(2, father(+name, -son))$, where the recall number (2) specifies the amount of maximal successful calls to this predicate in a clause, the $+T$ and $-T$ modes specify that the predicate argument associated is an input $(+)$ and an output $(-)$ variable, respectively, of type T.

Using this predicate declaration we can find constant-free features like

$$grandfatherOfTeenager(X) : -father(X, Z), father(Z, Y), age(Y, W), \\ instantiate(W)$$

saying that X is a grandfather of a teenager with a given age W that should be bound to a number of constants and generate a feature like

$$grandfatherOfTeenager(X) : -father(X, Z), father(Z, Y), age(Y, 12).$$

Both the SINUS and RSD systems include mechanisms to eliminate the exponential increase in the number and the size of the features. Both include constraints on the number of predicates, the maximum number of literals, variables and constant values to use. With these constraints the two systems obtain significant savings in terms of memory and run-time but may fail to explore the full solution space. One important

contribution of the SINUS system is the post-processing procedure that can translate the hypothesis found by the propositional learner back to flatten Prolog. With this procedure the user can obtain high understandable theories.

The WARMR [DT99] system can find frequent Datalog queries that can be translated to relational features. The Datalog representation formalism is a subset of clausal logic that is restricted to function-free definite clauses. The WARMR algorithm uses WRMODE language declarations to constrain the query language to a set of useful patterns.

In [RY01] Roth and Yih present a two step system that represents features using a subset of FOL and a collection of graph substructures that represent instances of the domain.

In all the systems present above feature extraction is decoupled from the theory learning phase. However, there are works which tightly integrate feature extraction with learning. Recently a wide range of the later systems have been presented.

In nFOIL [LKR07] the authors tightly integrate naive Bayes learning with FOIL. The naive Bayes classifier is used to guide the search for the most interesting patterns. In kFOIL [LPRF06] the generation of clauses is driven by the performance of support vector machines. The use of kernels allows this work to address both classification and regression problems. In SAYU [DBD+05] rule learning and Bayesian network learning are integrated. The Bayesian network is build in an iterative process that includes, in each step, the candidate rules that improve the performance of the classifier.

In nFOIL and kFOIL papers the authors argue that the results of these systems show that they generate high accurate models than the ones obtained with systems that decouple feature extraction and classifier learning. In SAYU [DBD+05] the authors observe that this is not always the case. Moreover, the authors report results that show that integrated systems produce more compact theories than the two-step systems.

Flach [FL04] introduced 2BC algorithm that extends the propositional Bayesian to relational data by defining a probability distribution over structured objects/predicates. The probability of each predicate is defined to be the joint probability of the object properties and the probability of related objects.

2.4.3 Propositional Classifiers

In this section we briefly introduce supervised learning and an example of inductive methodologies. In a basic inductive learning task, the goal is to obtain a classification model, $Y = f(X^1, \ldots, X^n)$, from a set of labeled examples $< x_1, \ldots, x_n, y >$. Each element $x_i \in Dom(X^i)$ is a feature, also known as an attribute, that describe each example and $y \in Dom(Y)$ is a discrete variable known as the class label of the example. If we have a two-class problem, i.e. $\#Dom(Y) = 2$, we say we have a binary classification problem.

Given that the set of examples is, typically, incomplete, we find a hypothesis function $\hat{f}$ that approximates Y. Moreover, the learned $\hat{f}$ function can be used to predict the class of unlabeled examples.

An example of a classifier that learns a decision tree using a set of labeled examples as input is the C4.5 algorithm [Qui93]. This algorithm has two main steps. In the first phase a divide-and-conquer strategy is used to build a tree. In the second phase a reduce-error pruning strategy is used to get a more general, with low variance, model.

Growing the tree uses a divide-and-conquer strategy and has four main issues: number of attributes to put in the test node, type of splits, splitting rule and leaf assignment. Attributes used in each inner node can be of three types: binary, nominal or continuous. Each one of these attributes can be used as inner node tests, univariate tests, using only one attribute and multivariate tests that use multiple attributes as an inner node test. The type of split could be binary or multiple-splits. When we parametrize the algorithm to do binary splits, in a parent inner node each split always creates two child nodes. With multiple-splits in each inner node the algorithm chooses the number of splits accordingly to the number of attribute values and some splitting rule. In each inner node the attributes that maximizes a merit function is chosen as a split rule. Leaf values can be one of two types: class value, where to each leaf is assigned a class label; or a probability that is the ratio between the number of instances that arrive at the leaf and the number of instances in the parent node. Having possible different configurations C4.5 defaults use binary splits, univariate tests and gain ratio as the merit function. Morever, the algorithm can be used both to solve classification and regression problems. In the classification setting the leafs contains class-values and in the regression setting each leaf contains a real number.

Since the growing process can cause over-fitting it can be defined a minimum number of instance to split and prune the tree. Moreover, C4.5 uses reduced error pruning

strategy to prune the induced tree.

For a training set of N examples having D features, the time complexity of training a decision tree can be approximated as follows. Assuming a binary decision tree, the number of levels in the tree is O(log N). At level i there are 2^i nodes and at each node an entropy calculation and a sorting operation needs to be done. Calculating the entropy can be done in O(N) time for each feature, and sorting can be achieved in O(N log N) time. As a result, the complexity at each node can be approximated as O(DN log N). At each level, there are a fixed number of nodes, so this is also in O(DN log N). Consequently, the total time complexity of learning a decision tree is O(DN log N).

2.5 Learning Issues

In this section we discuss some important issues that can arise when we use inductive inference to learn a classification or descriptive model. We discuss the limitations of the representation language and the strategies and metrics used to select the most interesting features.

2.5.1 Extending Language Bias

One of the main limitations of inductive learning systems is that they learn models using only the original data descriptors [BM98, Sta96, Kra95]. In general, the original descriptors do not provide the best information for learning a specific target concept. This problem becomes apparent when the expressiveness of the algorithm hypothesis language is limited or there exists a high correlation between the primitive descriptors. For instance, when we use high correlated and inadequate original descriptors to learn a propositional decision tree we can get high complex and low accurate models [BM98]. Typically the learned model overfits the training data. In the FOL setting we are not limited only by the hypothesis language but also by the well known scalability issues of ILP algorithms [BS03]. If the descriptors are not adequate we can fail to explore the hypothesis space and find general concepts. Moreover, models learned using only original descriptors can be hard to understand by humans [PH90].

To address the limitations of the original descriptors and build adequate descriptors, a set of methodologies can be used. In [WM94] and [Kra95] the authors study a set

of methodologies to build new descriptors. Whereas the methodologies presented in [WM94] were developed to build new descriptors using propositional data and learners, the methodologies presented in [Kra95, Sta96] were developed to learn new descriptors for the FOL learners. The former methodologies are known as *Constructive Induction* methodologies and the later ones are known as *Predicate Invention* methodologies.

In the following lines we give an overview of the strategies used to construct and select new features/predicates.

2.5.1.1 Constructive Induction

In the propositional setting there are two main reasons to use constructive induction. One is related with the limitations of the propositional representational language. There may be a need to learn high level (more abstract) features from low-level features. The second reason is the algorithm bias. For instance, algorithms that use greedy selection of features need a large set of conditions to express interrelationships among dataset features [Kra95].

The existing constructive induction system can be categorized in four groups [WM94]:

- In *Data-driven* methodologies, the new attributes are build by analyzing only the original data. Typically, new attributes are build using arithmetic or Boolean operators.

  Zheng [Zhe98] provides an overview on building new descriptors by combining primitive attributes using Boolean operators. The most used operators are conjunction, disjunction, negation, M-of-N and X-of-N.

- In *Hypothesis driven* methodologies, the new attributes are build by using an inductive learner and by analyzing the induced hypothesis. In general, these systems have two main phases. Firstly, an inductive system, like a decision tree learning algorithm, is used in conjunction with a feature selection strategy to find and select the best features/rules. Then the selected best features are used to augment the descriptive power of the original descriptors.

  In [PH90] decision trees, decision lists and Boolean operators are used to build high-level attributes. The authors observe that even some simple Boolean functions represented in DNF (Disjunctive Normal Form) are hard to learn using a decision tree model. In these cases, a tree model with a set of identical subtrees

is generated (this issue is known as a replication problem). Beyond producing less legible and complex models, replication can cause a decrease in accuracy.

- In *Knowledge driven* methodologies, experts provide domain knowledge that, sometimes in conjunction with the original data, is used to build new features.

- *Multi-strategy* methodologies combine two, or more, of the above strategies to build new features.

2.5.1.2 Predicate Invention

In ILP systems the need to build new predicates is more related with limitations of the language bias, that restricts the space of potential solutions to consider.

In [Kle52] Kleen proved that the introduction of a new predicate is useful for finding a finite hypothesis:

Theorem 2. *Any recursively enumerable set C of formulas in a FOL language $L_0 \subseteq L$ is finitely axiomatizable in a FOL language L using additional predicate symbols except for those in L_0.*

In the ILP setting, C is the set of positive examples E^+.

Moreover, in [Sta93] it is observed that for infinite or undecidable languages, like FOL, predicate insufficiency detection has to be made heuristically.

In [Sta93, Kra95] three main predicate invention strategies are identified:

- In *Reformulation and Inverse resolution* approach new intermediate predicates are introduced to generate more compact representations from the learned theories, i.e., to reformulate theories. This type of approaches build new predicates by using a kind of reverse deductive explanation, which replaces intermediate predicates by their definition. Some of the existing systems directly invert deductive resolution steps. Other systems build new predicates as combinations of known literals. This later systems use an abstract schema to restrict literals used in the combinations.

  For instance, FOCL [SP91], an extension of FOIL [QCJ95], overcomes the limitation of FOIL to add only the literal with the maximum information gain to the body of the clause. FOCL builds combinations of literals restricted to a schema.

- The *Data-Driven methods* try to detect situations where the language bias is insufficient to learn a finite theory. In that case a new predicate is invented. The decision to build a new predicate is heuristic whenever it is not possible to make an exact decision.

  MENDEL [Lin91] introduces an inverse resolution system that builds new predicates only if the theory grows beyond a fixed bound.

- *Clause-refinement methods* make an over-general clause consistent by adding literals which contain new predicates [SMB92]. This type of predicate invention is useful in discriminative applications where positive and negative examples are available.

2.5.2 Feature Selection

Many of the descriptors available in a dataset can be redundant or irrelevant for a classification or prediction task. Using such descriptors to learn a concept could reduce algorithms performance and the accuracy of the learned models, especially when there is an insufficient number of instance. Thus, there is a need to introduce techniques that remove statistically related attributes and/or select a subset of attributes such that the resulting probability distribution of the different classes is close to the probability distribution using the original set of descriptors [HKP11].

Most of the existing works cast the feature selection problem into a search problem. Typically a search method is combined with a metric to evaluate the merit of each feature or subset of features. Moreover, heuristic methods are typically used to explore the search space. Furthermore, the existing feature selection methods can be grouped according to the type of selection methodology. Regarding this, we have two main strategies: the filter and wrapper methodology [YL04]. The wrapper method is grounded in a predefined mining algorithm and uses the performance of the learned model to select features. For instance, we can learn a decision tree model and remove all attributes that do not appear in the inner nodes. The filter method relies on the data descriptors and use a measure of interest to select or sort features. Different from the wrapper methodology, the filter selection method does not use a mining algorithm. Both methodologies can be parametrized to select a set or a single attribute.

The metrics used to select the most interesting attributes typically measure the degree of independence among attributes or independence between the attributes and a certain class.

Table 2.2: Contingency table

	C	$\bar{\text{C}}$	
A	n(AC)	n(A$\bar{\text{C}}$)	n(A)
$\bar{\text{A}}$	n($\bar{\text{A}}$C)	n($\bar{\text{A}}\bar{\text{C}}$)	n($\bar{\text{A}}$)
	n(C)	n($\bar{\text{C}}$)	N

Consider the classification association rule $A \rightarrow C$ where A is an itemset/attribute and C is the class attribute. Moreover, suppose that $n(A)$ and $P(A)$ define the number of occurrences and the probability of attribute A, respectivelly, and N is the number of observations. Then we can build a contingency table (see Table 2.2) and compute a set of metrics to select the best attributes. In Table 2.3 we present a sample set of metrics [TKS02].

Then, we can combine any of the metrics presented in Table 2.3 with a basic heuristic method to select a subset of features. These metrics can be used both in filter and wrapper type feature selection. In [HKP11] Han presents four basic types of heuristic methods to select features: stepwise forward selection, stepwise backward elimination, a combination of forward selection and backward elimination and decision tree induction.

In [GH06] Geng surveys interesting measures and suggests a set of criterion's to evaluate the interestingness of association rules and classification rules patterns. The authors consider that association rules are descriptive patterns and classification rules are predictive. Considering interestingness as an imprecise and broad concept, the authors introduce nine criterions to objectively qualify patterns. The interest of a pattern depends on: conciseness, coverage, reliability, peculiarity, diversity, novelty, surprisingness, utility and actionability features. Since some of these features are domain-dependent, the author also categorizes them into objective, subjective and semantic-based. Objective measures like conciseness, reliability, peculiarity and surprisingness, are extracted from raw data; features that not only use raw data but also user domain and background knowledge such as surprisingness and novelty are subjective measures; other features like utility an actionability are considered semantic-based measures because they use interpretation and explanation of the patterns. Based on the nine criteria the authors characterize a set of metrics that provide an automatic way to determine pattern interestingness. Boolean, partial ordering and rank measures could be created. For instance, the support of a frequent itemset can be used as a

Table 2.3: Selection metrics for association rules

Measure	Formula
Support	$P(AC)$
Confidence	$P(C\|A)$
Lift	$\frac{P(C\|A)}{P(C)}$ or $\frac{P(AC)}{P(A)P(C)}$
Cosine	$\frac{P(AC)}{\sqrt{P(A)P(C)}}$
Conviction	$\frac{P(A)P(C)}{P(AC)}$
Leverage	$P(C\|A) - P(A)P(C)$
Laplace	$\frac{N(AC)+1}{N(A)+2}$
Information Gain	$\ln \frac{P(AC)}{P(A)P(C)}$
Correlation	$\frac{P(AC) - P(A)P(C)}{\sqrt{P(A)P(C)P(\overline{A})P(\overline{C})}}$
Chi-square	$N \times \frac{P(A)P(C)}{(1-P(A))(1-P(C))}$

measure of generality/coverage of a pattern. Moreover, using this metric with a real value threshold we can suppress less general or low utility patterns in a partial ordered set of patterns.

Webb [WZ05] observes that support and lift are complementary measures that can be used sequentially to discover interesting rules. One major drawback to this approach is that rules with high lift values apply to few cases since they have low coverage. In addition, the author argues that the combination of support and confidence not always select potential interesting rules. In some cases the user knows what to search and a specific rule, different from confidence measure could be more suitable. To address these issues Webb introduced the *Kord* algorithm. *Kord* algorithm takes as input a dataset, one measure of rule interest, a set of constraints, like the number of conditions in the antecedent, and the number of rules to be found. This algorithm explores the space of possible solutions and returns a specified number of sorted rules. The rules order is induced by the user specified interesting measure. The author experimental evaluation considers the leverage measure. Using this measure the dependence gain of the association rule can be measured.

In [BMS97] Brin introduces an alternative to the support-confidence framework. This author presents an algorithm that selects interesting correlations rules using chi-square statistics. The authors argue that rules with high support are statistically significant. In all major problems, mainly in market basket analysis, being frequent is highly interesting. Some investments and market campaigns are supported by this measure. On the other side, the confidence metric is a directional measure that can be misleading and can be unable to detect negative correlations.

Lenca [LMVL08] presents a methodology to select interesting non-redundant features based on the analysis of the properties of the measures and expert knowledge. This author observes the difficulty in defining an objective measure of rule interest and introduces a set of eight properties that can describe any measure. Using these properties, the authors study and describe a set of 20 measures. Some of these properties can be easily and objectively described without user contribution. One of these measures is rule asymmetry. Other properties like intelligibility are hard to define and evaluate incisively without user contribution. The methodology uses PROMETHEE, a multi-criteria decision making (MCDA) method, to select and visualize interesting measures based on ranks. The ranks of each measure results from a pairwise comparison that uses measures properties and respective weights.

In [YL04], the FCBF algorithm is introduced to find a subset of relevant and non-

redundant features. The obtained subset of features is found by decoupling relevance analysis from redundancy analysis. The authors say that almost all works select individual features by ranking them according to some heuristic and then select the top ones. This procedure, does not remove redundant features that are equally ranked. Using the filter method, the FCBF algorithm uses a symmetrical non-linear heuristic, named Symmetrical Uncertainty (SU), to rank features by measuring correlation between each feature and the available classes. Then all features that have a SU value higher than a selected threshold are considered relevant. Next, redundant features are eliminated using the SU metric to measure correlation between the selected features.

In supervised learning, the classical approach is to select features using heuristics that compute the degree of association/correlation between features and classes.

Liu [LHM98] propose an algorithm that integrates classification and association rule mining. Introducing minor changes to the *Apriori* algorithm, this algorithm starts by finding all classification association rules (CAR), a subset of association rules where the antecedent part is an attribute value conjunction ($attribute = integer - value$) and the consequent part is the class value. In the second phase, the confidence and the support values of each rule are used to define a total order relation on the set of CARs. The algorithm sorts all CAR rules and generates a raw classifier. Then, to refine the classifier, the sorted set and a covering strategy are used to select the best rules. The last rules, that do not improve accuracy, are deleted. The final classification model consists of the higher ranked rules and a default class rule, which labels an example as the majority class of the dataset.

Another work that selects features using a single measure and then induces a classification model was developed by Terabe [TWM+02]. First, the data descriptors are extended using classification rules. Using a modified version of *Apriori*, the algorithm finds new attributes that are conjunctions of primitive attributes. The authors use an approximation of information gain to select the most interesting candidates descriptors. Next, the selected descriptors are mapped and a new extended dataset is build. Last, the extended dataset is used as input to the C4.5 decision tree induction algorithm.

These algorithms proved to be effective but there are interesting problems that are best solved in rank-order spaces.

In [SVRH05] four strategies to select features in rank-order spaces are presented. The presented methodologies are used in spaces where features have a common domain and are easy to sort. To search for an optimal subset of features a stepwise meta-algorithm

is used. In each iteration a single feature is select for removal. This selection procedure is done in the rank-space with one of the four proposed strategies. Two of these strategies are grounded in information theory techniques and the other two techniques use algorithms from permutation theory. The algorithm removes features until we get a subset with a user-defined size or a divergence criterion is satisfied.

2.5.3 Classifier Evaluation

Classification is a multi-step process where a model is used to assign a class label (or category) to a set of unclassified examples, that are described by a set of features. In machine learning, classification is an instance of supervised learning where the label of each example available in the training set is known in advance. Typically, in the first step a classification model is learned in a train set that contains labeled data. Second, a test set with labeled data is used to evaluate the performance on unseen examples. Last, the best model is used to classify new and unlabeled examples. In this section, we give an overview of strategies used to evaluate the performance of classification models on novel data and how do we compare the performance of two classification models.

2.5.3.1 Evaluation Criteria

The usefulness of a classification model is related with the task we have in hands and the availability of resources. For instance, in a predictive task we are more interested in the predictive accuracy while in a descriptive task we are more interested in compact and understandable classification theories. In the following lines we present a set of criteria to evaluate classification models.

Predictive Accuracy This criterion is related with the ability of a classification model to correctly predict a class label of an unseen example. This is the most used and important criteria that gives an important sketch on the generalization ability of a predictor.

The accuracy of a predictor model is the fraction of correct predictions made by the predictor. In the simplest case where we only consider a zero-one loss function we can define accuracy for any classification model f as:

$$accuracy(f) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(y_i = \hat{y}_i),$$

where N is the number of samples, $\hat{y}_i = f(\vec{x_i})$ is the predicted class value for the ith vector sample and y_i is the true class label of the $i-$th sample, and $\mathbb{1}$ is the indicator function that takes value 1 when expression $y_i = \hat{y}_i$ is true and takes value zero when $y_i \neq \hat{y}_i$.

In most works the predictive error, that is the counterpart of predictive accuracy, is used to present and analyze the results. In this case the indicator function is replaced by a loss function, typically the zero-one loss function is used:

$$error(f) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{y}_i),$$

where

$$L(y_i, \hat{y}_i) = \begin{cases} 0 & \text{if } y = f(\vec{x_i}) \\ 1 & \text{if } y \neq f(\vec{x_i}) \end{cases}$$

Other loss functions exist that can be more useful in other circumstances. Among these loss functions we can find some that can weight misclassification errors and other that associate costs to misclassifications [FGMB10, FGMB13].

Interpretability and Simplicity Interpretability is related with the understanding and the insights provided by the classification model. To analyze the interpretability of a classification model there are no metrics and typically only domain experts can measure this subjective property of classification models. For instance, FOL decision rules are much more expressive and understandable than a neural network model [ADT95].

Moreover, the understandability of the learned models can be related with the simplicity of the obtained models. For instance a compact decision tree or a small number of classification rules are more understandable and give more insight than a neural network classification model [ADT95]. A well known method to select understandable and simple theories is the minimum description length (MDL) principle [Ris78]. This principle is inspired in the Occam Razor and states that the best hypothesis for a given dataset is the one that leads to the largest compression of the data.

In practical applications this principle means that we prefer small models over large ones. Typically we prefer decision trees with a small number of nodes and rule sets with few rules.

Speed and Memory Despite the hypothetical high expressiveness and generalization accuracy of a given algorithm, it can be useless if there are no computer resources available to run such a powerful algorithm. For instance, if we run an ILP algorithm in a computer with the current standard specifications to solve a large and complex problem we may fail to learn a classification model. Moreover, the probability of failing increases if we define a high expressive language bias.

Therefore, we must evaluate classifiers according to the run-time and amount of computer memory used to learn a theory. As the amount of memory used by one algorithm fluctuates during the learning process, typically we compute the peak of memory usage. Regarding the time and memory usage to test the classification model, these are almost always negligible when compared with time and memory usage at learning time. Moreover, the scalability of the algorithms can be important. It can be important to develop algorithms that scale to large amounts of data. Thus, in highly expressive models we can use heuristics to keep the problem tractable. As expected, we prefer algorithms that use few memory and spend less time learning a classification model.

Robustness Other important issue when learning a classification model is its sensibility to noisy data. Typically we want to develop, or use, robust algorithms that can correctly label examples given noisy data or data with missing values.

2.5.3.2 Estimating the Performance of a Predictive Model

When learning a model, typically we have algorithms that exhibit high prediction capabilities in the train set but perform poorly when predicting the labels of unseen data. This overfitting behavior of learning algorithms is well-known and a set of evaluation methodologies were developed to address this issue.

Here we present a brief overview of the existing validation methods and give a detailed description of stratified k-fold cross-validation methodology, the validation methodology used in this work. Moreover we present a detailed description of the methodology used in this work to evaluate multi-class and non-symmetrical classifiers [Für02].

Resubstitution validation In *resubstitution validation* the model is learned and tested using all the available data. The main issue with this evaluation strategy is that the learned models overfit the data and will perform poorly on future unseen data. A strategy introduced to eliminate the overfit issue of *resubstitution validation* is *hold-out validation*.

Hold-out validation In *hold-out validation* the data is split into two non-overlapping folds, the train and the test set, one fold is used to learn a model and the other fold is used to evaluate the model. The main issue with this methodology is that not all the available data is used to learn and important examples can be discarded.

The most important statistical method to estimate the generalization ability of an algorithm is cross-validation [Für02]. There exist a set of variants of this methodology all of them sharing a simple procedure. First, the data is divided into two segments, then one segment of data is used to train a model and the other segment is used to validate the learned model.

k-fold Cross-Validation In *k-fold cross validation* the data is first partitioned into k equally sized folds. Then, an iterative process with k iterations starts. In each iteration a classifier is trained using all the data available in $k-1$ folds and is tested in the k fold not used for training. This procedure guarantees that all the data is used both for training and testing the classifier. This way the results will not be skewed by the choice of the examples used to train or test the classifier. One important issue with this methodology is the way we arrange the data in each fold. If each fold does not represent the overall class distribution we can end-up with misleading results. One strategy to eliminate this issue is to use a variant called *stratified cross-validation*. With this strategy the examples are distributed among the folders such that every folder is a representative of the class distribution of the dataset. In [Koh95] Kohavi analyzes a set of methodologies to estimate accuracy and suggests that 10-fold cross-validation is the best method to use. He observes that this technique provides less biased estimation of the accuracy.

Two other well known variants of the $k-fold$ cross validation technique are *leave-one-out cross-validation* and *repeated $k-$fold cross-validation*. In the leave-one-out cross-validation technique we split a dataset with k examples into k folds, each fold containing one example. Then, the classifier is trained using $k-1$ folds and the remaining example is used to test the learned classifier. The main issue with this

technique is the high variance of the results. To obtain a large number of estimates other than k, we can run a technique known as *repeated k−fold cross-validation*. In [RTL09] we can see a detailed analysis of the validation methods presented here.

2.5.3.3 Learning and Evaluation of Multi-class Classifiers

Often real world classification problems have multiple classes but most of the existing learning algorithms are inherently binary. Thus, to solve these multi-class problems either we use a multi-class learning algorithm or we employ a class binarization technique to cast the multi-class problem into a set of binary problems [Für02]. As in this work we want to learn FOL classification models using binary FOL rule learners, we analyze binarization techniques to solve multi-class classification problems.

The classical approach in class binarization is known as *one-against-all*. To use this technique to solve a problem with c classes we build c binary problems. Each one of these c binary problems uses the examples in the $i-$th class as the positive examples and all the other examples of classes $j = 1, \ldots, c$, such that $j \neq i$, as the negative examples. This technique is used in CN2 rule learner [CB91] and is also known as *unordered class binarization*. This technique requires a combination criterion to obtain the final prediction. Typically, the votes of all binary classification models are counted and the class with the higher number of votes is used to label the example. An ordered version of this binarization technique that does not require a combination criterion can also be used [Für02]. The problem with these approaches, that join all the examples of the non-positive class into the negative examples set, is that in most of the times we get a large and unbalanced set of binary problems.

One alternative approach to these binarization techniques is *pairwise/round robin* binarization [Für02]. This technique translates a $c-$class problem into a set of $c(c-1)/2$ binary classification problems, where each binary problem is built using only examples of two classes $< i, j >$ such that $i = 1, \ldots, c-1$ and $j = i+1, \ldots, c$. Then, the classification model is learned using only examples of classes i and j and is tested using all the examples of all classes. To obtain the final classification the votes of all binary classification models are combined and the most voted class is used to label the example. In case of ties the class with the largest number of examples is used to label the example. In [Für02] it is shown that this technique produces high accurate classification models than the *one-against-all* strategy. Moreover, it is observed that pairwise decision boundaries can be considerable simpler and are learned in a more efficient way than the decision borders of the *one-against-all* strategy.

In the above definition of *pairwise/round robin* binarization we assume that the base classifier is class-symmetric, which means that the model learned for the $< i, j >$ classification problem is the same model learned for the $< j, i >$ classification problem. Examples of class-symmetric learners are the c4.5 decision tree algorithm [Qui93] and the SVM algorithm [VD04]. When we use non-symmetric learners like FOL rule learners [QCJ95, Sri03] as the base classifier, we learn rules that explain the positive examples. Therefore, depending on the class chosen to be the positive class, either the i or j class, we get different classification models. One strategy used to deal with non-symmetric base classifiers is *double round robin* [Für02]. This class binarization strategy translates a $c-$class classification problem into $c(c-1)$ binary classification models. This means that we learn a classification model for the $< i, j >$ binary problem and another classification model for the $< j, i >$ binary problem, for $i = 1, \ldots, c$ and $j = 1, \ldots, c$, and $j \neq i$. The final classification of each example is obtained using the same strategy described above for the round robin technique.

As an example, in this thesis we run experiments with the five class Protein dataset presented in Appendix B. Thus, we build 20 binary classification models in each one of the ten folds and use the majority vote among all pairwise classification problems to classify each example available in the test set. In case of ties we decide in favor of the majority class. Moreover, for this dataset we report the mean number of patterns, the peak memory usage and the run-time of each algorithm/framework of the first 10 binary problems (f1 vs f2; f1 vs f23; f1 vs f37; f1 vs f55; f2 vs f23; f2 vs f37; f2 vs f55; f23 vs f37; f23 vs f55; f37 vs f55). We use this strategy to clarify the advantage of mining each database partition individually. Different from this strategy, we compute the mean generalization accuracy of each system using all the results obtained by the 20 binary classification models.

2.6 Pattern Mining: Itemsets and Sequences

In this section we introduce a set of propositional and first order logic algorithms that can find frequent patterns. More precisely, we introduce algorithms that can find frequent itemsets and sequences that appear in propositional and FOL datasets.

2.6.1 Frequent Itemset Discovery

In this section we introduce concepts and algorithms especially developed to discover unordered frequent patterns from propositional and FOL datasets.

2.6.1.1 Finding Propositional Patterns

The goal of propositional frequent itemset mining (FIM) is to discover interesting relationships between variables available in transactional datasets. A typical example of a FIM task is market basket analysis. In this task the goal is to find associations (rules) among products that a customer buys in every supermarket transaction [AS94, RGF12]. For instance, knowing that 90% of the customers that buy champagne also buy caviar can be valuable for a retailer. This type of information can be used in many decision making processes such as planning marketing strategies or space occupation.

In the following lines we introduce some concepts and formally define the FIM task.

Let D be a dataset of transactions and $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items. A transaction $T \in D$ is a set of items such the $T \subseteq I$. Each transaction T is associated with a unique identifier *tid* and can also be represented by a tuple $< tid, T >$. A $k-itemset$ is an itemset that contains exactly k items.

We now introduce the *contain/covers* relation and the *support* of an itemset, two key concepts in FIM. We say that an itemset A is contained (or is covered) in a transaction T if $A \subseteq T$. Using this relation we define the support of an itemset A in the dataset D, the $supp_D(A)$ quantity. The *absolute support* of an itemset A is the number of transactions $T \in D$ that contain the itemset A, i.e., $supp_D(A) = \|\{k \in \{1, 2, \ldots, n\} | A \subseteq T_k\}\|$. Moreover, the *relative support* of an itemset A is the percentage of transactions $T \in D$ that contains the itemset A, i.e., $supp_D(A) = \frac{1}{n}\|\{k \in \{1, 2, \ldots, n\} | A \subseteq T_k\}\|$, where n is the number of transactions in D. Everywhere in the text we write *support* to mention the relative support quantity.

Now the FIM task can be formally defined.

Given:

- A set of items $I = \{i_1, i_2, \ldots, i_n\}$;
- A dataset of transactions $D = \{T_1, T_2, \ldots, T_n\}$;
- A minimum support threshold $\lambda \in [0, 1]$.

Find:

- The set of frequent itemsets $F_D(\lambda) = \{A \subset I | supp_D(A) \geq \lambda\}$.

The brute force strategy to solve this problem is to generate all possible itemsets, compute the support of each itemset and then prune the infrequent itemsets, the ones with $supp_D(A) \leq \lambda$. The main issue with this strategy is that the number of itemsets grows exponentially with the number of items in I. In Figure 2.1 we present a lattice structure that can be used to enumerate all itemsets that can be generated from the set of items $I = \{s, m, c, b\}$.

Agrawal introduces in [AS94] the Apriori algorithm, a high efficient candidate-generation and test algorithm. This is a seminal work that inspired most of the pattern mining algorithms that we use in this work. Moreover, this algorithm can be used to generate Boolean association rules from the frequent itemsets. In the next lines we will introduce some important concepts and then we present the Apriori algorithm.

Based on the idea that the support of an extended itemset do not increase over any of its subsets, that is $\forall A \subseteq B : supp_D(B) \leq supp_D(A)$, we define the *apriori* property. The *apriori property* says that the superset of an infrequent itemset can not be frequent.

An *association rule* is an implication $A \Rightarrow B$, where $A \subset I$, $B \subset I$ and $A \cap B = \emptyset$. The *confidence* of an association rule $A \Rightarrow B$ is the percentage of transactions in D containing A that also contain B. Formally, we can define confidence using the *support* measure:

$$confidence_D(A \Rightarrow B) = \frac{support_D(A \cup B)}{support_D(A)}.$$

The Apriori algorithm uses a levelwise strategy to generate all candidate itemsets. Moreover, the Apriori algorithm uses the apriori anti-monotonic property to prune the search space, i.e, to prune some branches of the lattice (see Figure 2.1).

In Algorithm 3 we present the pseudo-code of the Apriori algorithm. In line 1 the Apriori algorithm scans the database D to find all frequent items with size one, the L_1 set. Then, in the main loop, lines 2 to 8, the frequent itemsets found in the level $k-1$, the L_{k-1} set, are used to find frequent itemsets of size k, the L_k set. In line 3 all the frequent itemsets of size $k-1$ are joined to generate candidate itemsets of size k. Below we present a detailed description of this function. Next, in the loop starting in line 4 we scan each transaction to compute the support of each candidate itemset $A \in C_k$. In line 5 the subset C_T of itemsets in C_k that are contained in the current

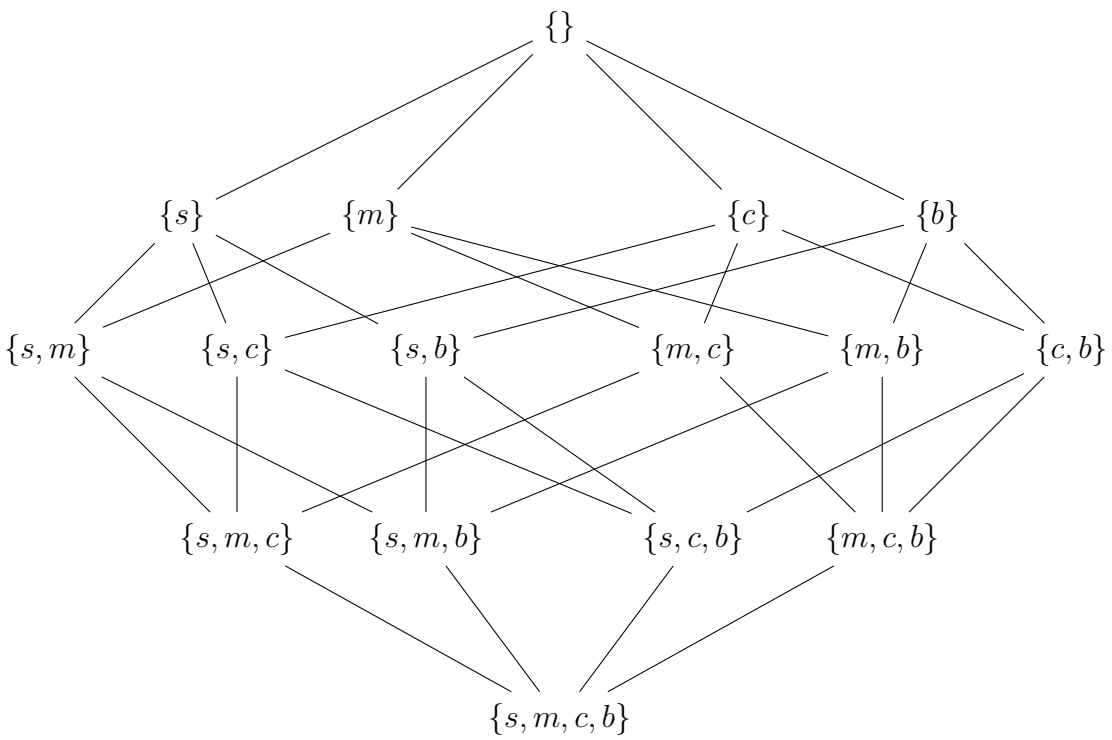


Figure 2.1: Partial order over the itemsets

Algorithm 3: Apriori algorithm

input : D, a dataset of transactions; λ, the minimum support threshold
output: The set of frequent itemsets

```
1 L_1 = find_frequent_1_itemsets(D);
2 for (k = 2; L_{k-1} ≠ ∅; k++) do
3     C_k = apriori_gen(L_{k-1}); //generate candidates
4     forall transaction T ∈ D do
5         C_T = subset(C_k, T); //get subsets of T that are candidates
6         forall candidates c ∈ C_T do
7             c.count++;
8     L_k = {c ∈ C_k | c.count ≥ λ};
9 return L = ∪_k L_k;
```

transaction is build. Next, in line 8, the counts of each itemset $A \in C_T$ is incremented by one. This means that one more transaction contains the A $k-$itemset.

In step 8 the counts of all itemsets $A \in C_k$ and the minimum support threshold λ are used to select all frequent $k-$itemsets, the L_k set.

In the last step the algorithm returns all the frequent itemsets found in each level.

The function $apriori_gen(L_{k-1})$, line 3, generates the candidates $k-$itemsets by combining a joint procedure with a pruning strategy that is grounded in the apriori property.

- In the join step, the set L_{k-1} is joined with itself to generate candidate $k-$itemsets, the C_k set. Each candidate $k-$itemset A is generated from two $(k-1)-$itemsets, $l_1, l_2 \in L_{k-1}$. Considering that each itemset is sorted using the lexicographic order, and that $l(i)$ represents the i^{th} item of the itemset l, we join the two $(k-1)-$itemsets $l_1 = l_1(1), l_1(2), \ldots, l_1(k-2), l_1(k-1)$ and $l_2 = l_2(1), l_2(2), \ldots, l_2(k-2), l_2(k-1)$ if $l_1(1) = l_2(1), l_1(2) = l_2(2), \ldots, l_1(k-2) = l_2(k-2)$ and $l_1(k-1) < l_2(k-1)$. This later condition ensures that no duplicates are generated. The resulting candidate $k-$itemset is the itemset $l_1(1), l_1(2), \ldots, l_1(k-2), l_1(k-1), l_2(k-1)$.
- In the pruning phase of this function, all candidate $k-$itemsets that contain an infrequent $(k-1)-$itemset are removed from the candidate set. This pruning is implemented using a hash tree of frequent itemsets.

As said above, once we get all frequent itemsets we can generate association rules. For each frequent itemset l, generate all the non-empty subsets of l. Then, for each subset a, generate the rule $a \Rightarrow (l-a)$ and keep the rules with confidence equal or higher than a specified minimum confidence value. The rules obtained this way are called association rules.

A more efficient algorithm that eliminates the expensive candidate generation procedure and the multiple database scans required by the Apriori algorithm is the FP-growth [HPY00] algorithm. This algorithm uses a divide-and-conquer strategy. First the algorithm represents the frequent items available in the database into a tree structure, called FP-tree. Then the algorithm extracts frequent itemsets directly from the FP-tree.

In [SA10] Silva and Antunes introduce the Star FP-growth algorithm to find frequent itemsets available in star schema databases [GMUW09]. This work is grounded in

the FP-growth [HPY00] algorithm. In the first phase the Star FP-growth algorithm constructs a FP-Tree for each dimension table. Then, all the trees are merged into a super tree. Last, the algorithm runs the FP-growth over the super tree to output all frequent itemsets. In [SA12] these same authors extend the Star FP-growth algorithm to deal with stream data.

2.6.1.2 Finding First-order Logic Patterns

Dehaspe et al. [DT99] present WARMR, an algorithm that discovers frequent DATALOG patterns in DATALOG databases using a levelwise strategy that is inspired in the APRIORI algorithm. The WARMR algorithm can be seen as an upgrade of the Apriori algorithm to work with FOL representations. In WARMR the user must specify what to count using a key predicate. Typically, the target table is used as the key predicate. The frequency of each query/pattern is the number of individuals, records in the key table, for which there are query bindings in the database. To search for frequent queries the algorithm uses the levelwise strategy and is guided by a declarative language bias, mode and type declarations, and background knowledge specified by a user. In the query generation phase the algorithm uses θ-subsumption to define an order and equivalence relation across queries. This way the algorithm prunes infrequent and redundant queries. Queries that are θ-specializations of infrequent queries and queries that are θ-equivalent to previous generated clauses are removed.

An optimized version of WARMR is presented in [NK03]. The problem with WARMR is that it explores time information by employing aggregation techniques. Thus, WARMR may fail to find and use relevant event sequences, especially short-term fluctuations.

2.6.2 Frequent Sequence Discovery

In this section we introduce concepts and algorithms developed to discover time-ordered patterns from propositional and FOL datasets.

2.6.2.1 Finding Propositional Patterns

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items and e an event such that $e \subseteq I$. A *sequence* is an ordered list of events $e_1 e_2 \ldots e_m$ where each $e_i \subseteq I$. Given two sequences $\alpha =$

$a_1a_2 \ldots a_k$ and $\beta = b_1b_2 \ldots b_t$, sequence α is called a *subsequence* of β if there exists integers $1 \leq j_1 < j_2 < \ldots < j_l \leq t$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \ldots, a_k \subseteq b_{j_l}$. A sequence database is a set of tuples $< sid, \alpha >$ where sid is the sequence identification and α is a sequence. The *count* of a sequence α in a database of sequences D, denoted $count(\alpha, D)$, is the number of examples in D that contain the α subsequence. The *support* of a sequence α is the ratio between $count(\alpha, D)$ and the number of sequences in D. We denote support of a sequence as $support(\alpha, D)$. Given a sequence database D and a minimum support value λ, the problem of sequence mining is to find all subsequences in D having a support value equal or higher than the user-defined value, the λ value. Each one of the obtained sequences is also known as a frequent sequence or a sequential pattern.

When searching for all sequential patterns in a sequence database we face one of the major problems of sequence miners, these algorithms find a huge number of redundant and non-interesting patterns that are hard to analyze. To reduce this problem we can run other sequence miners [YHA03, GRS02] or use post-processing strategies to find a subset of patterns, e.g., that can find *closed sequential patterns* or *maximal sequential patterns*.

Definition 15. (Closed Sequential Pattern) *A frequent sequence α is a closed sequential pattern if there exists no proper supersequence β having the same support value as α.*

Definition 16. (Maximal Sequential Pattern) *A frequent sequence α is maximal if it is not a subsequence of any other frequent sequence. The set of maximal sequential patterns is a subset of all closed sequential patterns.*

Moreover, in a classification problem, some sequential patterns have low discriminative power and could be eliminated. Thus, we introduce the concept of *Discriminative Sequential Patterns* to select inter-class discriminative sequential patterns.

Definition 17. (Discriminative Sequential Patterns) *Consider two sequence database partitions D_1 and D_2, and a support threshold λ. We define the set of discriminative sequential patterns using the xor ($\dot{\vee}$) operator, $S_{disc} = \{\alpha \mid support(\alpha, D_1) \geq \lambda \ \dot{\vee} \ support(\alpha, D_2) \geq \lambda\}$.*

Such discriminative patterns, and others like the emergent and the contrasting patterns, represent valuable information that can be extracted from a sequence database and that could be useful at theory learning time [NLW09]. To do so, we introduce the concept of *Sequence Relation* and *Enlarged Database*.

Definition 18. (Sequence Relation and Enlarged Database) *Consider a multi-relational database* $\mathbf{r}$, *a sequence database* D *encoded from* $\mathbf{r}$, *where each sequence id equals the primary key of* $\mathbf{r}$ *target table. Also consider the set of sequential patterns* S *obtained from solving the sequential mining problem. We define the sequence relation,* $\mathbf{r}_{sr}$, *to be the set of tuples* $(sid, \alpha_1^B, \alpha_2^B, \ldots, \alpha_n^B)$ *where sid is the sequence id in* D *and* α_i^B *is a binary attribute whose value is obtained according to the projection of the sequential pattern* α_i *in the sequence database. The enlarged database is the database resulting from the union of the multi-relational database and the sequence relation. Formally we define the enlarged database as being* $\mathbf{E_r} = \mathbf{r} \cup \mathbf{r_{sr}}$.

As the algorithms we introduce in this thesis explore timed data present in multi-relational databases, mainly sequence data, here we define sequence encoding, a strategy to translate time-series data into a sequence database.

Definition 19. (Sequence encoding) *Given a multi-relational database* $\mathbf{r}$, *having fact tables that register time events. We define sequence encoding as a procedure that for all, or a subset, of database fact tables,* $\mathbf{r_i} \in \mathbf{r}$, *translates all, or a subset, of attributes* $\mathbf{r_i.A_j}$ *into a database of sequences.*

There exists a wide range of algorithms that can explore sequential data efficiently. To the best of our knowledge, [AS95] introduced the sequential data mining problem. In [SA96] Agrawal and Srikant introduce the GSP algorithm, an algorithm that generalizes the original sequential pattern mining problem and that is also inspired in the search procedure of the well-known APRIORI algorithm [AS94]. GSP uses a candidate-generation strategy to find all frequent sequences, and uses a lattice to generate all candidate sequences. The algorithm starts by finding all frequent 1-items that constitute the lattice base. Then, in an iterative process, at each lattice level, frequent sequences found in previous levels are combined to build new frequent sequence candidates. Notice that at each lattice level l only the frequent subsequences of level $l-1$ are used to generate new candidates. We observe that GSP has limitations when dealing with large datasets, as candidate generation may require multiple database consults, and that in real problems it often finds a huge number of patterns.

Several approaches have been proposed to address the above mentioned issues. Different from GSP, that uses horizontal layout format, the SPADE [Zak01] algorithm uses a vertical layout format where each sequence in the lattice is associated with the *idlist*. This *idlist* is a list of all examples containing the candidate sequence, the list contains a set of pairs where each pair consists of both the sequence id (*sid*) and an event identifier (typically the event time). To search for frequent patterns, SPADE also uses

a candidate-generation strategy. To compute the support of each candidate pattern of level l, the *idlist*'s of sequences from level $l-1$ are joined using a *temporal join*. The support of each candidate pattern is the number of distinct *sid* in the candidate *idlist*. Like GSP, SPADE uses the apriori property to prune the search space.

One of the most efficient and well known algorithms that can find frequent sequences is PrefixSpan [PHMA+01]. This algorithm uses the efficient pattern-growth [HPY00] strategy to find the complete set of frequent sequences. Like GSP, the algorithm starts by finding all frequent itemsets. Then, for each one of these frequent items (the prefix) PrefixSpan partitions the current database into *prefix projections*. Each projection database has all the sequences containing the given prefix. This procedure proceeds recursively until all frequent sequences, according to a user-defined support threshold, are found. PrefixSpan's search strategy explores a tree having a prefix projection database at the nodes. This way the algorithm can easily handle large databases but, like GSP, in real problems the algorithm finds a huge number of uninteresting sequence patterns.

A similar difficulty has been observed in most of the existing sequence miners and some methodologies were developed to solve this issue. Algorithms such as CloSpan [YHA03] return a set of closed sequential patterns. SPIRIT [GRS02] constrains the search space using regular expressions. cSPADE [Zak00] extends SPADE algorithm to find a constrained set of sequential patterns. By using cSPADE we can find, among other, sequential patterns predictive of one or more classes. [LZO98] claims that using highly predictive sequences as input to a propositional classifier we can get accuracy gains that range between 10% and 50%.

All these sequence miners were designed to explore propositional data. However, there exists rich structured data representations that require other approaches. Therefore, there is a need to develop methodologies to explore temporal relational databases that include valuable logico-relational information and temporal information recorded in a subset of tables.

2.6.2.2 Finding First-order Logic Patterns

In this section we introduce the SeqLog FOL representation formalism. Different from Datalog, SeqLog represents sequences of logical atoms rather than relations [LR04]. In the following lines we review some fundamental concepts in SeqLog. We use a set of Prolog facts extracted from a discretized version of the database presented in Table 2.1

(Section 2.1) to illustrate some concepts.

In SeqLog, a *term* t_i can be a constant, a variable or an n-ary functor f applied to n terms. An *atom* is a predicate symbol $\mathrm{p}(t_1, \ldots, t_n)$, of arity n, where each t_i is a term.

In SeqLog, a *sequence* is an ordered set, possibly empty, of logical atoms separated by operators: $l_1\ op_1\ l_2\ op_2\ l_3\ \ldots\ l_{n-1}\ op_{n-1}\ l_n$. There are two types of order operators that can be used, the operator $\rhd$ indicates direct succession relation and the operator $<$ indicates a transitive closure succession relation. To represent input data only the $\rhd$ operator is used. For instance the sequence $q = urinalysis(1, A, plt, high) < urinalysis(1, B, ttp, normal) \rhd blood_analysis(1, C, high, normal)$ is a SeqLog sequence. This SeqLog sequence says that patient number one made a urine analysis where the *plt* parameter has a high value. Then, after an undetermined period, and a set of examinations, the patient repeated the urine examination and get a normal value for the ttp parameter. Next, and just after the urine examination where he gets the ttp parameter equal to normal value, patient number one made a blood examination and get a high level of red blood cells and a normal level of white blood cells.

In a SeqLog sequence, q, the head of the sequence, $head(q)$, is the maximal prefix of q that does not contain the $<$ operator. The part of the sequence that appears after the first $<$ operator is the tail of the sequence, $tail(q)$. If we consider the sequence q above we get $head(q) = urinalysis(1, A, plt, high)$ and $tail(q) = urinalysis(1, B, ttp, normal) \rhd blood_analysis(1, C, high, normal)$.

With these concepts we can now define subsumption in SeqLog, a key concept used to search and select the most promising sequential patterns.

We initially define *s-subsumption* for *simple sequences* that do not include the $<$ operator.

A simple sequence $u = l_1 \rhd l_2 \rhd \ldots \rhd l_n$ *s-subsumes* a sequence $r = s_1\ op_1\ s_2\ op_2\ s_3\ \ldots\ s_{m-1}\ op_{m-1}\ s_m$, if and only if there exists a substitution θ such that $(l_1 \rhd l_2 \rhd \ldots \rhd l_n)\theta$ is a subsequence of $s_i \rhd s_{i+1} \rhd s_{i+2} \rhd \ldots \rhd s_{i+n-2} \rhd s_{i+n-1}$ for some i.

A sequence $u = l_1\ op_1\ l_2\ op_2\ l_3\ \ldots\ l_{n-1}\ op_{n-1}\ l_n$ *subsumes* the sequence $r = s_1\ op_1\ s_2\ op_2\ s_3\ \ldots\ s_{m-1}\ op_{m-1}\ s_m$, if and only if there exists a substitution θ and an integer i such that $head(u)\theta$ *s-subsumes* $s_1\ op_1\ s_2\ op_2\ s_3\ \ldots\ s_i\ op_i\ s_i$ and $tail(u)\theta$ *subsumes* $s_{i+1} \rhd \ldots \rhd s_m$.

For instance, the pattern q above is subsumed by the pattern

$$u = urinalysis(1, A, plt, X) < urinalysis(1, B, ttp, Y),$$

as there exists a substitution $\theta = \{X/high, Y/normal\}$ such that $urinalysis(1, A, plt, X)\theta$ is a subsequence of $urinalysis(1, A, plt, high)$ and $urinalysis(1, B, ttp, Y)\theta$ is a subsequence of $urinalysis(1, B, ttp, normal) \rhd blood_analysis(1, C, high, normal)$. For simplicity we will write $u \preceq q$ to say that u *subsumes* q. Informally, we can say that u is more general than q or that q is more specific than u.

The *support* of a logical sequence q in a sequence database D is the percentage of sequences in D that are subsumed by q. Formally, we can write $supp(q, D) = |\{e \in D | B \cup e \models q\}|/|D|$, where the $\models$ symbol represents the logic entailment in the SeqLog calculus. For brevity sake we omit here the definition of entailment and the fix point semantics of SeqLog. In [LR04] we find the definition of these two concepts.

Refinement operators generate more specialized patterns and are fundamental for the mining process [Rae08]. SeqLog has an *optimal refinement operator*, ρ, that specializes each sequence pattern $\rho(s) \subseteq \{s' | s \preceq s'\}$, and has an associated refinement level vector [LR04]. This operator refines a sequence using one of four possible refinements:

- *Lengthening* adds a new atom to the current pattern s;
- *Promotion* promotes a $<$ operator to a $\rhd$ operator;
- *Univar* unifies two variables occurring in the pattern;
- *Instantiation* replaces a variable by a constant.

In the following we will write l, p, u and i to identify, respectively, the number of *Lengthening*, *Promotion*, *Univar* and *Instantiation* refinement operations used to generate a sequence pattern.

Each refinement operation originates an updated refinement vector. For instance, if pattern s has an associated (l, p, u, i) vector, once we generate a s' pattern using a *lengthening* refinement we get the updated vector $(l + 1, p, u, i)$ for pattern s'; if we generate a s'' from s using *promotion* refinement we get the vector $(l, p + 1, u, i)$. This strategy is used to generate all candidates and associated vectors. Thus, when searching the lattice using a levelwise strategy we use the vector associated with each pattern to define the current level of the search. Moreover, the L^1 norm is used to convert the vector into an integer number $v(s) = l + p + u + i$. For instance, pattern $< urinalysis(A, B, C, high), v(1, 0, 0, 1) >$ is in level $2 = 1 + 0 + 0 + 1$ and pattern $< urinalysis(A, B, C, high) \rhd urinalysis(D, E, F, G), v(2, 0, 0, 1) >$ is in level $3 = 2 + 0 + 0 + 1$.

Given a sequence database D, a language bias and a minimum support value λ, the problem of *sequence mining* is to find all subsequences q having $sup(q, D) \geq \lambda$. Each one of the obtained sequences is also known as a frequent (logical) sequence.

Like in itemset mining task (Section 2.6.1), the search for frequent sequence patterns typically generates a huge number of redundant and non-interesting findings. To reduce this issue we can use post-processing selection filters or constrain our search to find *closed sequential patterns* or *maximal sequential patterns*. A frequent sequence q is a *closed sequential pattern* if there exists no proper supersequence s having the same support as q. A frequent sequence q is a *maximal sequential pattern* if it is not a subsequence of any other frequent sequence. The set of maximal sequential patterns is a subset of all closed sequential patterns.

Last, a *contrasting frequent sequence* is a sequence q whose $\sup(q,\ D_{pos}) \geq \lambda_{pos}$ and $\sup(q,\ D_{neg}) \leq \lambda_{neg}$. Here D_{pos} and D_{neg} are two class partitions of D and λ_{pos} and λ_{neg} are two user-defined support thresholds. λ_{pos} defines the minimum support that the contrasting pattern q can have in the positive partition, the D_{pos} partition. λ_{neg} defines the maximum support that the contrasting pattern q can have in the negative partition D_{neg}.

In [NLW09] we can find a discussion about the utility of working with subsets of patterns. The authors put the focus on contrasting patterns, emerging patterns and subgroup discovering.

In [LR04], beyond presenting the SeqLog representation formalism, the authors introduce the MineSeqLog logical sequence miner to find logical sequences in a database of logical atoms. This algorithm combines ideas from version spaces [Bay98, KRH01] and levelwise search to find all frequent and contrasting patterns, including the maximal ones. The algorithm suffers from some important performance drawbacks that are well-known in the majority of the ILP-based algorithms [BS03]. When fed with large databases, that accumulate data over long periods of time, the algorithm is unable to find long patterns. Moreover, even when dealing with small databases, the algorithm can easily be unable to find contrasting patterns. This last issue results from two design problems. First, when used to find contrasting sequential patterns, the algorithm needs to find all sequential patterns available in the two class partitions (in a binary classification problem). Therefore, it is necessary to run the sequence miner with small support values to find patterns in the negative partition of the data. Second, to find a large pattern, maybe a high confidence one, the algorithm needs to find the complete set of small size patterns.

Moreover, it is well known that the enormous representational power of logic systems is obtained at the cost of scalability. In [BS03] it is observed that the ILP algorithms may not be effective to explore the large search spaces induced by sequence databases.

Amo and Furtado [dAF07] propose an extension of the regular expressions formalism to FOL, including an algorithm that finds FOL sequential patterns satisfying a regular expression constraint.

In [EMBF09] the Datalog formalism is extended to find FOL sequences that can include time, space and other dimensions and orderings between events. Based in the extended formalism the authors present a levelwise algorithm that can find all maximal FOL patterns.

In [MBFE11] the sequence miner presented in [EMBF09] is used to find sequential patterns. Then, using a wrapper selection strategy, an optimal subset of patterns is selected and used to learn a propositional classification model.

Notice that non-ILP approaches have also been used to find frequent patterns in multi-relational data. One popular approach is graph mining. Matsuda et al. [MHM+99] upgrade the original GBI(Graph-based Induction) to deal with direct or undirected graphs, with or without loops, colored or uncolored nodes and links. GBI tries to minimize the graph size using stepwise pair expansion (pairwise chunking), at each step a frequent pattern is compressed to a node. The algorithm efficiency complexity is linear with the graph size resulting from using greedy search without backtracking. GBI only extracts meaningful typical patterns of a certain size. The algorithm is tested on Promoter DNA sequence, carcinogenicity and mutagenecity data being the prediction of extracted classification rules evaluated by 10-fold cross validation. In [MMYW02] Matsuda et al. analyze the Hepatitis dataset with an improved GBI version. This new version introduces: beam search, achieving a quasi-complete search; a discriminative function, for finding typical patterns after chunking and canonical labelling for enumerating identical patterns.

Chapter 3

Constructive Induction with Ensemble Feature Selection

In this chapter we present a framework developed in the very beginning of this thesis to explore constructive induction technique and propositional pattern mining methods. The framework we present here combines a propositional association rule miner with a propositional classifier to learn a classification model. This work uses constructive induction technique and an ensemble of heuristics to select and include new and interesting patterns in the learning process.

3.1 Introduction

In this section we present the *rankCI* framework, a three-phase classification architecture that extends the original description language and learns a classification model using both primitive and new descriptors. This framework follows the Cascade Generalization architecture [GB00] and sequentially combines the Apriori association rule miner [AS94] and the C4.5 decision tree induction algorithm [Qui93]. In the first phase, the Apriori algorithm finds all frequent itemsets available in each class partition of the training data. Then, in the second phase, the most interesting and relevant itemsets are select and used to build new attributes. These new attributes are then mapped into the primitive dataset. With this strategy we obtain an enlarged dataset that includes the primitive descriptors and the new attributes that can be seen as intermediate concepts. In the third phase, we run the C4.5 algorithm to learn a decision tree model.

An important issue with this approach is the filtering strategy that we introduce to select the most interesting itemsets. In [WZ05] Webb and Zhang observe that *measures will often not actually directly capture the true value of a rule, as this will often depend on many factors that are difficult to formalize and incorporate in a computational process.*

Thus, we introduce a *xor-filter* to select inter-class exclusive itemsets and an ensemble of ranking heuristics to select the top most interesting itemsets. Our ensemble selection combines a heterogeneous set of heuristics and selects features in the rank order space. For each new itemset/feature candidate, we compute and use the degree of correlation/association between the itemset and the class of the example. We select for all heuristics in the ensemble an equal number of new itemsets, the top ones, according to a user-defined threshold. Our framework applies to nominal features and non-homogeneous spaces. Empirical experiments using fourteen datasets, with nominal attributes, prove the effectiveness of our methodology. The stability and performance of the results are related with the diversity of the heuristics.

The work described in this chapter presents the following contributions:

- Use a heterogeneous set of heuristics to select features in rank-order spaces. This way we can include *different views* in the feature selection problem.
- We introduce a xor filter to select inter-class exclusive features.

In the next section we present a detailed description of the *rankCI* framework. An illustrative example is used to clarify every step of the framework. In Section 3.3 we experimentally evaluate this architecture. In the last section we conclude and point to new research directions.

3.2 The rankCI Framework

As said above, the *rankCI* framework (presented in Algorithm 4) is based on the Cascade Generalization framework [GB00]. The framework has three main phases. In the first phase, we run the Apriori algorithm to find the frequent itemsets. Secondly, we select the most interesting itemsets using two new feature selection techniques: we introduce the *xor-filter* to select inter-class exclusive itemsets; and we introduce a *rank ensemble* that uses an ensemble of heuristics to sort features and generate a rank-order space. Next, the rank induced by each heuristic is used to select the

Algorithm 4: RankCI pseudo-code

input : a dataset, D; a set of heuristics, $Heurset$; two thresholds, δ and λ;
output: a decision tree model

```
1  (D_1, ..., D_c) ← Split (D);
2  for i in L_c do
3      T_i ← Apriori(D_i,δ);
4  T ← xor-filter(T_1, ..., T_c);
5  for heur in Heurset do
6      for itemset in T do
7          HeurValue[itemset, heur] ← heur(itemset);
8  HeurValueRank ← rank(HeurValue);
9  TSelect ← ExclTopItemsets(HeurValueRank, λ);
10 Dx ← projection(TSelect, D);
11 return(C4.5(Dx));
```

most interesting features, that are then mapped in the original dataset. This mapping operation generates a new Boolean set of descriptors. In the third phase, we run a classification algorithm to learn a classifier model on the extended dataset. Without loss of generality, in this work we use the C4.5 decision tree algorithm.

3.2.1 Constructive Induction - Phase 1

In this phase our goal is to obtain a heterogeneous set of new intermediate level concepts that give insight into the classification problem being solved.

First, we split (line 1) the original dataset, D, using the label of each example. Therefore, we get c partitions of the dataset D. Then (lines 2-3) we run the *Apriori* algorithm to find frequent itemsets in each dataset partition. The L_c set contains all class labels available in D. The minimum support threshold of the *Apriori* algorithm is the user-defined parameter δ. We arrive at the end of this phase with a large set of frequent itemsets, most of them redundant and non-interesting.

3.2.2 Feature Selection - Phase 2

To select discriminative and high interesting itemsets we run two filters. First, we introduce (line 4) the *xor-filter*. With this filter we obtain a set of exclusive itemsets $T = \{t|\forall_{i,j \in L_c}(t \in T_i \wedge t \notin T_{j \neq i})\}$. These exclusive itemsets are discriminative and frequent in one and only one partition.

Second, we combine a set of heuristics ($Heurset$) to select the most interesting itemsets. With each heuristic (lines 5-7) we measure the degree of association/correlation between each itemset and a class label. We compute the degree of association/ heuristic value using a contingency table where columns represent the itemsets and rows represent the class labels. Then, the itemsets are sorted (line 8) using the values of each heuristic. This sort operation originates a rank-order space where each dimension is associated with a heuristic. The top ranked itemsets in each dimension are the most interesting ones. Next, we introduce a filter that selects (line 9), for each heuristic, a user-defined number of itemsets. The number of top patterns to select is defined by the λ parameter. After these two filter procedures we obtain a set, $TSelect$, of discriminative and non-redundant itemsets. We get a minimum of λ itemsets, if all heuristics rank equal, or a maximum that equals λ times the number of heuristics in the ensemble, if the rankings are mutually exclusive.

To extend the description language we map (line 10) the selected itemsets into the primitive dataset. This way we obtain an enlarged dataset, Dx. For each itemset we add a new binary attribute that is the conjunction of all items in the itemset. The value of the new attribute is computed in the following way. If all items of a given itemset/attribute match a subset of primitive items that describe a given example, the new attribute takes value one, otherwise the attribute takes value zero.

3.2.3 Decision Tree Induction - Phase 3

After the previous two phases we get an enlarged dataset that includes both the primitive and the new descriptors. In this phase, we take this enlarged dataset as input to a classification algorithm and learn a classification model. In the *rankCI* framework we run the C4.5 algorithm as the default algorithm to learn the classification model.

By using this extended description language we can learn a classification model that can gain both in terms of accuracy and readability.

Table 3.1: Four instances of the *yellow-small+adult-stretch* dataset

Id	Color	Size	Act	Age	Class
1	yellow	small	stretch	adult	true
2	yellow	large	dip	child	false
3	purple	small	stretch	adult	true
4	purple	small	stretch	child	false

Table 3.2: Total order over the itemsets using the value of each heuristic

Rank	P-Corr	Gain	Cosine	Gini	Leverage	Laplace
1	26	26	1	1	26	1
2	27	27	2	2	27	2
3	1	28	3	3	1	3
4	2	29	4	4	2	4
5	3	30	5	5	3	5
6	4	31	6	6	4	6
7	28	32	7	7	28	7

3.2.4 Illustrative Example

We present an example that illustrates each step of our framework. We select six heterogeneous heuristics based on their properties [LMVL08, GH06]. Our rank ensemble members are Pearson-correlation, Information Gain, Cosine, Gini, Leverage and Laplace.

Among the fourteen datasets we use in the experiments (see Section 3.3), we select the *yellow-small+adult-stretch* dataset[1]. This artificial dataset represents balloon conditions according to the logic expression: *if ((color=yellow and size=small) or (age=adult and act=stretch)) then Inflated=true*. In Table 3.1 we present part of the example dataset. This dataset has four attributes and a binary class.

For this experiment we define the *Apriori* minimum support threshold equal to five percent, $\delta = 5\%$, and select the top five itemsets, $\lambda = 5$, in the rank-order induced by each heuristic. The results presented in this example were obtained in the first iteration of the 10-fold cross validation procedure described in Section 3.3.

[1]http://archive.ics.uci.edu/ml/datasets/Balloons

Table 3.3: Database enlarged with ten new attributes

ID	Color	Size	Act	Age	N26	N27	N1	N2	N3	N4	N28	N29	N30	N5	Class
1	yellow	small	stretch	adult	1	1	0	1	0	1	1	1	1	0	true
2	yellow	large	dip	child	0	0	0	0	1	1	1	0	0	0	false
3	purple	small	stretch	adult	0	1	0	0	1	1	0	1	0	0	true
4	purple	small	stretch	child	0	0	1	0	1	1	1	0	0	1	false

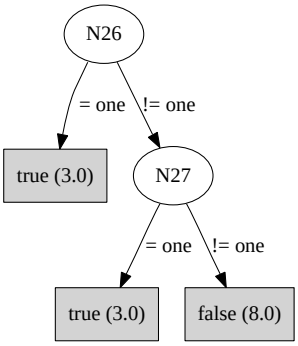


Figure 3.1: *Rank ensemble* selection

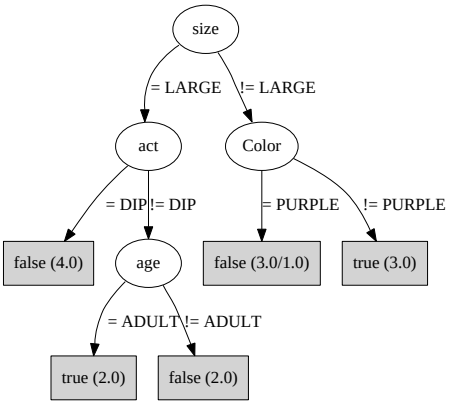


Figure 3.2: Stand-alone C4.5

Firstly, the Apriori algorithm finds 102 frequent itemsets, 55 in the positive class partition and 47 in the negative class partition. Second, we apply the xor-filter and get 25 exclusive positive itemsets and 17 exclusive negative itemsets. This filter reduces the number of candidate frequent itemsets by 59%, from 102 to 42. Then, we compute the degree of association/correlation between each one of the new attributes and each class. We do this for all itemsets and heuristics. Next, we introduce a total order on the itemsets using the value of each metric. In Table 3.2 we present the result of this sort operation. In this table we can see the top seven attributes according to each metric. For instance, Pearson Correlation was used to sort the 42 itemsets and ranks the itemset number 26 as the most interesting itemset, the itemset number 27 as the second most interesting, the itemset number 1 as the third most interesting, and so on.

This way, and using the order introduced by each heuristic, we select 30 itemsets (6×5), the top five of the rank space induced by each one of the six heuristics. From these 30 itemsets we eliminate repeated itemsets and get 10 unique elements. Then we map these 10 frequent itemsets into the primitive dataset producing an enlarged dataset. In Table 3.3 the original database is enlarged with ten new binary attributes.

Third, we take the enlarged dataset as input to the C4.5 decision tree algorithm and get a decision tree classification model (see Figure 3.1). For comparison purposes we also built a decision tree (see Figure 3.2) using only the primitive dataset as input to stand-alone C4.5 algorithm.

We can see in Figure 3.1 that the proposed methodology discovered and used two new intermediate concepts, two new attributes from the positive class: $N26 = (Color = Yellow \wedge size = Small)$ and $N27 = (act = Stretch \wedge age = Adult)$. This way we get a more accurate classification model and a more compact tree. In this iteration we get 0% misclassification error and a tree size equal to two. These results compare favorable with the baseline. With the baseline classifier we get 50% misclassification error and a tree with four nodes.

3.3 Experimental Evaluation

In this section, we describe the configuration of the experiments and present and discuss the obtained results in a set of benchmark datasets.

3.3.1 Configuration of the Experiments

In this section we present the datasets, the tools, and the procedures used to evaluate the *rankCI* architecture.

3.3.1.1 Datasets and Development Environment

We tested our algorithm using fourteen datasets: *adult+stretch, adult-stretch, kr-vs-kp, monks-1, monks-2, monks-3, promoters, tictactoe, yellow-small+adult-stretch, yellow-small, DNF0, DNF1, DNF2 and DNF3*. The first ten taken from the UCI machine learning repository[2], the other four datasets are artificial datasets, each one representing a Boolean function (see Appendix A). All datasets have nominal attributes, there are no missing values and all tasks are binary classification problems (Remark: Our framework also supports multivalued nominal classes).

We implemented our framework using the R language [R D11] and WEKA, a collection of machine learning algorithms [WF99]. To find frequent itemsets we use the Apriori

[2]http://www.ics.uci.edu/ mlearn/MLRepository.html

Table 3.4: Description of the datasets and the number of itemsets found by the Apriori algorithm and selected using the *xor-filter*

Datasets	Examples	Primitive Attr.	*Apriori*	xor-filter
adult+stretch	16	4	92	52
adult-stretch	16	4	92	52
dnf0	32	5	284	138
dnf1	256	8	1340	1305
dnf2	1024	10	8742	4284
dnf3	1024	10	8477	4105
kr-vs-kp	3196	36	1999	1539
monks1	432	6	727	305
monks2	432	6	504	200
monks3	432	6	684	418
promoters	106	57	2611	2611
tictactoe	958	9	2052	1104
yellow-stretch	16	4	102	42
yellow-small	16	4	92	52

algorithm available in the *arules* package of R. To induce the decision tree we use the J48 algorithm, a C4.5 clone available in the WEKA collection.

3.3.1.2 Organization of the Experiments

In these experiments, our goal is to evaluate the performance of the *rankCI* framework and present a sensibility analysis of the parameters. Here we present the results that we obtained by running a rank ensemble that consists of six heuristics. The heuristics that we select to participate in the ensemble are: Pearson Correlation, Information Gain, Cosine, Gini, Leverage and Laplace. Furthermore, we compare our algorithm against the stand-alone C4.5 algorithm that uses only the primitive descriptors as input. In addition, we evaluate our algorithm using a rank ensemble of size one. I.e., we built six instances of our framework where only one of the six heuristics is used to sort and select features.

To run our algorithm we must define two thresholds, the δ and the λ parameters. Using a validation set, we define the Apriori's minimum support threshold equal to

Table 3.5: Misclassification error using the six heuristics *ensemble*

	Number of top patterns to keep (λ)						
	5	10	15	20	25	30	Stand-alone C4.5
adult+stretch	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)
adult-stretch	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)
dnf0	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	9.17 (14.9)
dnf1	0.00 (0.0)*	0.00 (0.0)*	0.00 (0.0)*	0.00 (0.0)*	0.00 (0.0)*	0.00 (0.0)*	16.02 (6.4)
dnf2	1.08 (1.2)*	1.47 (1.2)*	1.17 (1.1)*	1.17 (1.1)*	1.56 (1.4)*	1.66 (1.3)*	5.18 (1.5)
dnf3	0.00 (0.0)*	0.39 (0.5)	0.00 (0.0)*	0.00 (0.0)*	0.00 (0.0)*	0.00 (0.0)*	0.78 (0.4)
kr-vs-kp	0.41 (0.6)	0.41 (0.6)	0.41 (0.6)	0.44 (0.5)	0.44 (0.5)	0.44 (0.5)	0.53 (0.5)
monks1	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)
monks2	5.33 (4.0)	3.93 (3.1)	1.84 (1.8)	1.84 (1.8)	4.62 (1.9)	4.62 (2.9)	3.00 (3.1)
monks3	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)
promoters	17.09 (9.0)	16.00 (9.8)	10.36 (9.2)	12.27 (11.0)	12.27 (11.0)	15.27 (10.3)	18.27 (13.7)
tictactoe	1.25 (0.4)*	1.67 (0.5)*	1.67 (0.5)*	1.67 (0.5)*	1.67 (0.5)*	1.67 (0.5)*	6.05 (3.8)
yellow-stretch	0.00 (0.0)*	0.00 (0.0)*	0.00 (0.0)*	0.00 (0.0)*	0.00 (0.0)*	0.00 (0.0)*	35.00 (33.7)
yellow-small	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)
Arithmetic Mean	1.797	1.704	1.104	1.243	1.469	1.69	6.714
Geometric Mean	0.449	0.427	0.413	0.42	0.458	0.467	1
Average Ranks	3.607	3.964	3.179	3.5	3.929	4.179	5.643
Av. Error Ratio	0.741	0.658	0.638	0.65	0.721	0.734	1
Number wins	8	8	9	9	8	8	0
Number losses	1	1	0	0	1	1	0
Wilcoxon p-value	0.027	0.020	0.004	0.004	0.020	0.020	1,000

Table 3.6: Theory size: number of tree nodes using the six heuristics *ensemble*

	Number of top patterns to keep (λ)						
	5	10	15	20	25	30	Stand-alone C4.5
Arithmetic Mean	22.414	19.143	18.643	19.1	19.129	17.557	34.314
Geometric Mean	0.675	0.578	0.571	0.581	0.568	0.556	1
Average ranks	4.643	3.643	3.25	3.607	3.464	3.036	6.357
Complexity Ratio	0.718	0.635	0.626	0.633	0.633	0.612	1
Number of Wins	11	12	12	12	12	13	0
Number of Losses	2	1	1	1	1	0	0
Wilcoxon p-value	0.03	0.007	0	0	0.007	0	1

five, $\delta = 5\%$. We use this parameter value for all datasets except for the *kr-vs-kp* and the *promoters* datasets. The different support values that we use for *kr-vs-kp*, 80%, and *promoters*, 20%, are due to the number of dataset examples and primitive attributes, otherwise we would end up with an intractable number of itemsets. The other user-defined threshold, λ, can take one of six user-defined values. We can select, for each heuristic, the top ranked 5, 10, 15, 20, 25 or 30 itemsets. By evaluating our algorithm for each one of these levels our goal is to study the relation between the number of new attributes we map and the performance of our framework. We want to analyze the stability and robustness of our framework.

To evaluate our algorithm we run, for each dataset, a stratified 10-fold cross validation procedure. Using these results we analyze the performance of our framework according to the misclassification error and the complexity of the generated models (number of nodes in the generated tree). In addition, we compute the misclassification error and the associated standard deviation, the average mean, the geometric mean, the average error ratio, the number of wins and losses and the Wilcoxon hypothesis test p-value. We compute Wilcoxon signed ranked pair-test to evaluate how significantly our algorithms differ from the reference C4.5 algorithm. The null hypothesis of the test is that the median of the differences is zero.

3.3.2 Results

Table 3.4 describes each one of the fourteen datasets. We present the number of instances and the number of primitive attributes. Moreover, we present the number of patterns found by the Apriori algorithm and the number of exclusive inter-class

Table 3.7: Misclassification error using a single heuristic

	Number of top patterns to keep (λ)											
	5						30					
Heuristic	P-Corr	Gain	Cosine	Gini	Leverage	Laplace	P-Corr	Gain	Cosine	Gini	Leverage	Laplace
adult+stretch	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)
adult-stretch	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)
dnf0	0.00 (0.0)	0.00 (0.0)	28.33 (18.9)	0.00 (0.0)	28.33 (18.9)	28.33 (18.9)	0.00 (0.0)	0.00 (0.0)	35.00 (11.7)	0.00 (0.0)	35.00 (11.7)	35.00 (11.7)
dnf1	0.40 (1.3)	0.00 (0.0)	13.95 (8.5)	1.17 (1.9)	13.95 (8.5)	13.95 (8.5)	0.00 (0.0)	0.00 (0.0)	5.45 (4.9)	0.00 (0.0)	5.45 (4.9)	5.08 (4.6)
dnf2	2.34 (1.8)	2.25 (1.8)	2.54 (1.3)	2.34 (1.8)	2.54 (1.3)	1.56 (1.4)	1.66 (1.3)	1.66 (1.3)	4.10 (2.3)	1.66 (1.3)	4.10 (2.3)	3.61 (2.1)
dnf3	0.00 (0.0)	0.39 (0.5)	0.39 (0.5)	0.39 (0.5)	0.39 (0.5)	0.39 (0.5)	0.00 (0.0)	0.39 (0.5)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)
kr-vs-kp	0.41 (0.6)	0.53 (0.5)	0.53 (0.5)	0.53 (0.5)	0.56 (0.5)	0.56 (0.5)	0.41 (0.6)	0.56 (0.5)	0.56 (0.5)	0.56 (0.5)	0.56 (0.5)	0.56 (0.5)
monks1	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)
monks2	1.39 (1.6)	1.39 (1.6)	7.86 (5.1)	1.39 (1.6)	7.86 (5.1)	7.86 (5.1)	3.70 (2.7)	3.70 (2.7)	11.07 (6.3)	3.70 (2.7)	11.07 (6.3)	11.07 (6.3)
monks3	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)
promoters	17.09 (9.0)	16.18 (8.3)	18.18 (14.9)	17.09 (9.0)	18.18 (14.9)	18.18 (14.9)	14.27 (10.3)	14.09 (10.9)	18.18 (14.9)	14.27 (10.3)	17.18 (15.8)	18.18 (14.9)
tictactoe	1.56 (1.5)	4.80 (1.8)	4.07 (2.1)	3.24 (1.7)	4.90 (3.2)	3.24 (1.7)	0.00 (0.0)	0.73 (0.9)	1.67 (0.5)	1.67 (0.5)	1.67 (0.5)	1.67 (0.5)
yellow-stretch	0.00 (0.0)	0.00 (0.0)	40.00 (39.4)	0.00 (0.0)	40.00 (39.4)	40.00 (39.4)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	40.00 (39.4)	40.00 (39.4)
yellow-small	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)	0.00 (0.0)
Arithmetic Mean	1.657	1.825	8.276	1.868	8.337	8.149	1.431	1.51	5.43	1.562	8.216	8.226
Geometric Mean	0.411	0.45	1.02	0.445	1.038	0.973	0.439	0.41	0.795	0.466	1.031	1.02
Av. Error Ratio	0.666	0.663	1.17	0.65	1.184	1.151	0.696	0.658	1.234	0.725	1.31	1.305
Wilcoxon p-value	0.004	0.008	0.742	0.008	0.82	0.82	0.012	0.027	0.426	0.02	1	1

Table 3.8: Theory size: number of tree nodes using a single heuristic

	Number of top patterns to keep (λ)											
	5						30					
Heuristic	P-Corr	Gain	Cosine	Gini	Leverage	Laplace	P-Corr	Gain	Cosine	Gini	Leverage	Laplace
Arithmetic Mean	22.086	26.757	28.671	25.457	30.414	29.643	17.543	17.286	26.657	17.714	26.643	25.786
Geometric Mean	0.706	0.756	0.891	0.75	0.922	0.902	0.605	0.551	0.778	0.558	0.78	0.763
Av. Error Ratio	0.752	0.788	0.93	0.778	0.952	0.942	0.715	0.611	0.855	0.617	0.856	0.838
Wilcoxon p-value	0.019	0.022	0.08	0.022	0.08	0.354	0.013	0	0.097	0	0.073	0.097

itemsets selected using the xor-filter.

In Table 3.5 and Table 3.6 we present the misclassification error (and the standard deviation) and the complexity, respectively, of the classification models that we obtained using an ensemble of six heuristics. In Table 3.5 we present the mean misclassification error and the standard deviation for a range of λ values. For comparison purposes we also present, in last column, the results obtained with the stand-alone C4.5 algorithm and using only the primitive dataset as input. In the last seven rows of this table we present a set of statistics that summarize the results. In Table 3.6 we present statistics that summarize the complexity of the trees learned with the ensemble method.

As mentioned above we also run our framework using only one of the six heuristic that participates in the ensemble to sort the itemsets. In Table 3.7 we present the misclassification error that we obtain by sorting the examples with a single heuristic. In Table 3.8 we summarize the complexity of the models. In these two tables we present relevant statistics values for two values of the λ parameter, $\lambda = 5$ and $\lambda = 30$.

3.3.3 Analysis of the Results

If we analyze the results presented in tables 3.5 and 3.7 we observe that the six heuristic ensemble generate the most accurate models that we learn using the *rankCI* framework. The six heuristic ensemble compares favorable against the single heuristic selection and the stand-alone models that were learned using only primitive attributes. Furthermore, we can say that when one of the heuristics used in the ensemble performs better than the stand-alone C4.5 algorithm, the six heuristic ensemble gains with the inclusion of this heuristic and obtains better results than the single heuristic selection method.

In addition, we observed that our framework is sensitive to the number of new attributes, typically we get better results when we select a number of itemsets ranging

between 15 and 20. Indeed, this range of values is associated with the high accurate classification models and with compact models. If we map a large number (> 25) of itemsets we get high compact classification models at the cost of low accuracy values. This behavior means that the prediction model is overfitting the data. Moreover, we can say that the C4.5 algorithm used at level 3 gains with the ensemble diversity. There are few cases where the single heuristic models beat the six heuristic ensemble. This is especially clear when the heuristic in the single rank is a member of the ensemble. In comparison with the stand-alone model and the single heuristic configuration, we get high accurate models with small standard deviation values for the ensemble configuration, with a high diversity of heuristics. This is a well known behavior of ensembles [VD04].

Concerning the analysis of the framework configuration that uses a single heuristic, we obtained high accurate and high compact models when we use Pearson correlation and measures from the field of information theory to sort the itemsets.

When we analyze the p-value of the Wilcoxon hypothesis test, we observe that our algorithm differ from the stand-alone C4.5 algorithm. Remember that our algorithm uses C4.5 at phase 3 to induce a decision tree model.

Beyond generating high accurate and compact models, our framework finds new concepts that give insights on the concept being learned. Typically, these concepts are new attributes that appear in the inner nodes of the tree. One example of such new descriptors is presented in the illustrative example of Section 3.2.4.

3.4 Conclusions

In this work we introduced the *rankCI* framework, a general three-phase architecture that combines sequentially an association rule miner and a propositional classifier. In the first phase the framework extends the primitive description language using conjunctions of items that are found by the Apriori algorithm. In the second phase we use a xor filter and a heterogeneous ensemble used to select the most interesting features. Last, we take the enlarged dataset as input to the C4.5 algorithm and learn a decision tree classification model.

We experimentally evaluate our algorithm using a set of benchmark datasets and a heterogeneous set of heuristics. The *rankCI* framework generates high accurate and more compact models than the baseline, the stand-alone C4.5 algorithm. Moreover,

we show that the ensemble methodology can gain with the diversity of the ensemble members. In comparison with a configuration that uses a single heuristic to sort the itemsets, the ensemble strategy obtains significant wins. Furthermore, we observe that the tree models generated by our framework include new attributes at the inner nodes, especially in positions near the root of the tree. In some cases, all the inner nodes of the generated models use new attributes. These new attributes are intermediate level concepts that can highlight important information hidden in the original dataset. These new attributes are selected using different metrics/views. By combining constructive induction with the ensemble strategy we reduce fragmentation and myopia, two well-known problems of decision tree models [PH90, JB03].

In the next chapter we extend some ideas of this work to multi-relational databases. We explore first-order logic representations and methodologies.

Chapter 4

Propositionalization with First-order Logic Features

In this chapter we present the *RUSE-WARMR* framework that we developed to explore patterns available in multi-relational databases and learn a propositional classification model. Moreover, the goal of this work is to find patterns without joining the tables in a pre-processing phase. Thus, in this work we run a first-order logic association rule miner to find patterns and use constructive induction and logic-based propositionalization techniques to learn a classification model.

4.1 Introduction

Different from the work presented in Chapter 3, the work presented in this chapter was developed to explore complex structured data, that may stem from a number of different and related sources. More precisely, we design a framework to explore information available in multi-relational databases and learn a classification model.

In this chapter we introduce the *RUSE-WARMR* framework, an architecture designed to explore multi-relational patterns available in relational databases. In this work we run WARMR [DT99], an ILP pattern miner algorithm, to find first-order logic (FOL) patterns available in multi-relational databases. Next, a filter strategy is used to eliminate redundant and class unrelated patterns. Then, the top-k most discriminative patterns found are mapped into the target table. Last, the top-k patterns and the primitive descriptors are used to learn a theory describing the target concept.

Previous work in this research area, like RSD [ZL06], SAYU [DBD+05], nFOIL [LKR07], kFOIL [LPRF06], suggest that an effective approach to explore multi-relational data is *propositionalization* (see Section 2.4 in Chapter 2). By using this technique we can gain with the huge amount of research in feature-based learning. The effectiveness of this technique depends both upon the algorithm we use to generate the patterns and upon the propositional classifier.

The majority of the new contributions in this area focuses in proposing new FOL pattern miners and strategies to select the most interesting patterns [ZL06, NLW09]. The main issue with this technique is the number of patterns found by the FOL miner, that grows exponentially when we use FOL powerful representation languages. On the one hand, if all these patterns are fed into the propositional learner, the learning task may become intractable and become very vulnerable to overfitting. On the other hand, if we constrain too much the number of patterns, we risk to miss important patterns that may be extremely useful to the learning task. In Chapter 3 and work [FG07] we study feature selection techniques.

The *RUSE-WARMR* algorithm is motivated by the excellent results of ILP based pattern miners, such as WARMR, namely in terms of efficiency and robustness. In contrast to systems such as nFOIL or SAYU, where the pattern mining task is tightly integrated with classifier construction, and therefore is heavily dependent on the training examples, rule mining provides an excellent framework for *efficiently* mining the interesting patterns that are relevant on their own and that can scale very effectively on large datasets.

The contributions of the *RUSE-WARMR* methodology are:

- We develop a strategy that finds discriminative patterns by decomposing the pattern mining problem. First the class partitions are mined separately and, then, all sets of patterns found are merged. With this procedure we achieve efficiency and a natural parallelized algorithm.
- We propose a novel method to select discriminative patterns among the set of all the patterns found. We introduce a relaxed version of θ-subsumption to implement this discriminative filter.

In Section 4.2 we introduce some concepts that are needed to design the *RUSE-WARMR* architecture. In Section 4.3 we present a detailed description of our framework. An illustrative example that explains every step of the architecture is also presented. In Section 4.4 we experimentally evaluate the *RUSE-WARMR* framework

and present the obtained results. Finally, in Section 4.5 we conclud stressing the need to develop new methodologies that can explore conveniently temporal databases.

4.2 Preliminaries

In this section we introduce some fundamental concepts that were important to the design of the *RUSE-WARMR* architecture.

When exploring a database, $\mathbf{r}$, to find frequent queries, Q, we define a language bias, L, that restricts the size of the search space and guides the search. We then aim to find a theory of frequent queries $Q = \{q \in L : sup(q, \mathbf{r}) > \lambda\}$, where $sup(q, \mathbf{r})$ is the number of times a query q succeeds in database $\mathbf{r}$, and λ is a user-defined support threshold. A query is said to succeed in a database if there are bindings, in the database, for all variables in the query, such that the query holds true.

A special case is defined when exploring a database partition. In this case we define $Q_k = \{q \in L : sup(q, \mathbf{r}_k) > \lambda\}$, where $\mathbf{r}_k$ is a database partition.

When searching for frequent patterns within and across database partitions there is a need to introduce an equivalence relation. One of the most used query equivalence relations is Plotkin's [Plo70] θ-subsumption.

Definition 1: Let q_1 and q_2 be two queries. If there exists a substitution θ such that $q_1\theta \subset q_2$, we say that q_1 θ-subsumes q_2.

Definition 2: Let q_1 and q_2 be two queries. We say that q_1 is θ-equivalent to q_2 if and only if q_1 θ-subsumes q_2 and q_2 θ-subsumes q_1.

In our work we need to select *discriminative* patterns. Notice that we will usually learn queries from different class partitions of the database. Queries of interest are queries that have significant different support values in different database partitions. The next definition formalizes this idea.

We define two queries q_1 and q_2 to be *agreeing queries*, and write $q_1 \cong_\theta q_2$, if

1. q_1 is θ-subsumption equivalent to q_2, and,
2. $|sup(q_1, \mathbf{r}_i) - sup(q_2, \mathbf{r}_j)| < \lambda$

where $sup(q_1, \mathbf{r}_i)$ and $sup(q_2, \mathbf{r}_j)$ are the support values of the queries in different dababase partitions, $i \neq j$, and λ is a user-defined threshold value. In our system

Algorithm 5: *RUSE-WARMR* pseudo-code

input : a dataset, $\mathbf{r}$; two thresholds, λ and δ
output: a decision tree model

```
1 r_1, ..., r_c ← partition (r);
2 for k = 1 to c do
3     Q_k ← WARMR(r_k,λ);
4 Q_disc ← discriminate(Q_1, ..., Q_c,λ);
5 Q_sel ← chi-square(Q_disc,δ);
6 r_enltarget ← Propositionalization(Q_sel, r);
7 return(C4.5(r_enltarget));
```

we use this λ value also to define the minimum support threshold of the WARMR algorithm.

This definition was motivated by the support value used to search for frequent patterns. We consider that clauses with a support value higher than a given threshold are interesting, and we further argue that equivalent queries whose threshold differ at least by a pre-specified amount can be interesting and discriminative.

4.3 The RUSE-WARMR Framework

In this section we present the *RUSE-WARMR* framework. In the end of this section we present an illustrative example the explains every step of our architecture.

Our framework proceeds in three steps. First, we use the WARMR algorithm to find frequent patterns, a set of Datalog queries that succeed with respect to a sufficient number of examples. Second, we select discriminative and interesting patterns using two filters: a discriminative filter and a chi-square filter. Then we project the selected patterns in the target table, obtaining an enlarged table with new Boolean attributes. Third, using this enlarged table as input to a propositional classifier, here we use Quinlan's C4.5 algorithm [Qui93], we induce a decision tree classification model.

Next we present a detailed description of the *RUSE-WARMR* architecture (see the pseudo-code presented in Algorithm 5). The framework has four input parameters: a database with logic atoms, $\mathbf{r}$; two thresholds, the λ minimum support threshold, that defines the minimum query frequency and is used to evaluate relaxed θ-subsumption equivalence; and the δ value, that specifies the confidence level of the chi-square

hypothesis test used to select features.

4.3.1 Finding First Order Descriptors - Phase 1

In phase 1 we explore the relational database to find frequent queries. In this phase we search for relational knowledge that summarizes information available in a multi-relational database of Prolog facts.

As we can see in Algorithm 5, in line 1 we use the labels of all classes to split the dataset into a set of class partitions $\mathbf{r}_k$, with $k = 1, \ldots c$. Then, we use domain knowledge to specify the mode and type declarations of the WARMR algorithm. The background knowledge can be included using a set of Prolog procedures. Then, in line 3, we run the WARMR algorithm to find the set of frequent queries Q_k in each class partition $\mathbf{r}_k$. I.e., we run WARMR to find queries q having $sup(q, \mathbf{r}_k) > \lambda$. At the end of this phase we typically end up with a large number of redundant and non-discriminative patterns.

4.3.2 Feature Selection and Propositionalization - Phase 2

In phase 2 we use two filters to select the most interesting patterns. The first filter removes non-discriminative and redundant patterns using the θ-subsumption operator. The second filter uses the chi-square statistic and the confidence level to select class related patterns. Next we map the selected patterns into the target table and obtain an enlarged target table. This step is known as *propositionalization*.

In line 4 of Algorithm 5 we remove agreeing queries using relaxed θ-subsumption equivalence, and get a set

$$Q_{disc} = \{q_i | \forall_{q_i \in Q_i} \ \forall_{q_k \in Q_{k \neq i}} \ q_i \not\cong_\theta q_k\}$$

of discriminative patterns.

Although we run this filter, we still get class unrelated queries among the Q_{disc} set. Thus, in line 5 we employ a selection filter to remove this type of patterns. This filter runs a chi-square hypothesis test using a user-defined confidence level, the input parameter δ.

Last, in line 6 we map each one of the most interesting patterns into the target table. For each one of these patterns we add one new Boolean attribute to the target table,

obtaining an enlarged table. To do so, for each example in the enlarged target table we set the value of the new attribute/query using the query bindings. If there exists a binding of the primitive attributes for which the query succeeds, the attribute takes value one for this example. Otherwise the attribute takes value zero.

4.3.3 Learning a Classification Model - Phase 3

In phase 3 we learn a classification model. In this work we run the C4.5 algorithm but we can run any other propositional classifier.

When we arrive at this phase we have an enlarged target table extended with new descriptors. Using this enlarged table we aim to learn (line 7) a model that besides being accurate should be easy to understand and give insights into the underlying concepts. To do this, we build a decision tree model using the C4.5 algorithm with the enlarged target table as input. The generated model can include any of the attributes available in the enlarged target table.

4.3.4 Illustrative Example

To highlight each step of our framework we proceed with the illustrative example already presented in [FGC08]. In this example we define a simple bias and run *RUSE-WARMR* to learn a classification model that discriminates between patients having hepatitis B subtype and hepatitis C subtype.

As a result of pre-processing the Hepatitis dataset, we get a dataset with a total of 503 labeled patients: 206 examples labeled as B subtype and 297 labeled as C subtype. Next, we split the training set in two partitions, one for hepatitis subtype B, the $\mathbf{r}_1$, and the other for hepatitis subtype C, the $\mathbf{r}_2$.

We then run WARMR in each class partition. In this example, we define the minimum support threshold to be $\lambda = 20\%$. This way the WARMR algorithm finds 107 frequent queries in train partition $\mathbf{r}_1$ and 100 frequent queries in train partition $\mathbf{r}_2$.

Thus, we get a total of 207 frequent queries. Among these queries there are low discriminative and redundant queries which we aim to eliminate. We eliminate these agreeing patterns using the discriminative filter and obtained 63 queries only.

Among these 63 queries some remain that are uncorrelated with any of the classes. To eliminate them we use the chi-square hypothesis test. We set the confidence level

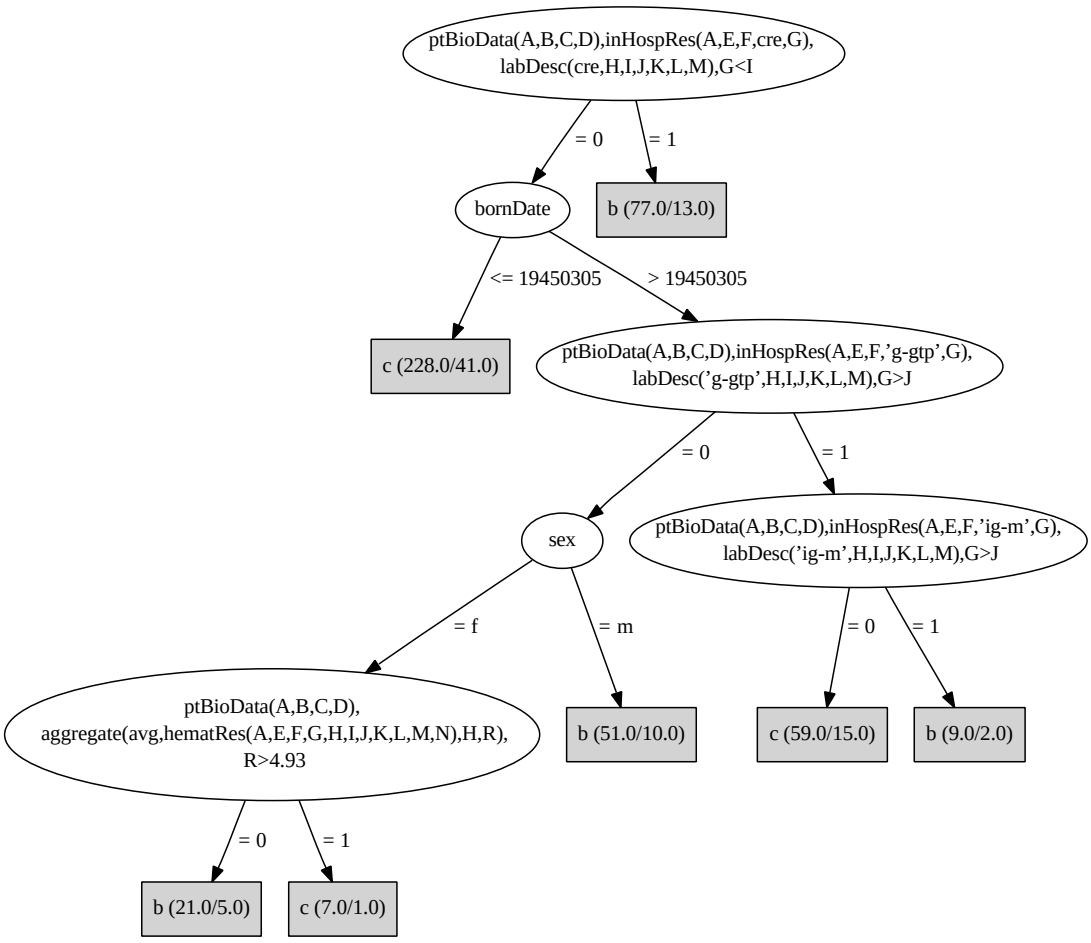


Figure 4.1: *RUSE-WARMR* induced decision tree for the Hepatitis subtype problem

Table 4.1: Enlarged target table

MID	subtype	sex	bornDate	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}	q_{11}
1	c	m	19520109	1	1	1	0	0	0	0	0	0	0	0
2	b	f	19750123	1	1	1	0	1	0	0	0	1	0	0

to 5%, $\delta = 5\%$, and selected 11 patterns. Then, we map all these patterns into new attributes on the target table.

In Table 4.1 we present two records of the enlarged target table. In this table we see four primitive attributes, the id, the label, the sex, the birth-date of each patient, and 11 new attributes, one for each query pattern. To illustrate these steps we present two of the generated queries, q_1 and q_2, and explain the mapping procedure:
q_1: ptBioData(A1,B1,C1,D1), aggregate(avg, hematRes(A1, E1, F1, G1, H1, I1, J1, K1, L1, M1, N1),J1,R1), $R1 > 46.4$
q_2: ptBioData(A2,B2,C2,D2), inHospRes(A2,E2,F2,'g-gtp',G2), labDesc('g-gtp', H2, I2, J2, K2, L2, M2), $G2 > J2$

For each example in the target table, we set the attribute q_1 value equal to 1 if the average of all *J1* parameter values, the *R1* value, is higher than the 46.4 threshold value. Otherwise the attribute takes value zero. The *J1* parameter represents the percentage of *hematocrit* registered in table *hematRes*. The attribute q_2 takes value one if the parameter *g-gtp* value, variable $G2$, in the patient urinalysis examination result is larger than the upper bound, $J2$, of a normal examination, i.e., if $G2 > J2$.

Using this procedure for each one of the eleven queries we get the target table enlarged with eleven new attributes.

Finally, we run the C4.5 propositional learner to induce a decision tree classification model. In Figure 4.1 we present an example of a tree model that we can learn.

4.4 Experimental Evaluation

In this section we run our framework to solve four problems available in the four datasets described in Appendix B, the Hepatitis, the Protein, the Financial and a balanced version of the Financial dataset. First we describe the configuration of the

Table 4.2: Number of patterns found in different phases of the *RUSE-WARMR* framework

			WARMR		Filters		C4.5
Datasets	Support	ChiValue	Part A	Part B	Exclusive	Chi-Square	Tree Size
Hepatitis	0.9	*No-Chi*	1.0 (0.0)	2.0 (0.0)	2.0 (0.0)	2.0 (0.0)	12.8 (2.4)
		10	1.0 (0.0)	2.0 (0.0)	2.0 (0.0)	0.9 (0.3)	9.2 (0.8)
		5	1.0 (0.0)	2.0 (0.0)	2.0 (0.0)	0.8 (0.4)	8.8 (0.9)
		Standard	1.0 (0.0)		–	–	4.6 (0.8)
	0.8	*No-Chi*	2.0 (0.0)	5.8 (0.4)	4.8 (0.4)	4.8 (0.4)	22.0 (4.1)
		10	2.0 (0.0)	5.8 (0.4)	4.8 (0.4)	1.2 (0.4)	9.2 (0.8)
		5	2.0 (0.0)	5.8 (0.4)	4.8 (0.4)	0.8 (0.4)	8.8 (0.9)
		Standard	4.0 (0.0)		–	–	10.4 (4.2)
Financial	0.9	*No-Chi*	67.8 (14.0)	131.7 (43.1)	167.9 (36.6)	167.9 (36.6)	2.0 (0.0)
		10	67.8 (14.0)	131.7 (43.1)	167.9 (36.6)	28.9 (21.1)	2.0 (0.0)
		5	67.8 (14.0)	131.7 (43.1)	167.9 (36.6)	9.9 (11.1)	2.0 (0.0)
		Standard	52.7 (12.7)		–	–	1.0 (0.0)
	0.8	*No-Chi*	746.1 (232.3)	335.1 (40.2)	653.2 (169.9)	653.2 (169.9)	2.0 (0.0)
		10	746.1 (232.3)	335.1 (40.2)	653.2 (169.9)	147.3 (102.6)	2.0 (0.0)
		5	746.1 (232.3)	335.1 (40.2)	653.2 (169.9)	60.6 (48.9)	2.0 (0.0)
		Standard	343.3 (47.7)		–	–	1.0 (0.0)
FinancialB	0.9	*No-Chi*	302.5 (49.5)	131.7 (43.1)	380.4 (53.6)	380.4 (53.6)	6.8 (2.7)
		10	302.5 (49.5)	131.7 (43.1)	380.4 (53.6)	66.2 (36.8)	2.0 (0.0)
		5	302.5 (49.5)	131.7 (43.1)	380.4 (53.6)	36.6 (27.6)	2.0 (0.0)
		Standard	113.9 (44.0)		–	–	1.2 (0.6)
	0.8	*No-Chi*	854.8 (106.2)	746.1 (232.3)	847.5 (104.9)	847.5 (104.9)	19.6 (3.7)
		10	854.8 (106.2)	746.1 (232.3)	847.5 (104.9)	234.3 (92.8)	17.6 (2.3)
		5	854.8 (106.2)	746.1 (232.3)	847.5 (104.9)	166.5 (79.3)	15.2 (1.6)
		Standard	639.4 (72.1)		–	–	2.4 (1.9)
Protein	0.9	*No-Chi*	2.8 (0.0)	0.0 (0.0)	1.8 (0.0)	1.8 (0.0)	1.0 (0.0)
		10	2.8 (0.0)	0.0 (0.0)	1.8 (0.0)	1.8 (0.0)	1.0 (0.0)
		5	2.8 (0.0)	0.0 (0.0)	1.8 (0.0)	1.8 (0.0)	1.0 (0.0)
		Standard	2.8 (0.0)		–	–	1.0 (0.0)
	0.8	*No-Chi*	4.8 (0.5)	0.0 (0.0)	4.8 (0.5)	4.8 (0.5)	1.9 (0.2)
		10	4.8 (0.5)	0.0 (0.0)	4.8 (0.5)	4.1 (0.8)	1.9 (0.2)
		5	4.8 (0.5)	0.0 (0.0)	4.8 (0.5)	3.9 (0.9)	1.9 (0.2)
		Standard	3.5 (0.1)		–	–	1.0 (0.0)

Table 4.3: Generalization accuracy of the *RUSE-WARMR* framework

			Generalization Accuracy	
Datasets	Support	ChiValue	C4.5	SVM
Hepatitis	0.9	*No-Chi*	64.63 (8.20)	62.50 (6.18)
		10	62.47 (5.97)	62.50 (6.25)
		5	62.47 (5.97)	62.90 (6.12)
		Standard	62.70 (6.07)	62.90 (6.44)
	0.8	*No-Chi*	63.42 (6.92)	63.29 (6.09)
		10	62.67 (5.57)	62.70 (6.36)
		5	62.47 (5.97)	62.90 (6.12)
		Standard	64.40 (7.29)	64.50 (6.13)
Financial	0.9	*No-Chi*	86.76 (1.23)	85.92 (3.85)
		10	86.76 (1.23)	86.76 (1.23)
		5	86.76 (1.23)	86.76 (1.23)
		Standard	86.76 (1.23)	85.92 (3.85)
	0.8	*No-Chi*	86.76 (1.23)	81.20 (6.50)
		10	86.76 (1.23)	86.34 (1.60)
		5	86.76 (1.23)	86.76 (1.23)
		Standard	86.76 (1.23)	85.92 (3.85)
FinancialB	0.9	*No-Chi*	54.10 (12.91)	49.40 (17.70)
		10	61.40 (1.90)	46.70 (16.91)
		5	61.40 (1.90)	60.20 (4.05)
		Standard	61.81 (2.20)	60.69 (5.71)
	0.8	*No-Chi*	59.20 (11.35)	54.40 (16.82)
		10	61.80 (17.72)	61.80 (17.72)
		5	61.80 (17.72)	67.00 (20.89)
		Standard	61.81 (2.20)	59.44 (8.83)
Protein	0.9	*No-Chi*	63.33 (1.22)	63.33 (1.22)
		10	63.33 (1.22)	63.33 (1.22)
		5	63.33 (1.22)	63.33 (1.22)
		Standard	63.33 (1.22)	63.33 (1.22)
	0.8	*No-Chi*	63.18 (1.18)	63.23 (1.49)
		10	63.18 (1.18)	63.23 (1.49)
		5	63.18 (1.18)	63.23 (1.49)
		Standard	62.32 (1.26)	62.32 (1.26)

experiments and then we present and analyze the obtained results.

4.4.1 Configuration of the Experiments

As said before, the architecture of our framework is general and can match a wide range of algorithms both to find FOL patterns and to learn a propositional classification model. In these experiments we chose to build two instances of our framework. We present experiments that combine WARMR algorithm with C4.5 algorithm and experiments that combine WARMR algorithm with a linear kernel SVM - Support Vector Machine [Pla98].

We further study the effectiveness of our filter methodology. Here we also present the results that we obtained when we discard lines 4 and 5 of our algorithm (see the pseudo-code of Algorithm 5). In the results we write *No-Chi* to identify the results that we obtained without the chi-square filter. Furthermore, we present results for a configuration that takes the full training set as input (no partitions!) and does not employ filters to prune the WARMR findings. We name this configuration as *Standard*. The results presented with these configurations are identified in the table columns headed with *ChiValue*, a column associated with the δ parameter of the *RUSE-WARMR* architecture.

Moreover, we evaluate our framework for a set of parameters. We set the support value of the WARMR algorithm, the input parameter λ, to be equal to 90% and 80%. Moreover, we set the δ parameter equal to 5% and 10%.

Furthermore, we limit the maximum level of refinement of the WARMR algorithm to 4. Without this limit it would be impossible to search the full solution space.

We use a tenfold stratified cross-validation strategy to evaluate all the configurations we evaluate in this work. For each configuration, we compute the mean and the standard deviation of the generalization accuracy. In the multi-class problem available in the Protein dataset we use the round-robin strategy presented in Section 2.5.3.3 of Chapter 2.

Moreover, we evaluate how significantly the two instances of the *RUSE-WARMR* framework, *WARMR + C4.5* and *WARMR +SVM*, and the *No-chi* configurations differ from a *Standard* approach. Therefore, we compute the Wilcoxon signed ranked pair-test, setting the confidence level to 95%. The null hypothesis of this test is that the median of the differences is zero.

For comparison purposes and to evaluate the contribution of the chi-square filter, we also introduce a slightly modified version of *RUSE-WARMR*. In this modified version we replace the chi-square filter, that uses the confidence level, by a filter that uses the chi-square statistic to sort and select the k most interesting patterns. With this strategy we eliminate the risk of having an extremely large number of patterns selected with a specified confidence level, the δ parameter.

To implemented all these versions we use the ACE [BDD+00] data mining System and WEKA [WF99] collection of data mining algorithms.

4.4.2 Results

In tables 4.2, C.2, C.1 (the last two tables are presented in Appendix C) we present, respectively, the number of patterns, the run-time, in seconds, and the peak memory usage, in kilobytes, of each step of the *RUSE-WARMR* framework.

In Table 4.2 we present in the first column the names of the datasets used. In column 2 we present the support value of the WARMR algorithm, the λ parameter of *RUSE-WARMR*. In column three we present the chi-square statistic threshold, the δ parameter of the framework, and identify the type of experiment. If we run the *RUSE-WARMR* without the chi-filter or if we use the classical/standard framework we write *No-Chi* and *Standard*, respectively. In columns four and five we present the number of patterns found by WARMR in each partition. We write *Part A* to identify the results obtained in the minority class partition and write *Part B* to identify the results obtained in the majority class partition. As said above, we present the average number of the patterns found in the multi-class problem. Remember that we use round-robin strategy (see Section 2.5.3.3) to solve the multi-class problem. In columns six and seven we present the number of patterns that we get after removing agreeing patterns and applying the chi-square filter. In the last column we present the size (number of nodes) of the trees that we get when we learn a tree model using the *RUSE-WARMR* framework. In Table C.2 (available in Appendix C) we use the same layout of Table 4.2 and present the run-time of each step of the *RUSE-WARMR* algorithm. Moreover, we present the time needed to run either the C4.5 algorithm or the SVM algorithm to learn a classification model in the enlarged target table.

In Table 4.3 we present the generalization accuracy of two instances of the *RUSE-WARMR* framework. We present the generalization accuracy using the C4.5 algorithm and the SVM algorithm in the last step of the *RUSE-WARMR* architecture.

In Table C.3 (see Appendix C) we show the results that we obtain if the chi-square filter is replaced by the filter that, first, uses the chi-square statistic to sort the patterns and then select the top-k most interesting patterns. We present results when we select and map the top ten and the top twenty most interesting patterns.

4.4.3 Analysis of the Results

The statistics presented in Table 4.3 show that our results are somewhat better than the base accuracy. Among these, the most important gains were obtained in the balanced version of the Financial dataset. In this dataset we get a generalization accuracy of 67% using the SVM algorithm in the last step of the *RUSE-WARMR* architecture. In comparison, by running the standard algorithm we get a generalization accuracy of 59.44%. In all other problems we get small gains ranging between 1% and 3% in terms of generalization accuracy.

In Table 4.2 we notice that we get a large number of patterns when we run the *RUSE-WARMR* algorithm than when we run the classical approach. This behavior was expected as in *RUSE-WARMR* framework we run the WARMR algorithm in both partitions and we can find patterns that will be impossible to obtain if we use the full dataset as input to the WARMR algorithm. We can see that by running the *RUSE-WARMR* we can get an equal or larger number of rules in one partition than the number of patterns that we found by taking the full dataset as input to the WARMR algorithm. This means that we can miss to find important class patterns if we do not run separate instances of the WARMR algorithm. The main issue with running WARMR in the two class partitions is that we get a large number of redundant queries. This is clear when we analyze the contribution of the discriminative filter. In column number six of Table 4.2 we present the number of non-agreeing patterns selected by the discriminative filter. Typically, we get a larger number of discriminative patterns than the number of patterns found by the standard algorithm. This is an important finding, with our approach we can find a large number of FOL patterns by running WARMR, with high support values, in each class partition. Most of the patterns obtained with this new strategy can only be found with the standard approach if we run the WARMR algorithm with low support values. Moreover, and depending on the characteristics of the dataset and on the computational resources, it can be impossible to find the same patterns using the standard approach.

Moreover, in columns seven and eight of Table 4.2 we observe that the number of patterns selected by the chi-square filter and the size of the tree induced, using the

C4.5 algorithm, decrease with the significant level, the δ parameter of the *RUSE-WARMR* architecture. One interesting result is the size of the induced tree using the standard approach and the Financial datasets as input. We observe that we feed the C4.5 algorithm with many patterns and yet we get small size decision trees. This phenomena can be related with the low discriminative patterns found when we use the full datasets as input to the WARMR algorithm. The C4.5 algorithm does not find any pattern that reduces the entropy and, in many cases, learns the classifier with a single leaf, the majority class.

Regarding the peak memory usage and the run-time presented in tables C.1 and C.2 (see Appendix C), we observe that running WARMR in any partition requires approximately equal or less memory than running WARMR using the full dataset as input. This is an interesting result that can be valuable if we run each WARMR instance in parallel. This issue should be explored in the future as running the WARMR algorithm is the most expensive step of the *RUSE-WARMR* framework and, as said before, a large number of patterns can be found when we run WARMR to explore each class partition than when we run WARMR to explore the full dataset. When we analyze the running times of the WARMR algorithm we observe that in the hepatitis problem the standard approach needs always more time than the time needed by WARMR to explore any partition of the dataset. In the other problems the maximum run-time measured in the two partitions is higher than the run-time required by the standard approach. This is especially clear when we explore the Financial datasets. In these problems we have many long patterns available in each partition, making the search last longer times. The run-time and memory usage of the other steps of the *RUSE-WARMR* framework are typically less expensive than the execution of the WARMR algorithm and both the run-time and memory usage decrease with the number of patterns available in each step. Furthermore, the run-time required to learn a classification model is always an order of magnitude less than the run-time needed to run the WARMR algorithm.

The results presented in Table C.3 (see Appendix C) show that mapping a large number of patterns can generate low accurate classifiers. In general we get worse results when we map twenty patterns than when we map ten patterns. Moreover, these results are almost the same as those presented in Table 4.3. This behavior can be explained by the small number of interesting patterns that remain after applying the chi-square test of the *RUSE-WARMR* framework.

4.5 Conclusions

In this chapter we presented a classification architecture that uses constructive induction and FOL pattern miners to explore complex structured data. In contrast to the classical approach that joins tables in a pre-processing step, our framework uses the FOL pattern miner to explore the logical relational structure of multi-relational databases and then uses propositionalization technique to use efficient propositional learners.

We developed a three phase architecture that combines sequentially a FOL pattern miner, two feature selection filters and a propositional classifier. In the first phase we search for frequent FOL patterns using the WARMR algorithm. In the second phase we introduce two filters to remove non-discriminative and select interesting class correlated patterns. To do so we introduce a new equivalence relation and we use the chi-square statistic test. Then, we take the selected patterns and use the propositionalization technique to extend the target table with new Boolean attributes. In the third phase we run a propositional classifier, such as a decision tree learner or a SVM.

In this architecture it is not required to find patterns in the full database. We find frequent patterns available in each class partition and then merge the obtained set of patterns. In the merge process we run a filter to select only discriminative patterns. With this strategy we achieve efficiency and a naturally parallel algorithm.

The models generated by our system are readable and almost all the obtained models have both ILP patterns and primitive attributes from the target table at the inner nodes. An interesting observation is that in some runs all the inner nodes of the induced trees contain only these ILP patterns.

Though our algorithm uses WARMR at phase 1 and C4.5 (or SVM) at phase 3, Wilcoxon significance test prove that our algorithm differs from the standard approach. Remember that the standard approach combines these same algorithms but do not uses any pattern pruning/selection strategy. The obtained results prove that our feature selection strategies remove non-interesting features.

In the next two chapters we extend the framework presented here to explore the temporal relations available in multi-relational databases. More specifically, we will study propositional and FOL algorithms that can find sequences of events. Moreover, we will run highly efficient sequence miners to find longer patterns. We also search for other feature selection methods to select the most interesting patterns.

Chapter 5

Exploring Multi-Relational Temporal Data using Propositional Techniques

In this chapter we present and analyze propositional methodologies to explore temporal information available in multi-relational databases. We introduce the *MuSer* framework that we develop in this thesis. At the core of this system is an encoding technique that translates the temporal information into a propositional sequence of events. By using this strategy we are able to explore the temporal information using a propositional sequence miner. Moreover, in this system we combine feature selection and propositionalization techniques to cast a multi-relational classification problem into a propositional one.

5.1 Introduction

Quite often, multi-relational databases accumulate data over long periods of time. To explore such temporal records we can use aggregation operators (see Chapter 4) or use special purpose techniques that explore the time order of the events. In this work we assume that data stored in a subset of tables are sequences of events that reflect the evolution of a phenomenon of interest. As an example, consider a multi-relational dataset in a medical domain. A patient is subject to a sequence of examinations and a set of measurements, corresponding to the results of different analysis, are taken. For each patient, we get a sequence of measurements over time. Our hypothesis is that the

evolution of these measurements might encode relevant information for the diagnosis of a certain disease. The problem we address here is therefore *how can we explore such information?* In more general terms, can we extract *implicit* multi-relational temporal information that can be used to learn predictive and highly accurate decision models? We believe that both intra-table and inter-table temporal patterns can be used for this purpose. To do so, we develop an architecture that is grounded in the propositionalization methodology [FGC08, ZL06].

We propose *MuSer* (MUlti-relational SEquential patteRn knowledge learning), a framework that discovers sequence patterns available in a subset of relational tables. The algorithm is organized as a pipeline. We start by converting the multi-relational temporal data into what we call a *sequence database*. In this new database the temporal information of each example is represented by a heterogeneous sequence, that can include events registered in one or more tables. In a second phase, we run a sequence miner to find frequent sequences in each database partition. Next, in phase three, we select a subset of sequence patterns. We extract the discriminative frequent ones, e.g. those that appear in only one class, and using a chi-square filter we remove class unrelated sequences. In phase four we map the selected patterns into a binary set of attributes, that indicate the presence or absence of a sequence in a particular example. Last, we induce a classification model using a propositional algorithm.

We believe that this methodology is suitable to explore time related information stored in multi-relational datasets. To evaluate our methodology, we analyze the four multi-relational datasets described in Appendix B. We evaluate the effectiveness of our methodology by measuring the generalization accuracy and the usefulness of the learned classification models. A comparison against the RUSE-WARMR framework [FGC08] is also presented. In Chapter 4 of this thesis we presented the RUSE-WARMR framework. Remember that the RUSE-WARMR system aggregates temporal data using a set of statistics.

The contributions of this work are therefore:

- We developed *MuSer*, a framework that explores heterogeneous sources of time data recorded in a multi-relational database. Our methodology explores temporal records by abstracting patterns over *time*. We do not deal with explicit time, instead we explore the order of the events.
- The *MuSer* architecture searches for sequence patterns in each class partition individually. With this strategy we find a large number of in-class and discriminative patterns.

Algorithm 6: *MuSer* pseudo-code

input : a multi-relational dataset $\mathbf{r}$; three thresholds λ, the support value of the sequence miner and k, the number of top patterns to map

output: a classification model

```
1  Pre-Processing Encoding
2  |  s ← conversion (r);
3
4  s_1, ..., s_m ← partition (s);
5  Finding Frequent Patterns
6  |  for i = 1 to m do
7  |  |  sf_i ← SequenceMiner(s_i,λ);
8
9  Filtering
10 |  S_disc ← discriminate(sf_1, ..., sf_m);
11 |  S_inter ← sort(S_disc,chi-square);
12
13 Mapping
14 |  r_EnlargedTarget ← Mapping(S_inter, k, r_target);
15
16 Classifier Induction
17 |  ClassifierIndution(r_EnlargedTarget);
18
```

In Section 5.2 we present a detailed description of the *MuSer* framework. Next, in Section 5.3 we present and discuss the obtained results. At the end of this chapter we conclude and point to the development of new methodologies that can include valuable domain knowledge.

5.2 The *MuSer* Framework

In this section we present the *MuSer* framework. This new architecture explores temporal and logico-relational information to build descriptive and highly accurate classification models. More precisely, the framework explores sequences of events registered in one or more tables of a multi-relational database. In the first phase we convert the multi-relational data into a sequence database. Then, in a second

Table 5.1: Relational database used to illustrate the encoding technique

Patient Info
(Target Table)

Id	*Sex*	*BornDate*	*Class*
1	m	19520109	**c**
2	f	19750123	**b**

Blood Analysis

Id	*Date*	*RBC*	*WBC*
1	19750102	high	normal
1	19780203	high	high
2	19770107	high	low

Urinalysis

Id	*Date*	*Exam*	*Value*
1	19741201	plt	high
1	19750102	alb	normal
1	19750102	ttp	normal
1	19760204	alb	normal
2	19800403	alb	normal

phase, we run a sequence miner to find all sequential patterns in each class partition. Next, considering the huge number of findings, we introduce two filters to select the discriminative and class correlated patterns. In the fourth phase, the most interesting frequent sequences are projected into the target table obtaining an enlarged target table. In the last phase, we learn a classification model by taking the enlarged target table as input to a propositional classifier.

To best explain the *MuSer* architecture we follow the pseudo-code presented in Algorithm 6.

5.2.1 Encoding Methodology - Phase 0

Here we introduce the strategy that converts the multi-relational temporal data into a convenient sequence database. We use the relational database presented in Table 5.1 to highlight the encoding methodology. This illustrative database is inspired in the PKDD 2005 Hepatitis dataset and has three tables registering the follow-up of two patients. One of these tables is the target table, named *Patient Info*. This table registers, for each patient, a masked ID, the sex, the date of birth and the class label. The other two tables record temporal blood analysis and urinalysis examinations. The *Blood Analysis* table records, for each patient, the examination date and the patient's RBC and WBC parameters. Table *Urinalysis* records for each patient the value of three parameters: *alb*, *plt* and *ttp*.

First, for each example available in the multi-relational target table we find all the

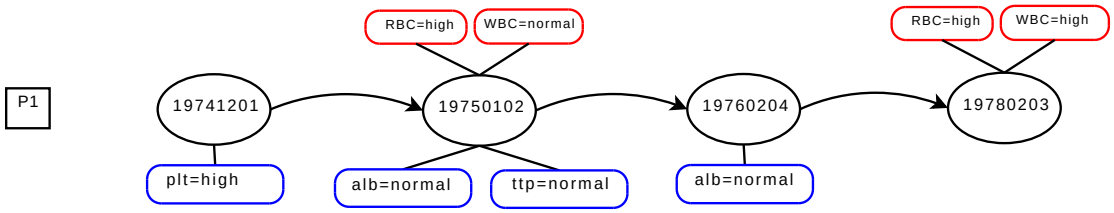


Figure 5.1: Patient one event sequence

Table 5.2: Sequence Database

ID	Sequence	Class
1	(9) (3 5 7 8) (7) (3 6)	c
2	(3 4) (7)	b

associated relations that record temporal information. Next, we sort all such records by using the order of occurrence of the events. After this process, we obtain a chronological sequence of multiple events for each example. In Figure 5.1 we present the event sequence of patient number one. In this case, the sequence includes a sequence of blood analysis and urinalysis events.

Next, we build an attribute-value sequence for each example. As an example, for patient one we get $(plt = high)(RBC = high, WBC = normal, alb = normal, ttp = normal)(alb = normal)(RBC = high, WBC = high)$. In this new sequence each itemset corresponds to all records registered at a given date/time. Then, we define a one-to-one encoding map $f : Attributes \times Values \longrightarrow \mathbb{N}$. This mapping associates a *unique* number to each attribute-value pair. The definition of this map is done accordingly to the type of attributes in each database relation. For each discrete attribute we find the range of attribute values and map each attribute-value pair onto a unique integer value. This map is a three element tuple having the attribute name, the attribute value and the unique integer number that identifies the attribute-value. When dealing with continuous attributes, first we apply a discretization strategy and then proceed as for discrete attributes. Discretization of the attributes will be problem and domain specific.

Table 5.2 shows the sequence database that we obtain when we apply this mapping technique to our example database. In this table each sequence corresponds to an example in the target table and each subsequence corresponds to all events that occur at the same time. In this example we define the one-to-one map $f(rbc, low) =$

1, $f(rbc, normal) = 2$, $f(rbc, high) = 3$, $f(wbc, low) = 4$, $f(wbc, normal) = 5$, $f(wbc, high) = 6$, $f(alb, normal) = 7$, $f(ttp, normal) = 8$, $f(plt, high) = 9$. Using this map the sequence of events registered for patient one is encoded into the sequence (9) (3 5 7 8) (7) (3 6).

5.2.2 Finding Frequent Sequence Patterns - Phase 1

Having the sequence database built in the pre-processing phase, we now need to use a methodology to find new and interesting descriptors. Thus, we run a sequence miner in each class partition of the sequence database (line 7 of Algorithm 6). This way we get all in-class frequent patterns, the sf_i set. Each one of these i sets contains patterns that have at least a minimum support equal to λ, a user-defined threshold. Regarding the nature of the sequence miner, the type and the dimension of the problem being addressed we usually get an overall huge number of patterns. Among these patterns there are highly discriminative and class correlated patterns that we must keep and uninteresting and redundant patterns that must be eliminated.

5.2.3 Feature Selection - Phase 2

To select discriminative sequences we introduce the *discriminative filter*(line 10), that is inspired in the successful approach introduced in Chapter 4. Using this filter we select a set of discriminative sequences, the S_{disc} set, a set of patterns that were found in only one of the partitions. This pruning operation is accomplished with a simple matching operation that keeps non-agreeing patterns.

Consider that when we run the sequence miner in the sequence database of partition **b** and we found the two sequences: S_{b1} : (9) (5 8) (3 6) and S_{b2} :(10) (7). Moreover, by running the same algorithm in the sequence database of partition **c** we obtain sequences S_{c1} : (3) (7) and S_{c2} : (10) (7). After applying the discriminative filter we eliminate S_{b2} and S_{c2} and keep sequences S_1 : (9) (5 8) (3 6) found in class partition **b** and S_2 : (3) (7) from class partition **c**.

After eliminating the non-discriminative patterns some class uncorrelated patterns remain. To eliminate some of these patterns we use the *Chi-square filter* that computes the chi-square statistic and uses the obtained value to sort the patterns.

By applying sequentially these two filters we get a sorted set of patterns. To include the descriptive power of these patterns in the learning process we need to map each

Table 5.3: Enlarged target table

MID	subtype	sex	bornDate	S_1	S_2
1	c	m	19520109	1	1
2	b	f	19750123	0	1

of the new sequences/features into the target table.

5.2.4 Mapping Features - Phase 3

In this section we explain how we map each one of the k (a user-defined parameter) most interesting patterns. For each pattern we add a new Boolean attribute to the target table. The value of each new attribute is computed using the following rule. If the sequence pattern associated with the new attribute is a subsequence of the example sequence, the new attribute takes value one. Otherwise the attribute takes value zero. This way we get an enlarged target table that includes the primitive attributes and the new Boolean attributes.

In Table 5.3 we present the augmented target table of the example database presented in Table 5.1. In this table we have two new attributes, corresponding to the two sequence patterns, S_1 and S_2, introduced above. For each patient and sequence pattern the value of the associated attribute is computed according to the existence of the sequence pattern in the patient related records. This way each patient is described by the primitive attributes in the target table, the sex, the date of birth and the two sequence patterns.

5.2.5 Classifier Induction - Phase 4

In the last step (line 17) we learn a classification model. We take the enlarged target table as input to a propositional classifier and learn a classification model. This model can include either primitive or new descriptors. In figures 5.2 and 5.3 we present two examples of decision tree models that we can learn.

5.3 Experimental Evaluation

In this section we experimentally evaluate the *MuSer* framework. First we describe the configuration of the experiments. Then we present and discuss the obtained results.

5.3.1 Configuration of the Experiments

In this section we present the configuration of the experiments that we define to evaluate the performance of the *MuSer* framework. We run *MuSer* to solve the three problems described in Appendix B.

The *MuSer* framework is a general architecture that can combine a large set of propositional sequence miners and classifiers. In this work we run the PrefixSpan sequence miner to find all the frequent patterns and the C4.5 decision tree algorithm. In fact, we run the J48 algorithm, a C4.5 clone algorithm available in WEKA [WF99]. Moreover, we define another instance of *MuSer* that runs a linear kernel SVM [Pla98] in the last step of the framework. The SVM algorithm used is also part of the WEKA collection.

To best evaluate our contribution we set the parameters of the *MuSer* framework to values that can highlight the strengths of the architecture and facilitate a comparison against the RUSE-WARMR framework that was presented in Chapter 4. We set the PrefixSpan minimum support value, the λ parameter of the *MuSer* framework, equal to 90% and 80% and map either the top ten or the top twenty ranked patterns, the k parameter of the framework. These same settings were used to evaluate the RUSE-WARMR framework (see Table C.3 in Appendix C). Remember that the RUSE-WARMR framework does not explore valuable temporal information and uses WARMR aggregation procedures to summarize temporal information. Moreover, the RUSE-WARMR framework requires the definition of the WARMR support value, the λ parameter, and like MuSer requires the usage of the chi-square statistic to select the most interesting patterns.

For each instance of the *MuSer* framework we compute the number of patterns, the run-time and the peak memory usage of each step of the *MuSer* framework. In the experiments that have been conducted in the multi-class problem, the protein fold classication problem, we compute and present the average of each score. In the multi-class problem available in the Protein dataset we use the round-robin strategy presented in Section 2.5.3.3. Moreover we present the generalization accuracy of all

obtained models.

The results were obtained using a stratified ten-fold cross validation and we compute the mean and standard deviation of all the results. Moreover we compute the Wilcoxon hypothesis test p-value to measure how significantly our algorithm differs from the RUSE-WARMR framework.

Furthermore, we run the YAP[1] Prolog compiler to: convert the multi-relational data into a sequence database, eliminate non-discriminative patterns and project the PrefixSpan findings into the target table.

5.3.2 Results

In this section we present the results that we obtained in the experiments we run to solve the problems available in the Hepatitis and in the Protein dataset. We do not show the results that we obtained when using the Financial dataset, either the original dataset or the balanced version. The reason for this is that the Financial dataset has too many long sequence patterns with a support value higher than 80%. This issue makes impossible to run the original version of the PrefixSpan algorithm to find all patterns. With the original implementation we are unable to constrain the algorithm to find short size sequence patterns. Moreover, even if run an algorithm that finds all patterns, we may be unable to map them. If we map all such patterns we get files with dozens of giga-bytes that are intractable.

In tables 5.4, D.1, D.2 (see Appendix D for the last two tables) we present, respectively, the number of patterns, the peak memory usage and the run-time required to complete each step of our framework. In each table we identify the dataset used as input, present the value of λ, the support value of the sequence miner, and k, the number of top ranked patterns we map, respectively, in columns one, two and three. Thus, each table row corresponds to an instance of our framework.

In Table 5.4 we present the number of sequence patterns found in each partition of the dataset. We use *Part A* and *Part B* column headers to identify the column where we present the patterns found in, respectively, the minority class and the majority class of the Hepatitis dataset. In columns six and seven we present the number of patterns that we get after running the discriminative and the chi-square filters. Moreover, in the last column we present the number of nodes of the decision tree learned. We

[1] http://www.dcc.fc.up.pt/~vsc/Yap/

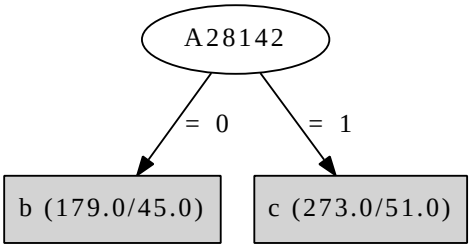


Figure 5.2: *MuSer*'s best induced decision tree for the Hepatitis subtype problem

present the size of the induced tree to highlight the relation between the support value of the sequence miner and the complexity of the generated decision tree models.

In tables D.1 and D.2, that we present in Appendix D, we use the same column descriptors of Table 5.4 and present the peak memory usage and the time needed to find these patterns and learn a classification model.

In Table 5.5 we present the generalization accuracy (and standard deviation) of the models that we learn using either the C4.5 algorithm or the linear kernel SVM algorithm in the last step of the *MuSer* system. Moreover, we present the generalization accuracy that we get when we run the baseline algorithm, the RUSE-WARMR. The RUSE-WARMR results were obtained when we run the C4.5 algorithm in the last step, and with the support value of the WARMR algorithm equal to 80% and by mapping the twenty most interesting patterns. Furthermore, we present the p-value of the Wilcoxon test that we obtained when we compare the generalization accuracy of each *MuSer* instance against the RUSE-WARMR baseline. We show two columns heading the (Wilcoxon) *p-value* text. The first one shows the p-value of each instance of the *MuSer* algorithm that uses the C4.5 algorithm in the last step of the *MuSer* framework. The second column shows the same information for the experiments that we run with SVM to learn the classification model.

In Figure 5.2 we present the tree model generated in the best run of the experiments we made with the Hepatitis dataset. These results were obtained in the first fold of the 10-cv evaluation. In this run we obtained an accuracy of 92%. This result was obtained by setting the support value of the sequence miner to 80% and mapping the ten most interesting patterns. In this specific example the split attribute that appears in the inner node of the tree is the sequence pattern A_{28142} : (30 33 37 48) that corresponds to the (che=normal d-bi=normal gpt=high t-bil=normal) itemset. This is a sequence of size one but we found trees and linear kernels that use sequences of larger sizes. In Figure 5.3 we present a tree that was found when we set the support value of the sequence miner to 90%. Different from the tree presented in Figure 5.2, this tree uses

Table 5.4: Number of patterns found in each step of the *MuSer* framework

			Sequence Miner		Filters		C4.5
Datasets	Support	Top-K	Part A	Part B	Discriminative	Chi-square	TreeSize
Hepatitis	0.9	10	0.1 (0.3)	67.6 (29.6)	67.5 (29.7)	10.0 (0.0)	7.4 (1.8)
		20	0.1 (0.3)	67.6 (29.6)	67.5 (29.7)	20.0 (0.0)	10.2 (3.4)
	0.8	10	122.1 (59.1)	56010.6 (19036.9)	55888.5 (19060.9)	10.0 (0.0)	3.0 (0.0)
		20	122.1 (59.1)	56010.6 (19036.9)	55888.5 (19060.9)	20.0 (0.0)	6.0 (0.0)
Protein	0.9	10	10006.2 (651.8)	356.0 (19.8)	9923.3 (651.4)	10.0 (0.0)	7.5 (2.1)
		20	10006.2 (651.8)	356.0 (19.8)	9923.3 (651.4)	20.0 (0.0)	11.2 (2.6)
	0.8	10	80537.2 (4301.2)	3230.3 (364.2)	81779.5 (4355.3)	10.0 (0.0)	7.6 (1.8)
		20	80537.2 (4301.2)	3230.3 (364.2)	81779.5 (4355.3)	20.0 (0.0)	10.5 (3.9)

Table 5.5: Generalization accuracy of the *MuSer* framework

			C4.5		SVM	
Datasets	Support	Top-K	Accuracy	Wilcoxon p-value	Accuracy	Wilcoxon p-value
Hepatitis	0.9	10	64.10 (6.97)	0.33	65.50 (5.32)	0.18
		20	65.50 (5.72)	0.21	65.50 (6.06)	0.23
	0.8	10	79.90 (5.92)	0.00	80.10 (6.31)	0.00
		20	79.90 (5.92)	0.00	80.10 (6.31)	0.00
	RUSE	–	63.42 (6.92)			
Protein	0.9	10	76.86 (0.03)	0.00	76.14 (0.02)	0.00
		20	78.24 (0.03)	0.00	78.20 (0.03)	0.00
	0.8	10	79.11 (0.02)	0.00	76.86 (0.03)	0.00
		20	79.97 (0.03)	0.00	77.87 (0.03)	0.00
	RUSE	–	63.18 (1.18)			

also primitive attributes, the borndate attribute.

5.3.3 Analysis of the Results

The results that we obtained are quite impressive. Even without using all the information available in each dataset and without including background knowledge from experts, we get significant accuracy wins over the baseline in the hepatitis subtype problem and in the protein fold classification problem, the two problems where we complete the experiments. Only when we run the *MuSer* framework with support value equal to 90% we do not get significant accuracy gains. In both problems we get accuracy gains of approximately 20 percentage points. These results show the contribution of algorithms that explore the time information using special purpose methodologies.

Moreover, the results that we present here are better than the ones obtained by algorithms that explore the order of the events. In the Hepatitis dataset we get gains of approximately 1 percentage point over the work presented in [OYG+04], that was developed with the special purpose to explore the Hepatitis dataset. This is remarkable as we do not explore information available in all tables and do not include expert knowledge in the mining process. [OYG+04] uses computational expensive sub-graph patterns found by the Graph-Based Induction [MHM+99] as input to a decision tree learner.

In the protein problem we obtained generalization accuracy gains of approximately 4

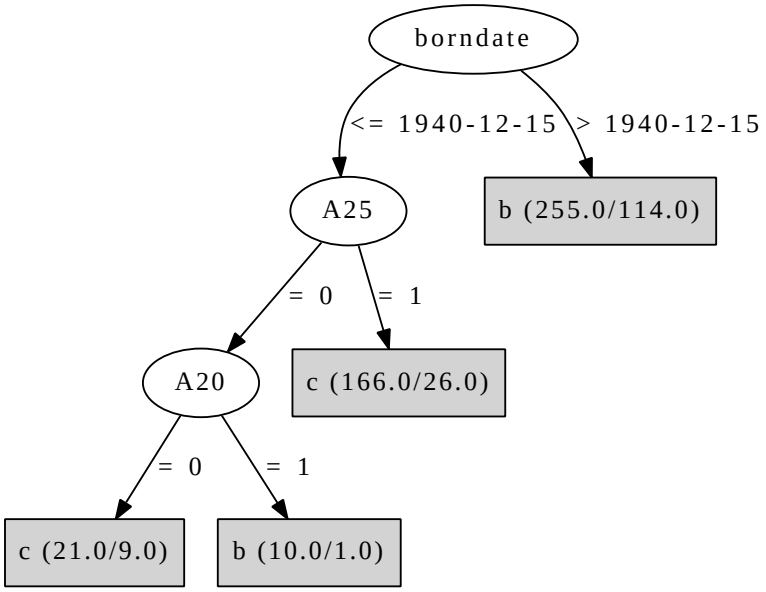


Figure 5.3: Example of a decision tree induced when the support value of *MuSer*'s sequence miner is set to 90%

percentage points over the work developed by Kersting [KRR06]. This work introduces Logical Hidden Markov Models, a methodology that can include expert/background knowledge in the mining process.

In these two problems we observe that the generalization accuracy increases with the decrease of the PrefixSpan support value. Moreover, if we increase the number of patterns to map from ten to twenty patterns, typically we get better results. This later behavior has some limitations as we explain in Chapter 3. If we map a larger set of patterns the propositional classifier can overfit the data.

Regarding the analysis of the number of patterns found in each step of the *MuSer* framework, we get an exponential growth in the number of patterns found. The high growth rate is clear from the analysis of columns 4 and 5 of Table 5.4. The lower the support value the higher the number of patterns. This issue of sequence miners is well known [PHMA+01] and was the source of many problems we have in this work. We were unable to find all patterns having support equal or less than 70%.

The effectiveness of the filters is clear when we analyze the number of patterns we found when we run the sequence miner with a support value equal to 80%. Typically we get numerous irrelevant and redundant patterns. Without the use of both filters, mainly the chi-square filter that selects class related patterns, we could not obtain the presented results or would get worst generalization ability of the obtained classification

models (see Chapter 3). The number of patterns found by the sequence miner makes almost impossible to map all patterns and/or run any classifier on the augmented target table (see chapters 3 and 4 of this thesis). We run some experiments with lower support values and we had to wait several weeks to learn a classification model in a single dataset fold.

Moreover, the number of patterns comes at high memory and run-time costs as we can see in Table D.1 and Table D.2 (see Appendix D). In these tables we do not present information about the discriminative filter, the one that removes matching patterns and is implemented with a simple matching operation that keeps non-agreeing patterns. We do so because the peak memory usage and run-times of this operation can be several orders of magnitude smaller than the other operations. This is especially clear when we run the sequence miner with low support values. The effectiveness of the discriminative filter can be apparent in datasets where we get many agreeing patterns.

Also, we think that the obtained patterns require the analysis of domain technicians. We are not specialists in hepatitis or protein fold structure but we obtained some sequence patterns that, we hope, can be useful to unveil the mechanisms of disease progression or protein structure.

5.4 Conclusions

In this work we presented *MuSer*, a framework that explores temporal information stored in multi-relational databases. The main contribution of the proposed system is a method that explores temporal patterns presented in relational data without losing valuable relational information, the ordering of the events. Based on the excellent results of other algorithms that use the propositionalization technique, the architecture of our algorithm consists of four steps. After a pre-processing phase, where we convert the temporal information available in the relational database into a database of sequences, in the first phase, we find all sequence patterns using the efficient PrefixSpan algorithm. Then, in a second phase, considering the huge number of uninteresting and irrelevant patterns we introduce two filters. The discriminative filter that prunes agreeing patterns and a chi-square filter that sorts patterns using the chi-square statistic. Next we map the most interesting patterns into the relational target table. This way we obtain an enlarged target. In the last phase, we induce a decision tree model by taking the augmented table as input to C4.5 algorithm.

We experimentally evaluate the *MuSer* framework using four datasets and defining a

large set of configurations. Our experiments show that the learned models are highly accurate and readable. Moreover, when we compare the results of *MuSer* against the baseline algorithm, the RUSE-WARMR algorithm presented in Chapter 4, we get significant better results. Remember that RUSE-WARMR aggregates temporal information, i.e., do not explore the temporal information conveniently. Moreover, in the hepatitis subtype problem we get the best results we can find in the literature. In the Protein dataset we are aware of better results but we beat the work of Kersting [KRR06]. This is an important work that was a reference during a set of years. These results are even more impressive as we do not use all the information available in the dataset and do not include expert knowledge.

The algorithm has some limitations related to the huge number of sequential patterns found and the inclusion of domain knowledge in the learning process. In the next chapter we introduce strategies to constrain the search space, eliminate expensive filtering operations and find a constrained set of sequence patterns.

Chapter 6

Discriminative Sequence Miner in Multi-Relational Learning

In this chapter we explore a set of methodologies to learn first-order logic (FOL) theories from multi-relational databases. We present a framework that uses a propositional predictive sequence miner that finds a constrained set of sequences and uses an ILP based algorithm to learn a classification model. With this architecture we can learn theories that can include spatial and both frequent and infrequent temporal data. Moreover, we reduce the need of heavy feature selection strategies and use predicate invention technique to include the findings of the sequence miner in the ILP learning task.

6.1 Introduction

In this work we start from the hypothesis that evolving data may hold relevant information that can be valuable if combined with non-temporal information registered in other relations. The problem we address here is *how best to explore such information?*

Here in this chapter we introduce the *XMuSer* (eXtended MUlti-relational SEquential patteRn knowledge learning) framework that allows us to explore multi-relational datasets that record different types of data. With this architecture we extend MuSer (see Chapter 5) to explore also non-temporal and infrequent data. On the one hand, we can benefit from computationally efficient propositional sequential miners such as cSpade [Zak00] to find the predictive sequence patterns without using complex filtering strategies. On the other hand, we still have access to the original data (static

or dynamic) and can take advantage of the flexibility of Inductive Logic Programming (ILP) to learn in an extended multi-relational database. Indeed, we argue that the first step provides a good insight into the search space, and may enable *XMuSer* to perform better than classical ILP based algorithms in large search spaces. We should observe that the sequence miner and the ILP learning algorithm are decoupled and we can explore subsets of sequential patterns. For instance we can run other sequential miners such as: PrefixSpan [PHMA+01], that finds all frequent patterns, CloSpan [YHA03], that finds closed sequential patterns, or SPIRIT [GRS02], that constrains the search space using regular expressions. Or we can introduce post-processing strategies to select a subset of valuable patterns.

The framework that we present here executes in five steps. First, we encode the multi-relational temporal data into a sequence database. In this new database each example is a heterogeneous sequence that was built regarding both intra-table and inter-table relations within the temporal data. In the second phase, we use a sequence miner to find frequent and class predictive sequences available in the sequence database. By class predictive we mean sequential patterns that are frequent in at least one class partition. In the third phase we use the chi-square statistic to sort all sequential patterns. In the fourth phase we map back the top-k most interesting sequential patterns by building a new relation, the *Sequence Relation*, where each example is characterized by the presence or absence of each pattern. In the last phase, we apply an ILP algorithm to learn a theory from the enlarged database, i.e., from the union of the original multi-relational database with the *Sequence Relation*.

Our system is experimentally evaluated using the four datasets presented in Appendix B: the Hepatitis dataset, the two versions of the Financial dataset and the Protein Dataset. We evaluate the effectiveness of our methodology by measuring the generalization accuracy of the generated classification models. Moreover, we use the well-known Aleph [Sri03] ILP system as a baseline in our comparison. The results prove the benefits of using *XMuSer*'s sequential patterns and non-temporal data.

The contributions of this work are therefore:

- We explore heterogeneous sources of temporal data, either sequence data or time-series data, using a propositional predictive sequence miner. By using this strategy we eliminate expensive filtering operations.

- *XMuSer* can learn theories that include both temporal, non-temporal, frequent and infrequent data.

In Section 6.2 we introduce some concepts that will be very useful in the next sections. In Section 6.3 we present a detailed description of the *XMuSer* framework using an illustrative example to clarify each step. Next, in Section 6.4, we present and discuss the obtained results. At the end of the chapter we conclude and present other research lines.

6.2 Preliminaries

One of the main contributions of the *XMuSer* system is to explore frequent sequence mining, an efficient and direct method to learn temporal patterns in the form of sequences. More precisely, to explore sequence patterns that are frequent in at least one class partition.

When searching for all sequential patterns in a sequence database we face one of the major problems of sequential miners, the huge number of redundant and non-interesting findings. To alleviate this issue we can run sequence miners to find a constrained set of sequential patterns.

As pointed out in Chapter 2, there are two important and valuable types of sequences that can be found among the set of frequent sequences: the closed and the maximal sequences. To find such patterns we can use a wide set of strategies. One strategy is to develop specialized algorithms that find a specific type of pattern, for instance CloSpan [YHA03] can find closed sequential patterns. The other way is to use a post-processing strategy to select a constrained subset of patterns among all frequent patterns. For instance, to find the set of *closed sequential patterns* we can use the following naive strategy:

- Find all frequent sequences.
- Sort descending all sequential patterns according to each one support value and, as a second criterion, sort using the length of each pattern.
- Top-down searches the sorted set of sequential patterns and, for each sequential pattern, eliminate all subsequences having equal support value. The remaining patterns are the set of closed sequential patterns.

To find all maximal sequential patterns we can use the following naive strategy:

- Find all frequent sequences.
- Sort descending all sequential patterns according to their length.
- Top-down searches the sorted set of sequential patterns and, for each sequential pattern, eliminate all its subsequences that appear after the given sequence. The remaining patterns are the set of maximal sequential patterns.

In a classification problem the goal is to find predictive sequences. In such problems a sequence miner takes as input a sequence database where each tuple, $(sid, \alpha, class)$, consists of a sequence, an *id* and a class label. Herein we introduce the concept of *Predictive Sequential Patterns* to select sequential patterns that are frequent in at least one class partition.

Definition 20. (Predictive Sequential Patterns) *Consider a set of sequence database partitions* $D_1, D_2, \ldots, D_N$ *and a support threshold* λ*. We define the set of predictive sequential patterns,* S_{pred}*, using the or* ($\vee$) *operator,* $S_{pred} = \{\alpha \mid support(\alpha, D_1) \geq \lambda \vee support(\alpha, D_2) \geq \lambda \vee \ldots \vee support(\alpha, D_N) \geq \lambda\}$.

To select highly discriminative sequential patterns among these predictive patterns we can compute the support difference across class partitions (see Section 4.2 of Chapter 4) or we can use other techniques from contrast set mining, emerging pattern mining or subgroup discovery [NLW09]. A simple and effective approach [BP01] is to use a metric like the chi-square statistic, and a contingency table, to measure the correlation between each pattern and a given class label.

Using one of these strategies we can select valuable patterns from the sequence database that could be useful at theory learning time [ZL06]. Thus, we introduce the concept of Sequence Relation and Enlarged Database to include such valuable patterns in the learning process.

Definition 21. (Sequence Relation and Enlarged Database) *Consider a multi–relational database* $\mathbf{r}$*, a sequence database* D *encoded (see Section 5.2.1 of Chapter 5) from* $\mathbf{r}$*, where each sequence id equals the primary key of* $\mathbf{r}$*'s target table. Also consider the set of sequential patterns* S *obtained from solving the sequential mining problem. We define the* sequence relation*,* $\mathbf{r}_{sr}$*, to be the set of tuples* $(sid, \alpha_1^B, \alpha_2^B, \ldots, \alpha_n^B)$ *where sid is the sequence id in* D *and* α_i^B *is a binary attribute whose value is obtained according to the projection of the sequential pattern* α_i *in the sequence database. The* enlarged database *is the database resulting from the union of the original multi-relational database and the sequence relation. Formally we define the enlarged database as* $\mathbf{E_r} = \mathbf{r} \cup \mathbf{r_{sr}}$.

Algorithm 7: *XMuSer* pseudo-code

input : a multi-relational database $\mathbf{r}$; two thresholds: λ, the sequence miner minimum support value, and k, the number of interesting patterns to map

output: a first-order classification model

```
1  Sequence Encoding
2  |  s ← SequenceCoding (r);
3
4  Find Frequent Sequential Patterns
5  |  fs_patterns ← SequenceMiner (s, λ);
6
7  Sort patterns according to some criterion
8  |  fs_sort ← sort (fs_patterns);
9
10 Mapping top-k patterns
11 |  r_sr ← Mapping(r_targetExamples, fs_sort, k);
12 |  E_r ← r ∪ r_sr;
13
14 Learn a theory with an ILP algorithm
15 |  ILP Algorithm(E_r);
16
```

6.3 The XMuSer framework

In this section we present a detailed description of the *XMuSer* architecture. The framework has five main steps. Firstly, if needed, we encode the temporal data into a sequence database. In the second phase, we run a sequence miner to find all predictive sequential patterns. In the third phase, we sort the predictive sequential patterns using the chi-square statistic value. Next, in phase four, we select the top-k most interesting sequential patterns and we build a relation where each example is characterized by the presence or absence of the k most interesting sequential patterns. Last, we learn a theory on the enlarged database. This enlarged database is the union of the original database with the new sequence relation.

Algorithm 7 presents the pseudo-code for *XMuSer*. Next, we explain each one of the major components in self-contained subsections. Throughout, we follow an illustrative example, a classification problem, presented in Table 5.1, Section 5.2.1 of Chapter 5. This example is inspired in the relational Hepatitis dataset. The example has three tables registering the follow-up of two patients. One of the tables is the target table, named *Patient Info*, where each record describes each patient, identified by a masked ID, and records the sex, the date of birth and the class label of each patient. The other two tables are fact tables registering blood analysis and urinalysis examinations made by each patient. The *Blood Analysis* table registers the examination date and the patient's RBC and WBC parameters. Table *Urinalysis* registers the examination date and the value of three parameters: *alb*, *plt* and *ttp*.

6.3.1 Data Encoding - Phase 0

Here we use the same encoding strategy that we introduced in Section 5.2.1 of Chapter 5 to convert the multi-relational temporal data into an amenable sequence database. Remember that for each example in the multi-relational target table we find all associated relations that have temporal records, and generate a chronological sequence of multiple events.

After this pre-processing stage we get a sequence relation suitable to be used by a sequence miner. In Table 6.1 we present a hypothetical sequence database that can be generated using this encoding technique. In this table we present the sequence database obtained from the illustrative relational database presented in Table 5.1 (see Chapter 5). We present the sequence of events for patients one and two. Moreover, we

Table 6.1: Sequence Database

ID	Sequence	Class
1	(9) (3 5 7 8) (7) (3 6)	c
2	(3 4) (7)	b
3	(1 5) (4)	b
4	(3 5) (7)	c
5	(5 8) (7)	c
6	(4)	b

present four additional patients, that were not presented in the illustrative database in Table 5.1, but will be useful to explain some steps of our algorithm. In the third column we show the class labels of each sequence, we have two classes, class **b** and class **c**.

6.3.2 Finding Frequent Sequential Patterns - Phase 1

To explore the sequence database encoded in the previous step we can fit any sequence miner that finds patterns that are frequent in the full dataset but outputs the pattern support in each class partition. The output of the sequence miner is very important, with the class support value we can compute metrics like the support difference and chi-square statistic without the need to use expensive operations like matching and mapping.

Our first step is to run a sequence miner on the sequence database to find predictive frequent sequences, the ones having, in at least one class partition, a support value equal or higher than a user-defined threshold. Following the illustrative example, if we set the support value to 0.5 and run cSPADE algorithm to find sequential patterns available in the *sequence database* (see Table 6.1) we get the sequential patterns presented in Table 6.2. cSPADE outputs the overall support of each pattern and the support of each pattern in each class partition. Moreover, we slightly modify cSPADE to compute and output a measure of interest for each pattern. Here, we set cSPADE to output the chi-square statistic for each pattern [BP01] but we can configure the cSPADE algorithm to compute other metrics like the support difference.

Considering the nature of the sequence miner, the type and the dimension of the

Table 6.2: Predictive sequential patterns found

	Support Value			
Predictive Sequential Patterns	Overall	Class **c**	Class **b**	Chi-square
(3)	3	2	1	0.667
(4)	3	0	3	6.000
(5)	4	3	1	3.000
(7)	4	3	1	3.000
(3)(7)	3	2	1	0.667
(5)(7)	3	3	0	6.000
(3 5)	2	2	0	3.000

Table 6.3: Ordered list of sequential patterns

Rank	Sequential Patterns	Chi-square
1	*(5)(7)*	6.000
2	*(4)*	6.000
3	*(3 5)*	3.000
4	(5)	3.000
5	(7)	3.000
6	(3)(7)	0.667
7	(3)	0.667

problem being addressed we usually get a large number of sequential patterns that is almost impossible to manage. Thus in some cases we may use post-processing strategies to select a subset of patterns that compress the information available in the frequent set of patterns. For instance, we may be interested in closed and maximal sequences and use the post-processing strategies introduced in Section 6.2. Moreover, we may be interested in discriminative or correlated patterns and we must use the metric value outputted by the modified cSPADE to select features.

Table 6.4: Sequence Relation

ID	S_1	S_2	S_3
1	true	false	true
2	false	true	false

6.3.3 Sort Sequential Patterns - Phase 2

To select the most interesting patterns we sort all the sequential patterns found using a metric. In Table 6.3 we present the ordered list of sequential patterns that we obtain by sorting the patterns available in Table 6.2. In this example, we sort the patterns according to the cSPADE chi-square statistic value. The k most interesting patterns (the top ones) will be used to build the final classification model.

6.3.4 Mapping Back Interesting Sequences - Phase 3

At this point it is important to remark that for each example we have the temporal data encoded into a sequence database and that we want to build a relation where each example is characterized by the most interesting sequential patterns.

To better explain our procedure we present Table 6.4, the sequence relation that was obtained by applying our mapping procedure on the example data. This table has four attributes, the example ID and the three most interesting sequences (the top three patterns of Table 6.3). If we get ties we select the longest pattern.

We develop a mapping strategy that maps each one of the selected sequential patterns into a Boolean attribute and, portray this mapping as a new table. This new table, named *sequential relation*, has an entry for each example on the target table and has $k+1$ attributes: the top-k most interesting features and the example ID, usually the target table primary key. To compute the value of each attribute we use the information encoded in the sequence database and the subsequence definition(see Section 2.6.2 of Chapter 2). We apply the following rule:

- If the sequence associated with the new attribute is a subsequence of the example, the new attribute takes value *true* for that example.
- Otherwise the attribute takes value *false*.

Using this procedure we get a new table, the *sequence relation*, that represents the most interesting temporal data available in the multi-relational database.

6.3.5 Learning a Theory - Phase 4

In this last step we learn a theory. We take all the primitive tables and the new sequence relation as input to an ILP algorithm, such as Aleph, and learn a set of FOL clauses. These clauses can therefore use the primitive tables and the new one, that encodes the temporal information. I.e., the ILP learner can use temporal, non-temporal, frequent or infrequent data. This is a major difference from what we have done in the MuSer framework. In that work we only learn classification models that use static attributes of the entities and temporal patterns.

To learn a theory using the illustrative example as input to the Aleph algorithm, we can define the following language bias:

```
patient_info(+id,+sex,+bornDate,+class).
blood_analysis(+id,-date,#binValues,#binValues).
urinalysis(+id,-date,#examName,#binValues).
sequence_relation(+id,#a,#a,#a). sequence_relation(+id,#a,-a,-a).
sequence_relation(+id,-a,#a,-a). sequence_relation(+id,-a,-a,#a).
binValues(low). binValues(normal).
binValues(high). examName(alb).
examName(plt). examName(ttp).
 a(true). a(false).
```

Here *+,-,#* specify input, output and constant modes. An example of a clause we can find with this hypothesis language is:

```
patient_info(A,B,C,c):-
  blood_analysis(A,D,high,normal),
  urinalysis(A,E,plt,high),
  sequence_relation(A,true,F,G).
```

This clause has the predicate *patient_info* at the head, and a call to the predicates *blood_analysis*, *urinalysis* and *sequence_relation* in the clause body. This clause explains (or covers) patient number one in the database, a patient from class **c**.

6.4 Experimental evaluation

In this section we describe the configurations and results of our experimental evaluation. First, we describe the configurations of the experiments. Then, we present the results we obtain when we run a set of instances of the *XMuSer* framework. At the end of this section we discuss the obtained results.

6.4.1 Configuration of the Experiments

When defining the experimental configuration to evaluate our framework we were constrained by the representational complexity and the amount of data. We use the YAP Prolog compiler [CRD12] to implement three tasks: converting the temporal data presented in the multi-relational database into a sequence database, that can be fed to the cSPADE algorithm; map the most interesting sequential patterns on the sequence relation; and run the Aleph algorithm. Moreover, we run the cSPADE algorithm to find frequent sequences available in the sequence database and use the Aleph algorithm to learn a classification theory.

Using these tools, we define two configurations of the input parameters and run *XMuSer* to solve three classification problems: the hepatitis subtype, prediction of successful loans (the financial problem) and protein fold recognition. These three problems are available in four datasets described in Appendix B.

We test two different sequence miner support values (the λ parameter): 90% and 80%. Using these minimum support values we found all patterns available in the Hepatitis and Protein datasets. In the experiments we run in the Financial datasets (original and balanced versions) we had to limit the size of the sequence patterns to four, otherwise we would be unable to complete the experiments. In the fourth phase we map the ten most interesting patterns ($k = 10$). For comparison purposes we also run the stand-alone Aleph algorithm to solve each one of the three problems. Furthermore, besides presenting *XMuSer* results using all sequential patterns found by cSPADE, we present results using two subsets of sequential patterns, the closed set and the maximal subset. The closed and maximal sequences are obtained using a post-processing strategy and the naive strategy described in Section 6.2. Therefore, we run three instances of *XMuSer*: Frequent, Closed and Maximal.

We evaluate our framework using a stratified ten-fold cross-validation procedure and compute: the mean number of patterns found after each step of *XMuSer*; the mean,

and the standard deviation, generalization accuracy of the learned models; the mean time spent and the maximum memory usage to complete each step of *XMuSer*. Moreover, we compute the mean number of rules learned by the Aleph algorithm, both as a component of *XMuSer* and as a stand-alone algorithm. In the multi-class problem available in the Protein dataset we use the round-robin strategy presented in Section 2.5.3.3. Concerning the generalization accuracy, we also compute the Wilcoxon hypothesis test p-value to measure how significantly our algorithm differs from the reference algorithm, the stand-alone Aleph algorithm. We set Wilcoxon significance level to 0.05.

Concerning the comparison, we use the same background knowledge and bias when running the Aleph algorithm, either using Aleph as a component of the *XMuSer* framework or by running it as stand-alone reference algorithm. The major difference is that when running Aleph as a component of *XMuSer* we allow extra refinements by introducing the sequence relation.

As our goal is to evaluate the contribution of our novel technique, we do not search for the best overall results but only want to prove that by using our methodology we can explore efficiently and effectively temporal information stored in multi-relational databases and yet obtain accuracy gains. Thus, we run Aleph algorithm using a relatively simple bias, relying on the predefined tables in the database.

6.4.2 Results

In this section we present the results that we obtained when running *XMuSer* and the stand-alone Aleph algorithm in the four datasets: Hepatitis, the two versions of the Financial and the Protein.

In Table 6.5 we present the mean number of patterns found in each step of *XMuSer*. We present the mean number of sequential patterns found by the cSpade sequence miner and the number of rules that were learned by the Aleph algorithm, both as a stand-alone algorithm or as the final step of the *XMuSer* framework. As we run cSPADE and the two naive strategies introduced in Section 6.2 to find frequent, closed and maximal patterns, we report the number of frequent, closed and maximal sequential patterns found. In Table 6.6 we present the mean generalization accuracy (and the standard deviation) of the *XMuSer* algorithm. Moreover, we present the p-value of the Wilcoxon test that we obtained when we compare the generalization accuracy of each *XMuSer* instance against the baseline, the stand-alone Aleph algorithm. We

Table 6.5: Number of patterns found in each step of the XMuSer framework

		Sequence Miner: cSpade			Theory Learning: Aleph		
Datasets	Support	Frequent	Closed	Maximal	Frequent	Closed	Maximal
Hepatitis	0.9	8.4 (4.4)	8.4 (4.4)	5.7 (3.8)	174.0 (5.9)	174.0 (5.9)	174.4 (6.3)
	0.8	1055.0 (211.2)	729.1 (131.4)	254.4 (33.6)	140.2 (7.6)	139.9 (8.0)	137.0 (7.5)
	Aleph	–	–	–	182.9 (1.3)		
Financial	0.9	45720.6 (2431.2)	6141.4 (497.8)	1846.4 (126.6)	35.7 (2.0)	35.7 (2.0)	26.9 (1.6)
	0.8	96759.2 (4448.2)	14379.4 (607.6)	2902.2 (157.5)	35.7 (2.0)	35.7 (2.0)	26.1 (1.9)
	Aleph	–	–	–	35.7 (2.0)		
FinancialB	0.9	63356.1 (6145.9)	5092.4 (313.4)	2214.0 (90.9)	24.6 (1.6)	24.6 (1.6)	19.9 (1.7)
	0.8	130709.4 (5313.6)	10851.2 (465.8)	3062.4 (163.7)	24.6 (1.6)	24.6 (1.6)	13.8 (5.5)
	Aleph	–	–	–	24.6 (1.6)		
Protein	0.9	1463.4 (68.5)	1341.0 (68.4)	301.4 (21.6)	184.3 (22.2)	231.7 (21.1)	196.0 (21.3)
	0.8	15283.6 (18741.7)	13577.3 (16087.4)	1985.2 (1536.5)	104.5 (72.7)	146.1 (91.5)	119.5 (78.0)
	Aleph	–	–	–	306.1 (5.322987)		

Table 6.6: Generalization accuracy of the XMuSer framework

		Accuracy and Wilcoxon p-value of the XMuSer					
Datasets	Support	Frequent	p-Value	Closed	p-Value	Maximal	p-Value
Hepatitis	0.9	60.71 (2.25)	0.065	60.71 (2.25)	0.065	60.32 (2.55)	0.065
	0.8	65.31 (5.65)	0.108	65.12 (6.00)	0.108	65.12 (5.86)	0.108
	Aleph	58.88 (1.07)					
Financial	0.9	79.15 (6.43)	1	79.15 (6.43)	1	80.80 (5.64)	0.539
	0.8	79.15 (6.43)	1	79.15 (6.43)	1	80.42 (7.07)	0.674
	Aleph	79.15 (6.43)					
FinancialB	0.9	58.06 (14.46)	1	58.06 (14.46)	1	65.56 (9.04)	0.233
	0.8	58.06 (14.46)	1	58.06 (14.46)	1	59.17 (16.87)	0.639
	Aleph	58.06 (14.46)					
Protein	0.9	66.82 (4.54)	0	73.01 (3.64)	0	73.78 (3.36)	0
	0.8	70.81 (19.30)	0.015	75.25 (16.04)	0.002	75.46 (21.77)	0.002
	Aleph	50.79 (3.37)					

show three columns heading p-value, one for each type of pattern: frequent, closed and maximal. Furthermore, in all experiments we use the same bias when running *Aleph* algorithm both as a component of *XMuSer* or as a stand-alone algorithm. The only difference is that the Aleph component of the *XMuSer* can also use the sequence relation to learn clauses.

In Appendix E, we present the run-time (Table E.2) and the peak memory usage (Table E.1) needed by the sequence miner, to map patterns and to learn a classification theory. We present the mean run-time needed by cSPADE sequence miner to find frequent sequences and do not present the time required by the naive strategy to find the closed and maximal subsets of patterns. We do this because the run-time of these post-processing strategies is negligible when compared with the run-time of the sequence miner. In all tables we also present the results that we obtained when we run the stand-alone Aleph algorithm. In Table E.3 (Appendix E) we present the results that we obtain if we run a propositional classifier in the last step of the XMuSer framework. We present results for two propositional algorithms, the C4.5 and the linear kernel SVM. Moreover we present results for $k = 10$ and $k = 20$. In Table E.4 we present the size of the decision trees learned with the C4.5 algorithm.

6.4.3 Analysis of the Results

We present results only for two λ values, 90% and 80%. For lambda values slightly larger than 90% we do not observe any significant change in comparison against the 90% value. Regarding the values lower than 80%. First, we are limited by the available computational power and time. With the current resources, if we set lambda parameter to values lower than 80% we are unable to run *XMuSer* to solve all problems. Second, in the datasets that we were able to run *XMuSer* we do not identify significant changes in the results. If we map a larger set of patterns, $k > 10$, we get small and non significant improvements over the results reported ($k = 10$).

Nevertheless, the development of the *XMuSer* allows us to generate classification models for the Financial problem, either using the original or the balanced Financial dataset. Remember (Section 5 of Chapter 5.3) that we were unable to explore the Financial datasets using MuSer with the original PrefixSpan algorithm. First, the sequence miner that we use in *XMuSer* allow us to constrain the size of the sequence patterns we can find. Here we limit the size of patterns found in the Financial dataset to four itemsets. Second, with the modifications that we introduce in the sequence miner we eliminate the need to use the two filters introduced in MuSer (see Chapter 5 and [FGC10, FGC11]). The chi-square filter used in MuSer requires that we map all the patterns found by the sequence miner. This is a highly expensive operation but that is needed to select class correlated patterns. Here, in *XMuSer*, the chi-square statistic is embedded in the sequence miner, the cSpade algorithm, and each pattern found is associated with the class support and the chi-square statistic value. As we can see in tables $E.2$ and $E.1$, the mapping operation in *XMuSer* requires less memory and less time to complete. For comparison purposes the reader should consider only the results with $k = 10$, the same number of patterns that we map in *XMuSer*. Moreover, we can see that the amount of peak memory usage needed by cSpade (component of the *XmuSer* framework) is larger than the one needed by PrefixSpan (component of the *MuSer* framework). cSpade takes the full dataset as input and PrefixSpan takes only one partition at a time. Furthermore, the run-time needed by PrefixSpan is slightly larger than the run-time needed by cSpade. This was expected as PrefixSpan explores a partition at a time and finds a large number of patterns.

As expected, *XMuSer* execution time is greater than the stand-alone Aleph algorithm. The execution time heavily depends on the input parameter λ. If we run *XMuSer* by setting λ input parameter to low values we get a huge number of sequential patterns and thus we get a run-time overhead. This is a typical behavior of sequence miners,

the lower the support the higher the execution time and the higher the number of patterns found. Regarding the post-processing strategies that we used to select the closed and maximal sequential patterns, we can say that the time needed to select the maximal sequential patterns is higher than the time needed to select the closed sequential patterns. This is especially clear when we run the sequence miner with 0.8 low support values.

Furthermore, if we analyze theoretically the run-time of the last step of *XMuSer*, we can say that by adding the sequence relation to the search bias we will have to explore a 2^k larger search space, where k is the number of patterns/attributes in the sequential relation.

In our experiments, when we compare the mean run-time of the last step of our framework against the stand-alone Aleph algorithm we can see that the time spent by the last component of our framework is almost the same as the run-time of the stand-alone Aleph algorithm. This means, that if we stay on low k values we can effectively and efficiently explore valuable temporal data with our framework.

Regarding the generalization accuracy of the learned models, we can say that in all classification problems we address, *XMuSer* obtained better results than the baseline, the stand-alone Aleph algorithm. The best accuracy gains were obtained in the hepatitis subtype and in the protein fold classification problems. In the protein problem we get significant wins for all parameters and type of patterns we found. Moreover, the best results were obtained when we map the most interesting closed or maximal patterns. If we compare the obtained results with the ones presented in Chapter 5, we can see that MuSer generates higher accurate results than *XMuSer*. This is especially clear when we analyze the results obtained in hepatitis subtype problem. At first sight, we can explain this generalization ability difference by the usage of a ILP based classifier in the last step of *XMuSer*. But this is not true. If we replace the ILP algorithm used in the last step of *XMuSer* by a propositional algorithm we continue to get worst results than the ones obtained by the MuSer framework. In Table E.3 we present results when we run the C4.5 or the linear kernel SVM propositional algorithms in the last step of *XMuSer*. Moreover, by using ILP algorithms we can easily introduce expert knowledge [Dom12] in the learning task and get potentially high accurate models.

The obtained results show the contribution of the strategy introduced in MuSer to find the sequence patterns. In MuSer we take each partition as input and find a larger number of patterns than when we run cSpade sequence miner. This is clear when we

analyze tables 5.4 and 6.5 that present the number of patterns found by PrefixSpan and cSpade. The dual strategy of MuSer finds a large number and high interesting patterns. If we observe the complexity of the theories learned by MuSer, last column of Table 5.4, and the propositional XmuSer (replacing the ILP learner by a c4.5 or SVM learner), Table E.4, we can see that MuSer generates models with lower complexity.

These remarks show that we should modify *XMuSer* to use the dual strategy and use the support value in each partition to reduce the mapping requirements. Without this modification and without expert knowledge to include in the learning process, we hardly obtain significant gains over the results that we report for the MuSer framework (see Table 5.5 in Chapter 5).

A final remark concerns the time and memory needed to learn a FOL theory. If we compare the memory and run-time usage of the XMuSer framework presented in tables E.1 and E.2 with the results presented in tables D.1 and D.2 (see Chapter 5) we can see that ILP algorithms require a larger amount of time and memory than the propositional ones. This performance issues show that it is always a good thing to try propositional learners, especially if you do not have valuable expert knowledge to include in the ILP learning task [Dom12].

Overall, we can say that finding other types (frequent, closed or maximal) of sequence patterns and the usage of infrequent and non-temporal data in the learning process are ideas that have potential to improve a classifier. Moreover, the number of patterns found using the dual strategy of the MuSer framewrok (see Chapter 5) also proved to be valuable. Therefore, we should develop new algorithms and frameworks that find highly interesting and expressive sequence patterns and include temporal, non-temporal, frequent or infrequent data in the learning process.

6.5 Conclusions and Future Work

In this chapter we have presented *XMuSer* a new multi-relational framework that explores temporal and non-temporal information stored in a database. With this architecture we can include information of frequent and infrequent patterns in the classification model. The *XMuSer* architecture is an effective alternative to previous ILP approaches that incorporate temporal data in the final first-order theory by either using time aggregation strategies [DT99], like time window aggregation, or by only refining ILP clauses or predicates that explicitly explores time. Differently from these approaches, our methodology, grounded in sequence data representation and in the

strengths of predictive sequential miners, explores efficiently and with time abstraction any kind of temporal data. Our approach is general and can be used in conjunction with any classical ILP algorithm.

To design our five step architecture we introduced some new methodologies. We use sequence encoding, a technique to encode the temporal data available in the original database into a sequence database (see Section 5.2.1). Regarding the number of findings of the sequential miner and the need to obtain class correlated sequential patterns among these findings, we introduce a filter that selects the top class related sequential patterns. We use the chi-square statistical to sort the sequential patterns and map the top most interesting ones. Given that these sequential patterns encode interesting temporal data we create a technique to incorporate this information in the final FOL theory. We define the sequential relation, a relation that encodes the time information of each example, the sequential patterns found by the sequential miner. Last, we define the enlarged database, a database that is the union of the original database and the sequence relation, and induce a first-order theory on the enlarged database.

The experimental evaluation presented in this chapter shows that *XMuSer* can find sequence patterns that have strong impact in the classification performance. If available, *XMuSer* can include valuable expert knowledge in the learning process. This capability is beyond the MuSer (presented in Chapter 5) capabilities. The experiments show that learners are highly sensible to the type (either frequent, closed or maximal) of sequences. In this study we found that we can obtain high accurate models when we map the closed or maximal patterns. Moreover, if we compare these results with the ones obtained with MuSer (see Chapter 5), we can see that MuSer can find highly discriminative patterns by mining each class partition individually.

In the next chapter we introduce algorithms and methods that search for highly discriminative temporal patterns. We discard the use and development of propositional techniques and will explore FOL temporal representation formalisms.

Chapter 7

Contrasting Logical Sequences in Multi-Relational Learning

In this chapter we study first-order logic (FOL) sequence miners and introduce the *BeamSouL* FOL sequence miner to find contrasting logical sequences. Unlike the work presented in the previous chapters, this algorithm explores directly the logic relational information available in a multi-relational database. Moreover, the algorithm includes a beam search strategy that can find long logical sequences. Furthermore, we develop a first-order classification framework, the *ConFOLding*, to evaluate the discriminative power of the long patterns found by the *BeamSouL* algorithm. This architecture uses a predicate invention technique to include the most interesting contrasting patterns in the ILP learning phase.

7.1 Introduction

Finding FOL temporal patterns in temporal databases is an important challenge in domains as diverse as video processing, computational biology, elderly monitoring. One of the main issues is the difficulty in finding long and low support discriminative sequential patterns. This problem has been initially identified in the propositional setting [Bay98] and efficient algorithms have been proposed [Zak00, DZ09]. A major issue with these algorithms is that they are unable to explore multi-relational temporal data without losing valuable logico-relational information [FGC08, FGC10, FGC12b].

In the last decade a few special purpose algorithms have been proposed to find sequence patterns in multi-relational temporal databases [LR04, EMBF09, FGC12b]. Using

either levelwise or version spaces search methodologies, these algorithms were designed to find the complete set of frequent patterns. Depending on the complexity of the search space, these algorithms may fail to find the longest (with small frequency values) and most interesting sequences available in large multi-relational databases. Moreover, typically they search for patterns using only the frequency of the pattern as a quality criterion. Thus, high confidence [HKP11] and discriminative patterns will not be found.

Now we overview the *BeamSouL* (Beam Search for Sequences of Logical Atoms), a FOL sequence miner that can find contrasting sequence patterns available in multi-relational temporal databases. Our algorithm builds upon SeqLog, a representation formalism akin to DataLog [DT99] that is suitable to represent logical sequences [LR04] (see Section 2.6.2.2 of Chapter 2 for a brief introduction to the SeqLog formalism). *BeamSouL* uses a hybrid levelwise search procedure to find logical sequences. In the first step, the algorithm applies exhaustive search to find all frequent short size patterns. In this step the algorithm uses a user-defined threshold to limit the exhaustive search to the first levels of the lattice. In the second step, the algorithm employs a k-beam search strategy to explore deeper levels of the solution space and find a subset of long and interesting patterns. In each level the confidence metric, or other metric, is used to select the k most interesting patterns that will be specialized. The algorithm uses SeqLog's optimal refinement operator to generate candidate sequences and its subsumption operator to select contrasting patterns [NLW09] without mining all database partitions.

To evaluate the discriminative power of the *BeamSouL* findings, we introduce the *ConFOLding* multi-relational classification framework. This framework uses an ILP algorithm [Rae08] to learn a theory and a predicate invention [Sta96] technique to include logical sequential patterns in the learning process. We experimentally evaluate our framework using the four datasets presented in Chapter B. Moreover, for each problem we learn a classification model using either frequent, closed or maximal contrasting patterns.

The main contributions presented in this chapter are:

- A new and efficient hybrid algorithm that finds long FOL sequences available in large multi-relational databases. The algorithm combines exhaustive and beam search and can be tuned to find all or a subset of contrasting patterns.

- Use the confidence metric to guide the beam search by selecting the most inter-

esting frequent sequences to refine in the upper levels.

- An ILP-based classification framework that uses the Predicate Invention technique [Sta96] to include the most interesting contrasting patterns in a FOL classification model.

In Section 7.2 we present the *BeamSouL* sequence miner and the FOL classification framework that we develop to evaluate the contribution of the patterns found by the *BeamSouL* algorithm. In Section 7.3 we evaluate and analyze the results of the *BeamSouL* algorithm. Last, we conclude and present some research directions that we will explore in the near future.

7.2 The BeamSouL Algorithm

We first present the *BeamSouL* sequence miner. Then we present the ILP based classification framework that we develop to assess the benefits of *BeamSouL*.

7.2.1 BeamSouL: A FOL Sequence Miner

The *BeamSouL* algorithm uses a levelwise hybrid search strategy to find a subset of contrasting patterns. In the first phase the algorithm runs an exhaustive search and then changes to k-beam search. In each level the frequent patterns are found using candidate generation-and-test strategy. Moreover, the contrasting patterns are found by mining only one database partition. The support in the other partition is obtained using the subsumption operator.

In Algorithm 8 we present the algorithm pseudo-code. In the pseudo-code we assume that the sequence database is partitioned into two sets D_{pos} and D_{neg}. And our goal is to find FOL patterns using a predefined set of predicates specified in the set F.

7.2.1.1 Initialize Parameters and Find Patterns of Size One

In the initialization phase the predicates in F and the logical sequences in the positive partition of the database, D_{pos}, are used to generate level (or size) one candidate patterns. In line number two of the algorithm we can see the refinement level vector $(1, 0, 0, 0)$ indicating the refinement level of pattern $p = f(X_1, \ldots, X_n) \in F$. For

Algorithm 8: Beam-wise Logical Sequence Miner

input : the positive and negative partitions of a logical sequence database, $D_{\mathbf{pos}}$ and $D_{\mathbf{neg}}$; two support thresholds, λ_{pos} and λ_{neg}; a set of predicates with their arity, F; the level where the search method changes, l_{ch} and the beam size, k

output: a subset (or all) of logical contrasting sequences

1 **Initialize candidate and frequent set**
2 $C_1 \leftarrow \{(f(X_1, \ldots, X_n), (1, 0, 0, 0))|(f, n) \in F\}$;
3 $L_1 \leftarrow \{(p, sup_{pos}, sup_{neg}, conf_{pos})|p \in C_1 \wedge sup_{pos}(p) \geq \lambda_{pos}\}$;

4 $l \leftarrow 2$;
5 **repeat**
6 **if** $(l \leq l_{ch})$ **then**
7 $C_l \leftarrow$ `GenerateCandidates` (L_{l-1});
else
8 $L_{l-1}^{top} \leftarrow$ selectTop(L_{l-1},k);
9 $C_l \leftarrow$ `GenerateCandidates` (L_{l-1}^{top});
10 $L_l \leftarrow \{(p, sup_{pos}, sup_{neg}, conf_{pos}) \in C_l|sup_{pos}(p) \geq \lambda_{pos}\}$;
11 **if** $(C_l = \emptyset \vee L_l = \emptyset)$ **then**
12 break;
13 $l++$;
until
14 **return** `GetContrastingSubset` $(\bigcup_{j=1}^{l} L_j, \lambda_{neg})$;

simplicity we omit the refinement level vector throughout the algorithm pseudo-code and the algorithm description. Next, see line 3, this initial set of candidates patterns are evaluated and the patterns $p \in C_1$ satisfying the anti-monotonic constraint $sup_{pos}(p) \geq \lambda_{pos}$ are selected as frequent patterns. We also compute the support of each pattern p in the negative partition of the database, the $sup_{neg}(p)$. This latter support value will be used only in the last instruction of the algorithm, when we return all the contrasting patterns found, to prune the sequence set. This is required since successive refinements can reduce the frequency of a pattern. Moreover, we compute the confidence of each pattern in predicting the positive class, the $conf_{pos}$ value (see Section 2.5.2 of Chapter 2 to obtain details about the confidence metric). Thus, for each pattern p we get a vector $(p, sup_{pos}(p), sup_{neg}(p), conf_{pos}(p))$.

7.2.1.2 Hybrid Search and Candidate Generation-and-Test Strategies

As said above, we apply a hybrid search strategy. More precisely, at a certain level l_{ch} we move from exhaustive search to a beam search strategy, allowing the algorithm to find long and interesting contrasting patterns. The l_{ch} parameter is a user-defined parameter with the best value depending on the size of the two database partitions and the characteristics of the machine where the algorithm runs. Currently, exhaustive search is run to assess the practical limits of the exhaustive search.

At each refinement level l the algorithm uses the frequent patterns of the previous refinement level, $l-1$, to generate the candidate patterns. The optimal refinement operator, ρ, presented in Section 2.6.2.2 of Chapter 2 is used to specialize the patterns of level $l-1$. Only the patterns satisfying the inequality $sup_{pos}(p) \geq \lambda_{pos}$, are considered frequent (see line 10 of the Algorithm 8). For each pattern satisfying this anti-monotonic constraint we compute, again, the support in the negative database and the confidence of each pattern in the positive database. Thus, for each sequential pattern p, we get a vector $(p, sup_{pos}(p), sup_{neg}(p), conf_{pos}(p))$ describing the pattern. This way we do not need to find all frequent patterns in both partitions.

The main difference between the two search strategies is the technique used to select the patterns to refine in the current level. In the exhaustive phase we take all the frequent patterns of the previous level to generate new candidates (see line 7). In the beam phase only a subset of high-confidence patterns of the previous level is used to generate candidates in the current level. The confidence value of each pattern of level $l-1$ is used and only the k patterns with high confidence are selected to be specialized. We can see this selection procedure in line 8.

7.2.1.3 Return Contrasting Patterns

The algorithm finishes the search when no more candidates or frequent patterns can be found (see line 11). Then, the algorithm selects and returns all contrasting patterns found in all levels of the lattice (see line 14). The selection strategy uses the monotonic constraint $sup(p, D_{neg}) \leq \lambda_{neg}$ to eliminate frequent but non-contrasting sequences.

It is important to note that by defining the adequate parameters, our algorithm can find the complete set of contrasting patterns or find frequent patterns (contrasting and non-contrasting) in only one database partition. The former task can be accomplished by setting l_{ch} equal to a large value and the latter task can be accomplished by setting

Algorithm 9: *ConFOLding* pseudo-code

input : a multi-relational database, D; the *BeamSouL* algorithm parameters; the number of interesting patterns to map, m

output: a first-order classification model

1 **Find contrasting patterns**
2 $\quad \mathbf{fs}^{pos}_{patt} \leftarrow$ `BeamSouL` $(D_{pos}, D_{neg}, \lambda_{pos}, \lambda_{neg})$;
3 $\quad \mathbf{fs}^{neg}_{patt} \leftarrow$ `BeamSouL` $(D_{neg}, D_{pos}, \lambda_{pos}, \lambda_{neg})$;

4 **Sort patterns using confidence metric**
5 $\quad \mathbf{fs}^{top}_{rank} \leftarrow$ `rankSel` $(\mathbf{fs}^{pos}_{patt}, m) \cup$ `rankSel` $(\mathbf{fs}^{neg}_{patt}, m)$;

6 **Mapping top-m patterns**
7 $\quad \mathbf{r}^{new} \leftarrow$ `MappingPI(`$\boldsymbol{fs}^{top}_{rank}, D$`)`;
8 $\quad \mathbf{E_r} \leftarrow \mathbf{D} \cup \mathbf{r}^{new}$;

9 **Learn a theory with an ILP algorithm**
10 $\quad$ `ILP Algorithm(`$\mathbf{E_r}$`)`;

the negative database equal to the empty set.

7.2.2 Contrasting Patterns in Relational Learning

In Algorithm 9 we present the pseudo-code of the *ConFOLding* framework. This architecture uses a new predicate to include the most interesting contrasting patterns, found by the *BeamSouL* algorithm, in the theory learning phase.

7.2.2.1 Find Contrasting Patterns with *BeamSouL*

In the first phase, we run *BeamSouL* to find contrasting patterns in the database. We search for patterns p, in the D_{pos} positive class partition, that satisfy both the anti-monotonic constraint $sup_{pos}(p) \geq \lambda_{pos}$ and the monotonic constraint $sup_{neg}(p) \leq \lambda_{neg}$ in the negative class partition D_{neg} (see line 2 of Algorithm 9). This run of the *BeamSouL* algorithm is called the *positive-run* in the rest of text. Then, we exchange the class partitions and run the *BeamSouL* algorithm to find contrasting patterns in

the negative class partition of the database (see line 3). This run of the *BeamSouL* algorithm is called the *negative-run* in the rest of the text. This way we get two sets of highly discriminative contrasting patterns.

7.2.2.2 Select Interesting Patterns using the Confidence Metric

The previous step usually generates a huge number of contrasting patterns. Thus, to improve the performance of the classifier we need to select a few patterns, keeping only highly discriminative and class related patterns. To do so we use the confidence metric value of each pattern to sort and select, in each class partition, the top k class-related patterns (see line 5).

7.2.2.3 Mapping into a New Predicate

After selecting the top-$2k$ patterns (in a binary problem we select, if available, k patterns in each partition) we build a new ternary relation/predicate

$$logSeqMap(ID, seqNumber, seqMapping).$$

For each example, the *logSeqMap* predicate defines the ID of the example and, for each one of the top-$2k$ sequences ($seqNumber = 1 \ldots, \ldots, 2k$), it maps each sequence (see line 7) into one of two Boolean values: $seqMapping = true$ or $seqMapping = false$. Thus, for each example we get a number of *logSeqMap* facts that equals the number of contrasting sequences found by the *BeamSouL* algorithm. The value of the *seqMaping* argument indicates whether the pattern numbered *seqNumber* subsumes the example number *ID* or not. With this strategy we get an enlarged database that includes the primitive predicates and this new predicate.

7.2.2.4 Learn a Theory

In the last step we take the enlarged database as input to an ILP learner, such as Aleph [Sri03], and learn a FOL theory. Like in the *XMuSer* framework presented in Chapter 7 we allow refinements that include all the predicates, thus, non-temporal, temporal, frequent or infrequent data can be include in the final rule set.

Table 7.1: Relational database used to clarify some details of the *BeamSouL* algorithm and of the *ConFOLding* framework

Patient Label
(Target Table)

Id	*Class*
1	**c**
2	**c**
3	**c**
4	**b**
5	**b**

Blood Analysis

Id	*Date*	*RBC*	*WBC*
1	19720102	high	normal
1	19780203	high	high
2	19770107	normal	low
3	19810107	high	low
3	19830210	high	normal
4	19810107	high	low
5	19810421	low	normal
5	19831017	low	low

Urinalysis

Id	*Date*	*Exam*	*Value*
1	19741201	plt	low
1	19760204	alb	high
2	19800403	plt	low
2	19810301	alb	high
3	19831102	plt	normal
4	19820404	alb	normal
4	19831211	plt	low
5	19841104	plt	low

SexBornDate

Id	*Sex*	*BornDate*
1	m	19520109
2	m	19660301
3	f	19680424
4	m	19750123
5	f	19760213

7.2.3 Illustrative Example

To clarify some details of the *BeamSouL* algorithm and each step of the *ConFOLding* architecture we use the database presented in Table 7.1. This example database is a modified version of the illustrative database used in the previous chapters. Here we split the *Patient Info* table (see Table 5.1 in Chapter 5) into two new tables: table *Patient Label* registers only the ID and the class label of each patient and table *SexBornDate* that registers the sex and the date of birth, of each patient. Moreover, here we add three more patients and remove some non-useful parameters.

Consider that we run the *ConFOLding* system to learn a classification model in the database presented in Table 7.1. Moreover, consider that we set the *ConFOLding* parameters $\lambda_{pos} = 60\%$, $\lambda_{neg} = 40\%$ and $m = 5$.

Consider also that in a pre-processing phase we translate all information in the relational database into a set of Prolog facts. For instance, all information available about patient number one is translated into:

```
patient_label(1, c).
sexBornDate(1, m, 19520109).
blood_analysis(1, 19720102, high, normal).
urinalysis(1, 19741201, plt, low).
urinalysis(1, 19760204, alb, high).
blood_analysis(1, 19780203, high, high).
```

Moreover, consider that the sequence of examinations made by patient one is translated into the SeqLog sequence:

$$q = blood_analysis(1, 19720102, high, normal) \rhd urinalysis(1, 19741201, plt, low) \rhd urinalysis(1, 19760204, alb, high) \rhd blood_analysis(1, 19780203, high, high)$$

Now that we have defined the input and the parameters of the *ConFOLding* system, we run the *BeamSouL* algorithm to find a set of contrasting patterns. First (line 2 of Algorithm 9), the *BeamSouL* algorithm finds all contrasting patterns having at least $\lambda_{pos} = 60\%$ of support in the majority class partition (D_{pos}) and having a support value small than $\lambda_{neg} = 40\%$ in the minority class partition (D_{neg}). Here the D_{pos} class partition is the partition containing patients having hepatitis subtype **c**, the majority class, and the D_{neg} class partition is the partition containing patients having hepatitis subtype **b**. Among the set of patterns ($\mathbf{fs}^{pos}_{patt}$) found in this step, we get pattern

$$s = blood_analysis(A, B, high, C) \rhd blood_analysis(A, D, high, E)$$

that has approximately 66% of support in the hepatitis subtype **c** class partition and 0% support in the hepatitis subtype **b** class partition. This means that this pattern is a subsequence of the SeqLog sequence of patients number one and number three (check that $s \preceq q$), that are both in **c** subtype class partition. In other words, this means that both patient one and three have two consecutive blood analysis examinations with high RBC values. Moreover, the confidence value of this pattern is $Conf_{pos=c} = 2/2$. This means that pattern s is a sequence of two examples in subtype **c** partition and of no other example in the full dataset.

Second (line 3 of Algorithm 9), we exchange the roles of the two database partitions and find patterns that are frequent in the D_{neg} partition and infrequent in the D_{pos}

partition (the $\mathbf{fs}_{patt}^{neg}$ set). Among the patterns found in this step we get the sequence

$$l = blood_analysis(A, B, C, low) \rhd urinalysis(A, D, plt, low)$$

that has 100% support in the hepatitis subtype **b** class partition (D_{neg}) and approximately 33% of support in the hepatitis subtype **c** class partition (D_{pos}). This means that all patients with hepatitis subtype **b** have this sequence of examinations and only one patient with hepatitis subtype **c**, the patient number two, has this sequence of results. The support of this pattern says that patients number two, four and five had a blood analysis test where a low WBC level is observed and, after some time, these same patients made a urine test and get low plt levels. Moreover, the confidence of this pattern is $Conf_b = 2/3$. This means that sequence l is a subsequence of two examples in the subtype **b** class partition and is a subsequence of three examples in the full dataset.

Next (in line 5), we sort the set of patterns $\mathbf{fs}_{patt}^{pos}$ and $\mathbf{fs}_{patt}^{neg}$ using the confidence level of each pattern. In this illustrative example we consider that patterns s and l above were ranked in the top-5 (remember that the *ConFOLding* parameter m was set to 5) positions of each set of patterns and that we find at least five contrasting patterns in each run. For instance, pattern s has the higher confidence value in the $\mathbf{fs}_{patt}^{pos}$ and pattern l has the third best confidence level in the pattern set $\mathbf{fs}_{patt}^{neg}$. Overall, consider that pattern s is ranked in position two and pattern l is ranked in position eight.

Then, we check if patterns s or l subsume all the SeqLog sequences available in the Prolog database (see Section 2.6.2.2 of Chapter 2). For instance, if we consider sequence q above, that describes the examinations made by patient number one, we add ten new logical facts that describe patient number one. We add two logical facts for sequences s and l and eight other facts for each one of the other eight sequences. As sequence s subsumes sequence q ($s \preceq q$) we add the Prolog fact $logSeqMap(1, 2, true)$ to the Prolog database. As sequence l does not subsume sequence q we add the Prolog fact $logSeqMap(1, 8, false)$ to the Prolog database. Thus, after this step the Prolog database contains the following description of patient number one:

```
patient_label(1, c).
sexBornDate(1, m, 19520109).
blood_analysis(1, 19720102, high, normal).
urinalysis(1, 19741201, plt, low).
urinalysis(1, 19760204, alb, high).
blood_analysis(1, 19780203, high, high).
```

```
logSeqMap(1, 2, true).
logSeqMap(1, 8, false).
```

After applying this procedure to all 10 top patterns we get an enlarged database that includes the original database and new facts encoding temporal information.

Last, we run an ILP algorithm, such as Aleph, to learn a set of clauses. One illustrative example of a clause we can find is:

```
patient_label(A,c):-
      sexBornDate(A,m,B),
      urinalysis(A,C,alb,high),
      logSeqMap(A,2,true)
```

This clause covers patients number one and number two in the hepatitis subtype **c** class partition and covers no patient in the hepatitis subtype **b** class partition. It is important to note that in this case we allow a limited number of clause refinements with original predicates that record temporal information and that were explored by the *BeamSouL* algorithm. We do so to include low confidence and infrequent facts in the rule set.

7.3 Experimental Evaluation

In this section we evaluate empirically the *BeamSouL* algorithm and the *ConFOLding* frameworks. First we present the experimental setup that we use in the experiments. Then we present and discuss the results that we obtained.

7.3.1 Configuration of the Experiments

We evaluate the *BeamSouL* algorithm and the *ConFOLding* framework using the four datasets presented in Appendix B. Like in the previous chapters, all algorithms are evaluated using a ten-fold stratified cross-validation procedure. In the Hepatitis and the Financial datasets we define the positive partition to be the majority class. In the protein fold classification task we run a double round robin approach (see Section 2.5.3.3 of Chapter 2).

Using these procedures we compute: the number of frequent, closed and maximal patterns, the peak memory usage and the run-time of the *BeamSouL* algorithm. Regarding the classification framework, we analyze the peak memory usage and the time needed to learn an ILP theory and compute the generalization accuracy.

Moreover, as the classification framework runs the *BeamSouL* algorithm two times to find contrasting patterns in the two partitions, we present the mean and the standard deviation of the number of patterns found and the mean run-time in the two runs.

For comparison purposes, we also evaluate the MineSeqLog algorithm [LR04]. Remember that, if we set the beam size equal to zero, the *BeamSouL* algorithm finds the same patterns as the MineSeqLog algorithm.

Like other combinatorial algorithms, the number of patterns found in the exhaustive mode of the BeamSouL algorithm grows exponentially with the maximum refinement level (the *BeamSouL* l_{ch} parameter). Thus, we run *BeamSouL* in exhaustive search mode to find the maximum refinement level that can give results in an acceptable run-time and without exceeding the maximum memory available in our computer. We run these experiments with the maximum refinement level ranging from zero to five. We test this range of refinement levels to check both the limits of the *BeamSouL* algorithm and of the baseline. In Figure 7.1 we plot the number of frequent, closed and maximal patterns found by the *BeamSouL* algorithm and in Figure 7.2 we present the run-time needed by the *BeamSouL* and the *MineSeqLog* algorithms to find the complete set of contrasting patterns available in the majority partition of the Hepatitis dataset (the positive-run of the *ConFOLding* architecture). In this particular experiment we set the *ConFOLding* parameters $\lambda_{pos} = 0.4$ and $\lambda_{neg} = 0.3$. In Appendix F, Figure F.1 presents the run-time needed to find the complete set of patterns in the minority partition of the Hepatitis dataset (the negative-run of the *ConFOLding* architecture). These figures show the typical behavior of sequence miners. We can see that the number of patterns grows exponential and that for refinement levels higher than four it can be impossible to run the baseline in acceptable run-times. Regarding the peak memory usage, if we run both algorithms in exhaustive mode, the two require almost the same amount of memory. For the maximum refinement level that we evaluate both algorithms we get a peak memory usage of approximately 11 gigabytes of ram memory. As we have a machine with 100 gigabytes the run-time is our major constraint.

As a result of these experiments and to present a fair comparison, we decide to set the maximum level of exhaustive search equal to four ($l_{ch} = 4$) for all the experiments. That is to say that we run the *BeamSouL* algorithm in exhaustive mode until we

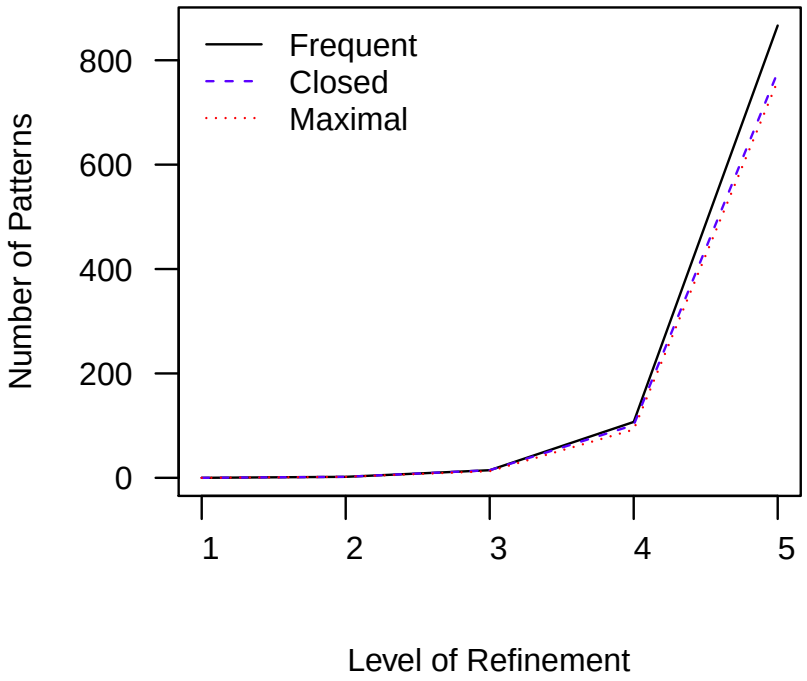


Figure 7.1: Number of patterns found by the *BeamSouL* algorithm in the majority partition of the Hepatitis dataset. We present the number of frequent, closed and maximal patterns found for a range of maximum refinement levels

explore all patterns of level 4. Then, the *BeamSouL* algorithm enters the heuristic phase and runs until no more frequent patterns are found. In each level, after the fifth level, a fixed number of patterns (the beam size) is refined to generate new candidate patterns. It is important to note that we set the MineSeqLog algorithm to explore the same search space that the *BeamSouL* algorithm explores in the exhaustive search phase.

Concerning the support threshold used to find the frequent, the closed and the maximal patterns, we set the thresholds $\lambda_{pos} = 0.8$ and $\lambda_{neg} = 0.4$ for the Financial and Protein datasets. For the Hepatitis dataset we set the thresholds $\lambda_{pos} = 0.4$ and $\lambda_{neg} = 0.3$. If we use the same support values that we use either for the Financial and Protein datasets, both the *BeamSouL* algorithm and the MineSeqLog algorithm fail to find any contrasting patterns. Moreover, we run the *BeamSouL* algorithm for a range of beam sizes. We evaluate *BeamSouL* for k equal to 0, 5, 20, 35 and 50.

For the classification framework, we set the number of patterns to map equal to ten ($m = 10$) for all the three tasks.

As each run of the classification framework includes two runs of the *BeamSouL* al-

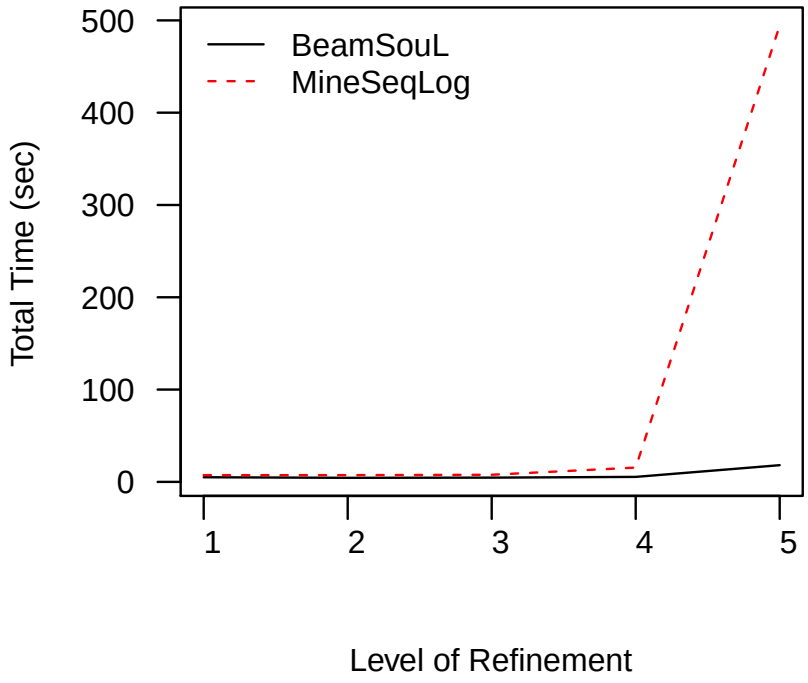


Figure 7.2: Run-time needed by the *BeamSouL* and the MineSeqLog algorithms to find patterns in the majority partition of the Hepatitis dataset. We present the run-time of these algorithms for a range of maximum refinement levels

gorithm, we compute and present the number of patterns and the run-time of the *BeamSouL* algorithm for the two runs. In the first run, the positive-run, we run the *BeamSouL* algorithm to find patterns having support higher than λ_{pos} in the majority class partition and having support value less than λ_{neg} in the negative partition. In the second run, the negative-run, we find contrasting patterns having a support value higher than λ_{pos} in the minority class partition and a support value less than λ_{neg} in the majority class partition.

Furthermore, we compute the p-value of the Wilcoxon hypothesis test to evaluate the contribution of the beam search strategy. We set the baseline to be the MineSeqLog algorithm.

7.3.2 Results

In Table 7.2 we present the mean and standard deviation of the number of patterns found by the *BeamSouL* algorithm, both for the *positive-run* and the *negative-run* of the *ConFOLding* framework. For each dataset we present the results that we obtain

Table 7.2: Number of patterns found in each run of the *BeamSouL* algorithm

Dataset	BeamSize	Positive-Run			Negative-Run		
		Frequent	Closed	Maximal	Frequent	Closed	Maximal
Hepatitis	0	42.3(28.6)	37.6(24.9)	37.6(24.9)	9.5(1.6)	4.5(0.5)	4.5(0.5)
	5	46.6(28.5)	38.4(23.9)	35.4(23.8)	13.5(1.6)	3.5(0.5)	1.5(0.5)
	20	100.5(51.7)	57.8(32.6)	42(25.7)	13.5(1.6)	3.5(0.5)	1.5(0.5)
	35	144.6(86.4)	74.8(44.7)	50.1(31.9)	13.5(1.6)	3.5(0.5)	1.5(0.5)
	50	188.6(106.7)	90.1(51.2)	57.4(40.2)	13.5(1.6)	3.5(0.5)	1.5(0.5)
	MineSeqLog	42.3(28.6)	37.6(24.9)	37.6(24.9)	9.5(1.6)	4.5(0.5)	4.5(0.5)
Financial	0	76.8(1.9)	69.8(1.9)	69.8(1.9)	0(0)	0(0)	0(0)
	5	93.6(3.0)	76.1(2.9)	67.4(2.5)	0(0)	0(0)	0(0)
	20	236.2(30.1)	149.9(22.6)	117.1(15.8)	0(0)	0(0)	0(0)
	35	436.7(50.5)	254.7(22.9)	152.5(9.5)	0(0)	0(0)	0(0)
	50	614.6(49.0)	337.1(18.2)	208.3(30.1)	0(0)	0(0)	0(0)
	MineSeqLog	76.8(1.9)	69.8(1.9)	69.8(1.9)	0(0)	0(0)	0(0)
FinancialB	0	75.9(1.9)	68.9(1.9)	68.9(1.9)	0.2(0.4)	0.2(0.4)	0.2(0.4)
	5	80.2(7.5)	68.9(3.3)	67.6(1.8)	5.3(10.2)	3(7.1)	2.6(6.5)
	20	175.6(3.3)	99.5(1.4)	97.6(1.4)	47.2(42.5)	18.2(18.0)	15.6(15.2)
	35	310.7(5.7)	182.4(6.9)	176.3(2.8)	71.5(72.4)	23.2(24.9)	20.2(21.3)
	50	502.1(24.7)	307.5(19.8)	298(20.3)	86.9(92.1)	29.1(36.3)	24.3(29.7)
	MineSeqLog	75.9(1.9)	68.9(1.9)	68.9(1.9)	0.2(0.4)	0.2(0.4)	0.2(0.4)
Protein	0	0.9(0.2)	0.9(0.2)	0.9(0.2)	11.5(0.4)	10.7(0.4)	10.7(0.4)
	5	48.7(2.8)	33.5(2.2)	12.3(2.1)	20.6(0.9)	14.7(0.8)	9.8(0.5)
	20	184.4(13.6)	116.7(13.0)	39.9(6.8)	107.0(8.7)	52.8(4.1)	27.8(3.5)
	35	345.6(35.6)	210.2(25.0)	74.0(12.0)	173.0(10.8)	82.9(5.0)	45.3(4.4)
	50	469.7(23.6)	280.5(15.8)	93.1(8.5)	242.4(15.5)	119.8(5.8)	66.4(5.8)
	MineSeqLog	0.9(0.2)	0.9(0.2)	0.9(0.2)	11.5(0.4)	10.7(0.4)	10.7(0.4)

Table 7.3: Generalization accuracy of the *ConFOLding* framework

		Frequent		Closed		Maximal	
Dataset	BeamSize	Gen. Acc.	Wilcoxon p-value	Gen. Acc.	Wilcoxon p-value	Gen. Acc.	Wilcoxon p-value
Hepatitis	0	79.49 (7.26)	1	79.49 (7.26)	1	79.26 (7.37)	1
	5	84.13 (6.95)	0.064	84.13 (6.95)	0.049	83.93 (7.03)	0.049
	20	84.13 (7.31)	0.064	84.13 (7.31)	0.049	84.13 (7.31)	0.049
	35	84.52 (7.42)	0.064	84.52 (7.42)	0.049	84.32 (7.22)	0.049
	50	84.32 (7.16)	0.064	84.52 (7.42)	0.049	84.52 (7.13)	0.049
	MineSeqLog	79.49 (7.26)	–	79.49 (7.26)	–	79.26 (7.37)	–
Financial	0	86.76 (1.23)	1	86.76 (1.23)	1	86.76 (1.23)	1
	5	87.17 (1.98)	1	87.17 (1.98)	1	87.17 (1.98)	1
	20	87.17 (1.98)	1	87.17 (1.98)	1	87.17 (1.98)	1
	35	87.17 (1.98)	1	87.17 (1.98)	1	87.17 (1.98)	1
	50	86.76 (1.23)	1	86.76 (1.23)	1	86.76 (1.23)	1
	MineSeqLog	86.76 (1.23)	–	86.76 (1.23)	–	86.76 (1.23)	–
FinancialB	0	61.81 (2.20)	1	61.81 (2.20)	1	61.81 (2.20)	1
	5	61.81 (2.20)	1	61.81 (2.20)	1	61.81 (2.20)	1
	20	63.06 (4.73)	1	61.81 (2.20)	1	61.81 (2.20)	1
	35	61.81 (6.29)	1	60.56 (4.30)	1	60.56 (4.30)	1
	50	61.81 (6.29)	1	60.56 (4.30)	1	60.56 (4.30)	1
	MineSeqLog	61.81 (2.20)	–	61.81 (2.20)	–	61.81 (2.20)	–
Protein	0	62.24 (3.17)	1	62.43 (2.98)	1	62.43 (2.98)	1
	5	67.28 (3.57)	0.002	65.93 (3.53)	0.006	69.57 (3.56)	0.006
	20	66.44 (2.64)	0.018	69.18 (3.88)	0.006	69.75 (3.18)	0.009
	35	67.99 (3.14)	0.006	72.64 (4.49)	0.002	74.41 (2.99)	0.002
	50	69.33 (3.95)	0.01	73.06 (3.14)	0.006	75.27 (3.49)	0.002
	MineSeqLog	62.24 (3.17)	–	62.43 (2.98)	–	62.43 (2.98)	–

by running the *BeamSouL* algorithm with different beam sizes. The results obtained by running the MineSeqLog algorithm are presented in the table row having the beam size equal to *MineSeqLog*.

In Table 7.3 we present the generalization accuracy that we obtained when running our classification framework to learn a discriminative model in the four datasets. For each dataset we present the mean and standard deviation for the three instances of the framework. Remember that each instance of the classification framework is associated with one type of sequence pattern: Frequent, Closed and Maximal. Again, the results obtained with the MineSeqLog algorithm are presented in the row identified with *MineSeqLog* text.

In these tables we can see that by setting the beam size equal to zero, $k = 0$, we obtained the same results as if we run the MineSeqLog algorithm.

In Table F.2 (Appendix F) we present the run-time needed by the *BeamSouL* algorithm to find a subset of patterns. We present the run-time spent in the *positive-run* and *negative-run*. Moreover, we present the run-time of the ILP-based learner. We present the run-time of the three instances of the framework: Frequent, Closed and maximal.

In Table F.1 (Appendix F) we present the peak memory usage for each step of the *ConFOLding* framework. We present the overall (including the *positive-run* and the *negative-run*) memory peak usage of the *BeamSouL* algorithm, the mapping procedure and the ILP-based learner. Again, we present results for the three instance of the framework and for different beam sizes. Moreover, we present the peak memory usage of the reference sequence miner, the MineSeqLog algorithm.

In Table F.3 (Appendix F) we present the size of the theory learned by the Aleph algorithm, the last step of the *ConFOLding* framework. We present the number of rules learned by each one of the three instances of the framework.

In Table F.4 (Appendix F) we present the results that we obtain if we run a propositional classifier in the last step of the *ConFOLding* framework. We present results for two propositional algorithms, the C4.5 and the linear kernel SVM. Moreover we present results when we map ten ($m = 10$) and twenty ($m = 20$) frequent, closed and maximal patterns. In Table F.5 we present the size of the decision trees learned by the C4.5 algorithm.

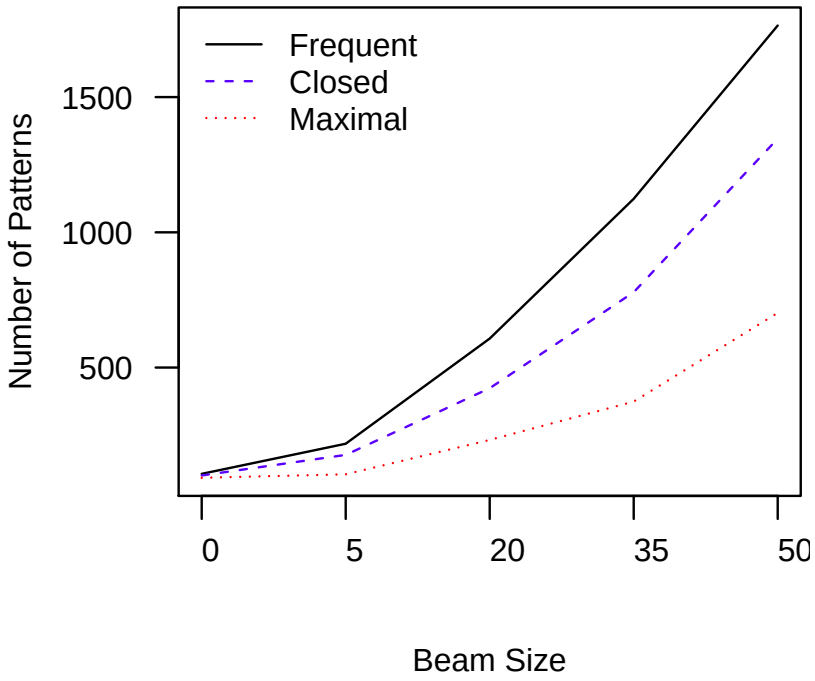


Figure 7.3: Number of patterns found by the *BeamSouL* algorithm in the majority partition of the Hepatitis dataset. We present the number of frequent, closed and maximal patterns found for a range of beam sizes

7.3.3 Analysis of the Results

In Table 7.2 we can see that we get the same number of patterns when running the MineSeqLog algorithm and the *BeamSouL* algorithm with the beam size set equal to zero ($k = 0$). Running the *BeamSouL* algorithm with $k = 0$ means that the algorithm runs only in exhaustive search mode.

If we analyze the output of the *BeamSouL* algorithm for different beam sizes, it is clear that the number of frequent patterns found grows with the size of the beam. The number of frequent patterns has a high growth rate than the growth rate observed in the closed and maximal sequence patterns. This behavior was expected, especially in the case of maximal sequences. As we get longer and low support patterns the likelihood of a maximal pattern subsume a large number of patterns increases. Indeed, the number of findings in each run always obeys to the rank: 1- Frequent, 2- Closed and 3- Maximal. In Figure 7.3 and Figure F.2 (see Appendix F) we highlight this behavior by presenting the number of patterns found in the Hepatitis dataset for a range of beam sizes. In Figure 7.3 we plot the number of patterns found in the majority partition of the Hepatitis dataset. In Appendix F we present Figure F.2 showing the

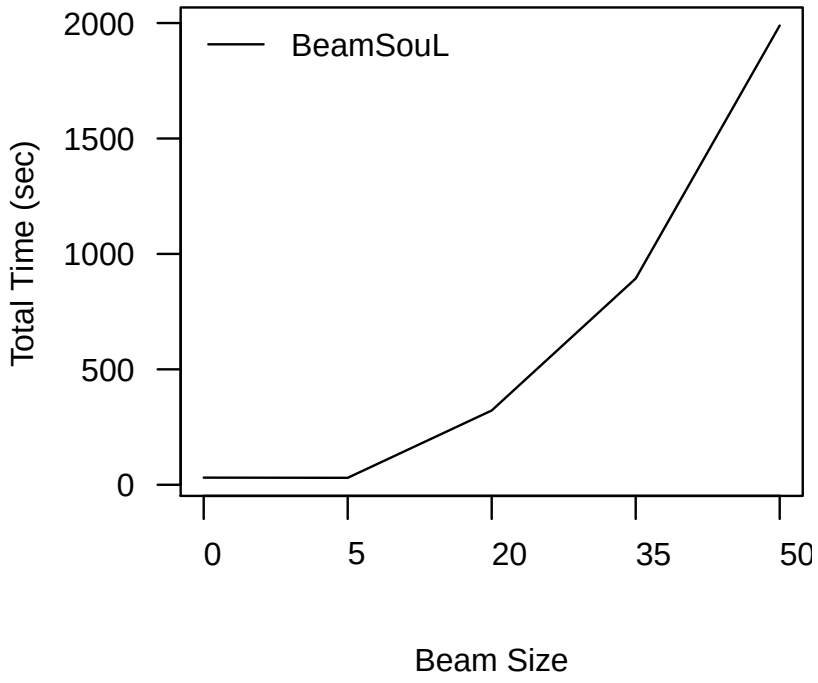


Figure 7.4: Run-time needed by the *BeamSouL* algorithm to find patterns in the majority class of the Hepatitis dataset. We present the run-time of the *BeamSouL* algorithm for different beam sizes

number of patterns found in the minority class.

Moreover, we can see that both the number of closed and maximal patterns can decrease with the increase in the beam size. In the Financial dataset we do not find any frequent pattern in the negative-run. This means that there are no patterns having support higher than 80% in the minority class and less than 40% in the majority class partition. By reducing the λ_{pos} and the λ_{neg} values we can get a higher number of frequent patterns in the majority class. This behavior of pattern mining algorithms is well known and comes with the cost of higher run-times.

By inspecting the peak memory usage presented in Table F.1 and the run-time presented in Table F.2 (both tables presented in Appendix F) we can see that running the BeamSouL algorithm in exhaustive mode requires approximately equal or less computer resources than the MineSeqLog algorithm. Even if we consider the *BeamSouL* with the beam size set to the larger value ($k = 50$), the *BeamSouL* algorithm can be faster than the MineSeqLog algorithm. This behavior is clear when we analyze the results obtained in the Hepatitis and Financial datasets. This is remarkable since we run the *BeamSouL* algorithm in hybrid mode and setting the maximum exhaustive search level equal to the maximum level that we run the MineSeqLog algorithm.

When the *BeamSouL* algorithm leaves exhaustive search we start to explore parts of the solution space that will not be explored by the MineSeqLog algorithm. This advantage of the *BeamSouL* algorithm can be explained by the search strategy used by the MineSeqLog algorithm. Different from the *BeamSouL* algorithm, the MineSeqLog algorithm uses the version spaces search where two borders are needed to find the contrast patterns. The positive-border is obtained by finding all maximal patterns in the positive partition and the negative-border is found by mining the maximal patterns available in the negative partition. To obtain the positive-border the MineSeqLog algorithm needs to find all patterns with support value higher that λ_{pos}. To find the negative-border the MineSeqLog algorithm needs to find all the maximal patterns having support value higher than λ_{neg}. Different from this strategy, the BeamSouL algorithm only needs to find, in both partitions, patterns with support value higher than the highest support value, the λ_{pos} value. In Appendix F we present figures F.3 and F.4 that highlight this behavior. These figures show the run-time needed by the *BeamSouL* algorithm to find frequent patterns in the majority partition (Figure F.3) and the minority partition (Figure F.4) of the Hepatitis dataset.

Regarding the memory requirements, we observe that the peak memory usage increases at a very small rate with the increase in the beam size. In Table F.1 we can see that the peak memory usage of the *BeamSouL* algorithm to find patterns in the Hepatitis dataset range between 11.2 GB and 11.4GB if we set beam size equal to zero and fifty, respectively.

As we can see in Table 7.3 the *ConFOLding* classification framework gains with the large number of long patterns found by the *BeamSouL* algorithm. In all tasks we get high accurate models when we set *BeamSouL*'s beam size to values higher than zero. This is especially clear when we analyze the results obtained in both the Hepatitis and the Protein dataset. In these tasks the generalization accuracy of the *ConFOLding* framework monotonically increases with the value of the beam size. In both tasks we get significant wins over the baseline, the MineSeqLog algorithm. The results that we report in the Hepatitis dataset are the best that we can find in the literature [OYG+04, SBC+13]. It is important to note that in this work we discretize the date of birth attribute using knowledge gathered in the previous work presented in the previous chapters of this thesis.

In the Financial datasets the longer patterns found by the *BeamSouL* algorithm generate small improvements in the generalization accuracy. We get approximately 1% gains. In this particular task we show that gains can be obtained if we replace the FOL learner by a propositional one. In Table F.4 (Appendix F) we present the best results

that we obtained in the Financial task. The generalization accuracy results obtained by the *ConFOLding* framework are at least 10 percentage points better than the ones presented for the *RUSE-WARMR* and the *XMuSer* frameworks. Remember that we could not complete the experiments with the *MuSer* framework and the Hepatitis dataset.

Overall, we get better results than the ones obtained by running the *RUSE-WARMR*, the *MuSer* and the *XMuSer* frameworks presented in the previous chapters. The obtained results are remarkable as we do not include any expert knowledge and do not explore all the information available. An exception to this are the results reported for the protein fold task. In this task we get better results than the ones obtained by the *XMuSer* framework but we get a generalization accuracy loss of 4 percentage points in comparison with the results reported for the MuSer framework (see Table 5.5 in Chapter 5). These results can be explained by the number of long patterns found in both frameworks. In tables 6.6 (see Chapter 7) and 7.3 we can see that both *XMuSer* and *ConFOLding*, respectively, gain with the number of maximal patterns found by the sequence miners.

These results show that long contrasting patterns can be valuable and that a FOL sequence miner is an important contribution in the pattern mining field.

7.4 Conclusions

In this chapter we introduce the first FOL sequence miner that can be tuned to find all or a subset of long logic contrasting patterns available in real datasets. The algorithm employ a levelwise hybrid search strategy and can include user expertise to select the level of regime change, the level where the search method changes from exhaustive search to heuristic search. This way the algorithm can find all contrasting patterns of short size and find a subset of long patterns available in large multi-relational databases. This strategy proved to be valuable when working with real problems. Moreover, the efficiency of the *BeamSouL* algorithm relies on the strategy introduced to compute the support of each pattern in the positive and negative partitions. *BeamSouL* finds contrast patterns by mining only one partition of the data and uses the subsumption operator to obtain the support of each pattern in the other partition. Furthermore, the *BeamSouL* algorithm uses the confidence metric to select the $top - k$ interesting patterns to refine in the next levels of the lattice.

Moreover, we developed the *ConFOLding* classification framework to show that the

inclusion of the *BeamSouL* long patterns can be valuable in multi-relational learning process. By including the *BeamSouL* patterns we can generate high accurate theories.

In the future we will study other techniques to select the patterns in each level of the lattice and to improve the run-time of the *BeamSouL* algorithm. Moreover, we will extend the *BeamSouL* algorithm to find sequence patterns in data streams.

Chapter 8

Conclusions

In this chapter we summarize the work developed in this thesis. We present our main contributions, discuss the major issues of this work and present future research directions.

8.1 Summary

In this work we study methodologies to explore temporal patterns available in multi-relational databases. In particular, we search for methods that can learn highly expressive patterns and theories from non-temporal and temporal data and using logico-relational information.

We developed a set of frameworks and algorithms that explore propositional and/or relational methods. We run propositional and first-order logic (FOL) algorithms both to find sequences and to learn classification theories. Moreover, we explore propositionalization and predicate invention techniques to learn either propositional or FOL classification theories.

The first contribution is the *MuSer propositional* framework aimed to explore temporal information using well-know and efficient propositional methodologies. At the core of this framework we have a new encoding methodology that translates the temporal information available in a multi-relational database into a propositional sequence database that is suitable to be explored by a propositional sequence miner. Moreover, we make use of propositionalization technique to induce a propositional classification model.

Our second contribution, the *XMuSer* framework, combines propositional and relational techniques to learn a FOL classification theory. In this framework we use the encoding methodology introduced in the *MuSer* framework and run a propositional predictive sequence miner that can find a constrained set of temporal patterns. We find a constrained set of patterns and use predicate invention technique to learn a FOL theory from non-temporal and evolving data. Moreover, the learned clause set may include frequent and infrequent temporal data.

The third contribution is the *BeamSouL* algorithm, an ILP based algorithm that uses the SeqLog representation formalism to represent and explore sequences of logical atoms. We designed the *BeamSouL* algorithm to find all or a subset of long contrasting patterns. Its efficiency relies in a matching procedure that eliminates the need to find low support patterns in the negative partition and in a hybrid search strategy. By using a beam search procedure the algorithm can find long size patterns impossible to find with the existing methods. Different from the two previous contributions, this algorithm finds FOL patterns.

The fourth contribution is the *ConFOLding* framework, it was developed to explore the discriminative power of contrasting patterns found by the *BeamSouL* algorithm.

All these proposals were evaluated empirically with four real databases. Moreover, we compare our proposals with a set of algorithms. In this set we include the *RUSE-WARMR* framework that we develop in this thesis. The *RUSE-WARMR* architecture uses arithmetic aggregation operators and a FOL association rule miner to explore temporal data. This way, and like many other algorithms, the *RUSE-WARMR* does not properly explore temporal data.

Moreover, as all our proposals generate at some point a large number of patterns, we analyze feature selection techniques and introduced the *rankCI* methodology to select features in a rank order space induced by an ensemble of metrics. Each metric induces a rank over the features and then a user-defined number of patterns is selected in each induced rank.

8.2 Findings

In this thesis we obtained two types of results. We obtained new results that indicate new research directions and a set of results that are consistent with general thoughts introduced in other fields of science.

Concerning new results that indicate future research directions:

- Exploring temporal data available in multi-relational databases

  We showed that significant gains can be obtained, in terms of expressiveness and accuracy, if we explore temporal data conveniently. In general, MRDM methodologies ignore information about the order of the events and use aggregation strategies that are more suitable to deal with non-temporal data. Here we compare the performance of the *RUSE-WARMR* framework that finds FOL association rules with the performance of frameworks especially developed to explore temporal data (*MuSer*, *XMuSer* and *ConFOLding* frameworks). With the latter systems we can either find propositional or FOL sequence patterns. Moreover, we showed that significant gains can be obtained if we include temporal patterns in a learning process. In *XMuSer* and *ConFOLding* frameworks we use predicate invention technique to include both non-temporal data and temporal patterns in an ILP theory learning task.

- Finding long discriminative patterns

  The long contrasting patterns found by the *BeamSouL* algorithm prove to be valuable. Adding to the undeniable value of a long contrasting pattern, the experiments with the *ConFOLding* framework show that, in some domains, by including these long patterns in a learning task we can obtain significant gains in terms of accuracy. The long patterns found by the *BeamSouL* are highly discriminative and can not be found by existing FOL sequence miners.

  With the *BeamSouL* algorithm we can find contrasting patterns without having to run a sequence miner with low support values in one of the partitions. This highly expensive operation is replaced by a less time consuming subsumption operation. Moreover, the hybrid search strategy that we use in the *BeamSouL* algorithm, that combines exhaustive search with beam search, can be tuned to tradeoff search completeness and running time. The parameter configuration we define to run *BeamSouL* is domain dependent. In simple and small databases/problems we can run *BeamSouL* in exhaustive mode to find all patterns. In highly complex domains we can only run *BeamSouL* in beam search mode. Like many other ILP based algorithms, the configuration of the *BeamSouL* algorithm requires a prior investigation of database complexity and domain knowledge available.

  Furthermore, the *BeamSouL* algorithm can combine other evaluation metrics (current version of *BeamSouL* uses the confidence metric) to find patterns that

are considered more adequate to solve a given task.

- Finding the top interesting patterns

  It is well known that sequence miners find a large number of redundant and irrelevant patterns. Thus, to eliminate these patterns and keep only the most interesting in most of the frameworks we select a constrained subset of patterns. We run the frameworks to select frequent, closed and maximal patterns. Overall, we can say that we obtain high accuracy when we map closed patterns. With this type of patterns we eliminate subpatterns that have the same support as one super-pattern and yet we can access the support value of each frequent pattern.

  Moreover, we modify the cSpade algorithm to output the top most interesting discriminative patterns. By using this strategy we eliminate the need of an additional post-processing filter.

  Furthermore, we develop a methodology to find discriminative patterns in parallel. In the *RUSE-WARMR*, *MuSer* and *ConFOLding* frameworks we run a sequence miner to find patterns in each class partition. Then a matching operation is used to eliminate non-discriminative patterns. By using these strategies we get a large number of discriminative patterns and important reductions in terms of computer memory usage.

The second type of results are consistent with *expressiveness of theories comes at expense of higher computational costs* and that *there is no best overall feature selection methodology.*

- Expressiveness of patterns and theories

  We can now say that the frameworks that find propositional temporal patterns are more efficient in terms of memory and run-time but are unable to find highly expressive patterns. This is the case of the propositional sequence miners used in the *MuSer* and *XMuSer* frameworks developed in this work. On the other hand, the *ConFOLding* framework can find highly expressive patterns but requires high processing time and memory usage. In large real databases the FOL frameworks can run for several months and fail to explore the complete solution space. This behavior becomes apparent when we run algorithms like MineSeqLog, that (try to) search the complete solution space to find FOL patterns.

  This behavior is also apparent in theory learning time. We can see this when we analyze the time and memory usage when we learn propositional and FOL theories, respectively, in *MuSer* and *XMuSer* frameworks.

- Feature selection

  We develop the *rankCI* framework in the very beginning of this work to study feature selection techniques. We confirm that there is no overall best evaluation criterion. In this thesis we showed that a feature selection method that uses an ensemble of evaluation metrics generates potentially highly discriminative patterns. Moreover, we found that the number and type of features used in the learning process can determine the performance of the learner. If we use a large number of features the learner may overfit the data.

8.3 Limitations

We have two main difficulties. First, we would like to evaluate our proposals in a large number of real databases but there are few real multi-relational datasets available that register temporal data. Second, the time needed to run the experiments makes impossible to present an in-depth analysis of all the parameters of our frameworks and algorithms. This last issue becomes even harder when we run the *BeamSouL* competitor, the MineSeqLog algorithm.

Moreover, we present a set of methodologies that use different representation formalisms and bias that makes impossible to present a fair comparison. The results are always dependent on the bias used. We have algorithms like the *RUSE-WARMR* that use aggregation strategies to explore temporal patterns. Other like *MuSer* and *XMuSer* explore temporal patterns using propositional calculus. The *ConFOLding* framework finds highly expressive FOL patterns and, like the *XMuSer* framework, can learn FOL theories that include expert knowledge.

8.4 Future Work

Currently we observe an ever-growing need to explore large amounts of data stored in multi-relational databases. Thus, there is a need to develop new methodologies that can explore this type of data. With this kind of data it is impossible to explore the full solution space, even if we use propositional representation and propositional miners. Thus, there is a need to update the existing propositional sequence miners to include a hybrid search strategy like the one we introduce in the *BeamSouL* algorithm. Concerning the existing ILP based algorithms, there is a need to study high efficient

divide-and-conquer strategies to replace the candidate-generation methodology used by algorithms such as *BeamSouL*. The pattern-growth strategy used by the PrefixSpan propositional sequence miner is a high efficient and promising strategy. Moreover, there is a need to develop incremental sequence miners, both propositional and FOL miners, that can explore stream data.

Another research line is the development of new propositionalization strategies that can adapt to concept drift. We will explore mapping strategies that map a non-fixed subset of patterns and that use non-Boolean attributes.

Furthermore, we will parallelize most of the frameworks and algorithms that we introduce in this work. Most of the architectures include pieces of code that run independent of each other and the workload can easily be distributed among the cores available in the new multi-core platforms. For instance, we can run in parallel two instances of a sequence miner to find frequent patterns in a binary classification problem using the *MuSer* or the *ConFOLding* frameworks. This way we can explore parts of search space that are now impossible to reach and find longer patterns available in highly complex databases.

References

[ADT95] Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, 1995.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, Santiago de Chile, Chile, 1994. Morgan Kaufmann.

[AS95] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995.

[Bay98] Roberto J. Bayardo. Efficiently mining long patterns from databases. *SIGMOD Record*, 27(2):85–93, June 1998.

[BDD+00] Hendrik Blockeel, Luc Dehaspe, Bart Demoen, Gerda Janssens, Jan Ramon, and Henk Vandecasteele. Executing query packs in ilp. In *ILP*, pages 60–77, 2000.

[BM98] Eric Bloedorn and Ryszard S. Michalski. Data-driven constructive induction. *IEEE Intelligent Systems*, 13(2):30–37, 1998.

[BMS97] Sergey Brin, Rajeev Motwani, and Craig Silverstein. Beyond market baskets: Generalizing association rules to correlations. In Joan Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 265–276. ACM Press, 1997.

[BP01] Stephen D. Bay and Michael J. Pazzani. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery*, 5:213–246, 2001.

[BR98] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[BS03] Hendrik Blockeel and Michele Sebag. Scalability and efficiency in multi-relational data mining. *SIGKDD Explorations*, 5(1):17–30, 2003.

[CB91] Peter Clark and Robin Boswell. Rule induction with cn2: Some recent improvements. In Yves Kodratoff, editor, *Machine Learning EWSL-91*, volume 482 of *Lecture Notes in Computer Science*, pages 151–163. Springer Berlin Heidelberg, 1991.

[CRD12] Vítor S. Costa, Ricardo Rocha, and Luís Damas. The yap prolog system. *TPLP*, 12(1-2):5–34, 2012.

[dAF07] Sandra de Amo and Daniel A. Furtado. First-order temporal pattern mining with regular expression constraints. *Data & Knowledge Engineering*, 62(3):401–420, 2007. Including special issue: 20th Brazilian Symposium on Databases (SBBD 2005).

[DBD+05] Jesse Davis, Elizabeth Burnside, Inês C. Dutra, David Page, and Vítor S. Costa. An integrated approach to learning bayesian networks of rules. In *Proceedings of the 16th European conference on Machine Learning*, ECML'05, pages 84–95, Berlin, Heidelberg, 2005. Springer-Verlag.

[Dom12] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[DT99] Luc Dehaspe and Hannu Toivonen. Discovery of frequent datalog patterns. *Data Mining Knowledge Discovery*, 3(1):7–36, March 1999.

[Dž03] Sašo Džeroski. Multi-relational data mining: an introduction. *SIGKDD Explor. Newsl.*, 5(1):1–16, July 2003.

[DZ09] Kang Deng and Osmar R. Zaïane. Contrasting sequence groups by emerging sequences. In João Gama, Vítor S. Costa, Alípio Jorge, and Pavel Brazdil, editors, *Discovery Science*, volume 5808 of *Lecture Notes in Computer Science*, pages 377–384. Springer, 2009.

[EMBF09] Floriana Esposito, Nicola Di Mauro, Teresa M. A. Basile, and Stefano Ferilli. Multi-dimensional relational sequence mining. *Fundamenta Informaticae*, 89(1):23–43, 2009.

[Emd94] Werner Emde. Inductive learning of characteristic concept descriptions from small sets of classified examples. In *Proceedings of the European conference on machine learning on Machine Learning*, ECML-94, pages 103–121, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[FG07] Carlos A. Ferreira and João Gama. Rank ensemble features for constructive induction. In *Proceedings of the Workshop on General Artificial Intelligence, in the 13th Portuguese Conference on Artificial Intelligence (EPIA)*, pages 45–57, Guimarães, Portugal, 2007.

[FGC08] Carlos A. Ferreira, João Gama, and Vítor S. Costa. RUSE-WARMR: Rule Selection for Classifier Induction in Multi-relational Data-Sets. In *20th IEEE International Conference on Tools with Artificial Intelligence*, volume 1, pages 379–386, Dayton, Ohio, USA, 2008. IEEE Computer Society.

[FGC10] Carlos A. Ferreira, João Gama, and Vítor S. Costa. Sequential pattern mining in multi-relational datasets. In *Proc. of the 13th Conference of the Spanish Association for Artificial Intelligence, Lecture Notes in Computer Science*, volume 5988, pages 121–130, Seville, Spain, 2010. Springer Heidelberg.

[FGC11] Carlos A. Ferreira, João Gama, and Vítor S. Costa. Constrained sequential pattern knowledge in multi-relational learning. In Luis Antunes and Helena S. Pinto, editors, *Progress in Artificial Intelligence, 15th Portuguese Conference on Artificial Intelligence (EPIA)*, volume 7026 of *Lecture Notes in Computer Science*. Springer, 2011.

[FGC12a] Carlos A. Ferreira, João Gama, and Vítor S. Costa. Predictive sequence miner in ilp learning. In *Inductive Logic Programming*, volume 7207 of *Lecture Notes in Computer Science*, pages 130–144. Springer Berlin Heidelberg, 2012.

[FGC12b] Carlos A. Ferreira, João Gama, and Vítor S. Costa. Predictive sequence miner in ilp learning. In *Proceedings of the 21st Inductive Logic Programming Conference*, Lecture Notes in Computer Science, pages 130–144, Windsor Great Park, United Kingdom, 2012. Springer.

[FGC14] Carlos A. Ferreira, João Gama, and Vítor S. Costa. Contrasting logical sequences in multi-relational learning. *Data Mining and Knowledge Discovery Journal*, (under review), 2014.

[FGMB10] Carlos A. Ferreira, João Gama, Vladimiro Miranda, and Audun Botterud. A survey on wind power ramp forecasting. In *Report ANL/DIS 10-13*, Argonne National Laboratory, 2010.

[FGMB13] Carlos A. Ferreira, João Gama, Vladimiro Miranda, and Audun Botterud. Probabilistic ramp detection and forecasting for wind power prediction. In *Reliability and Risk Evaluation of Wind Integrated Power Systems*, pages 29–44. Springer, 2013.

[FL04] Peter Flach and Nicolas Lachiche. Naive bayesian classification of structured data. *Machine Learning*, 57(3):233–269, 2004.

[Fla99] Peter Flach. Knowledge representation for inductive learning. In Anthony Hunter and Simon Parsons, editors, *ESCQARU*, volume 1638 of *Lecture Notes in Computer Science*, pages 160–167, London, United Kingdom, 1999. Springer.

[Für02] Johannes Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2:721–747, 2002.

[GB00] João Gama and Pavel Brazdil. Cascade generalization. *Machine Learning*, 41(3):315–343, December 2000.

[GH06] Liqiang Geng and Howard J. Hamilton. Interestingness measures for data mining: A survey. *ACM Computing Surveys*, 38(3), September 2006.

[GMUW09] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009.

[GRS02] Minos Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Mining Sequential Patterns with Regular Expression Constraints. *IEEE Trans. on Know. and Data Eng.*, pages 223–234, 2002.

[HKP11] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *SIGMOD Record*, 29(2):1–12, May 2000.

[JB03] Aleks Jakulin and Ivan Bratko. Analyzing attribute dependencies. In *PKDD 2003, volume 2838 of LNAI*, pages 229–240. Springer-Verlag, 2003.

[KB97] Aram Karalić and Ivan Bratko. First order regression. *Machine Learning*, 26:147–176, 1997.

[KHS01] Arno J. Knobbe, Marc Haas, and Arno Siebes. Propositionalisation and aggregates. In *In Proceedings of the Fifth European Conference on Principles of Data Mining and Knowledge Discovery*, 2001.

[Kle52] Stephen C. Kleene. *Finite axiomatizability of theories in the predicate calculus using additional predicate symbols.* Number 10 in Two Papers on the Predicate Calculus. Memoirs of the American Mathematical Society. American Mathematical Society, 1952.

[KLF00] Stefan Kramer, Nada Lavrač, and Peter Flach. Propositionalization approaches to relational data mining. In Sašo Džeroski, editor, *Relational Data Mining*, pages 262–286. Springer-Verlag New York, Inc., New York, NY, USA, 2000.

[Kno05] Arno J. Knobbe. *Multi-Relational Data Mining*, volume Frontiers in Artificial Intelligence and Applications 145. IOS Press, 2005.

[Koh95] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[Kra95] Stefan Kramer. Predicate invention: A comprehensive view. Technical Report OFAI-TR-95-32, Austrian Research Institute for Artificial Intelligence, Vienna, 1995.

[KRH01] Stefan Kramer, Luc De Raedt, and Christoph Helma. Molecular feature mining in hiv data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 136–143, New York, NY, USA, 2001. ACM.

[KRR06] Kristian Kersting, Luc De Raedt, and Tapani Raiko. Logical hidden markov models. *Journal of Artificial Intelligence Research (JAIR)*, 25:425–456, 2006.

[KRZ+03a] Mark-A. Krogel, Simon Rawles, F. Zelezný, Peter Flach, Nada Lavrač, and Stefan Wrobel. Comparative evaluation of approaches to propositionalization. In Tamás Horváth, editor, *Inductive Logic Programming,*

13th International Conference on Inductive Logic Programming (ILP–2003), volume 2835 of *Lecture notes in computer science*, pages 197–214. Springer Verlag, 2003.

[KRZ+03b] Mark-A. Krogel, Simon Rawles, Filip Zelezný, Peter Flach, Nada Lavrač, and Stefan Wrobel. Comparative evaluation of approaches to propositionalization. In *In Proceedings of the 13th International Conference on Inductive Logic Programming*, pages 197–214. Springer-Verlag, 2003.

[KW01] Mark-A. Krogel and Stefan Wrobel. Transformation-Based Learning Using Multirelational Aggregation. In C. Rouveirol and M. Sebag, editors, *Proceedings of the Eleventh International Conference on Inductive Logic Programming (ILP)*, 2001.

[LD93] Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming: Techniques and Applications*. Routledge, New York, NY, 1993.

[LDG91] Nada Lavrač, Sašo Džeroski, and Marko Grobelnik. Learning nonrecursive definitions of relations with linus. In Yves Kodratoff, editor, *Machine Learning EWSL-91*, volume 482 of *Lecture Notes in Computer Science*, pages 265–281. Springer Berlin Heidelberg, 1991.

[LDN99] Nada Lavrač, Sašo Džeroski, and Masayuki Numao. Inductive logic programming for relational knowledge discovery. *New Generation Computing*, 17(1):3–23, 1999.

[LHM98] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *KDD*, pages 80–86, 1998.

[Lin91] Charles X. Ling. Inventing necessary theoretical terms in scientific discovery and inductive logic programming. Technical Report 302, Department of Computer Science, University of Western Ontario, 1991.

[LKR07] Niels Landwehr, Kristian Kersting, and Luc De Raedt. Integrating naïve bayes and foil. *Journal of Machine Learning Research*, 8:481–507, 2007.

[LMVL08] Philippe Lenca, Patrick Meyer, Benoît Vaillant, and Stéphane Lallich. On selecting interestingness measures for association rules: User oriented description and multiple criteria decision aid. *European Journal of Operational Research*, 184(2):610–626, 2008.

[LPRF06] Niels Landwehr, Andrea Passerini, Luc De Raedt, and Paolo Frasconi. kfoil: Learning simple relational kernels. In *AAAI*, pages 389–394, 2006.

[LR00] Wim Van Laer and Luc De Raedt. How to upgrade propositional learners to first order logic: case study. In Sašo Džeroski, editor, *Relational Data Mining*, pages 235–256, New York, NY, USA, 2000. Springer-Verlag New York, Inc.

[LR04] Sau D. Lee and Luc De Raedt. Constraint based mining of first order sequences in seqlog. In *Database Support for Data Mining Applications: Discovering Knowledge with Inductive Queries, Lecture Notes in Computer Science*, volume 2682, pages 155–176. Springer-Verlag Berlin, 2004.

[LZO98] Neal Lesh, Mohammed J. Zaki, and Mitsunori Ogihara. Mining features for sequence classification. Technical Report TR98-22, MERL - Mitsubishi Electric Research Laboratories, 201 Broadway, Cambridge, MA 02139, December 1998.

[MBFE11] Nicola Mauro, Teresa M. A. Basile, Stefano Ferilli, and Floriana Esposito. Optimizing probabilistic models for relational sequence learning. In Marzena Kryszkiewicz, Henryk Rybinski, Andrzej Skowron, and ZbigniewW. Ra, editors, *Foundations of Intelligent Systems*, volume 6804 of *Lecture Notes in Computer Science*, pages 240–249. Springer Berlin Heidelberg, 2011.

[MF90] Stephen Muggleton and Cao Feng. *Efficient Induction Of Logic Programs*. Academic Press, 1990.

[MHM+99] Takashi Matsuda, Tadashi Horiuchi, Hiroshi Motoda, Takashi Washio, Kohei Kumazawa, and Naohide Arai. Graph-based induction for general graph structured data. In *Discovery Science, Lecture Notes in Computer Science*, volume 5808, pages 340–342, Porto, Portugal, 1999.

[Mic93] Ryszard S. Michalski. Inferential theory of learning as a conceptual basis for multistrategy learning. *Machine Learning*, 11(2-3):111–151, 1993.

[MMYW02] Takashi Matsuda, Hiroshi Motoda, Tetsuya Yoshida, and Takashi Washio. Knowledge discovery from structured data by beam-wise graph-based induction. In *Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence*,

PRICAI '02, pages 255–264, London, United Kingdom, 2002. Springer-Verlag.

[MR94] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19:629–679, 1994.

[Mug95] Stephen Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3&4):245–286, 1995.

[NCdW96] Shan-Hwei Nienhuys-Cheng and Ronald de Wolf. Least generalizations and greatest specializations of sets of clauses. *Journal of Artificial Intelligence Research*, 4:341–363, 1996.

[NK01] Siegfried Nijssen and Joost Kok. Faster association rules for multiple relations. In *Proceedings of the 17th international joint conference on Artificial intelligence - Volume 2*, IJCAI'01, pages 891–896, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[NK03] Siegfried Nijssen and Joost N. Kok. Efficient frequent query discovery in farmer. In *In Proceedings of the 7th PKDD, volume 2838 of Lecture Notes in Computer Science*, pages 350–362. Springer, 2003.

[NLW09] Petra K. Novak, Nada Lavrač, and Geoffrey I. Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal Machine Learning Research*, 10:377–403, June 2009.

[OYG+04] Kouzou Ohara, Tetsuya Yoshida, Warodom Geamsakul, Hiroshi Motoda, Takashi Washio, Hideto Yokoi, and Katsuhiko Takabayashi. Analysis of hepatitis dataset by decision tree graph-based induction. In *Proceedings of Discovery Challenge*, pages 173–184, 2004.

[PH90] Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5(1):71–99, May 1990.

[PHMA+01] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. *ICDE*, pages 215–224, 2001.

[Pla98] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING, 1998.

[Plo70] Gordon D. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5, 1970.

[QCJ95] J. Ross Quinlan and R. M. Cameron-Jones. Induction of Logic Programs: FOIL and Related Systems. *New Generation Computing*, pages 287–312, 1995.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[R D11] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011.

[Rae08] Luc De Raedt. *Logical and Relational Learning*. Cognitive Technologies. Springer, 2008.

[RD97] Luc De Raedt and Luc Dehaspe. Clausal discovery. *Machine Learning*, 26(2-3):99–146, 1997.

[RGF12] Melissa Rodrigues, João Gama, and Carlos A. Ferreira. Identifying relationships in transactional data. In Juan Pavón, Néstor D. Duque-Méndez, and Rubén Fuentes-Fernández, editors, *IBERAMIA*, volume 7637 of *Lecture Notes in Computer Science*, pages 81–90. Springer, 2012.

[Ris78] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465 – 471, 1978.

[RTL09] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-Validation. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 532–538. Springer US, 2009.

[RY01] Dan Roth and Wen-T Yih. Relational learning via propositional algorithms: An information extraction case study. In Bernhard Nebel, editor, *IJCAI*, pages 1257–1263. Morgan Kaufmann, 2001.

[SA96] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In Peter Apers,

Mokrane Bouzeghoub, and Georges Gardarin, editors, *Advances in Database Technology EDBT '96*, volume 1057 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 1996.

[SA10] Andreia Silva and Cláudia Antunes. Pattern mining on stars with fp-growth. In Vicen Torra, Yasuo Narukawa, and Marc Daumas, editors, *MDAI*, volume 6408 of *Lecture Notes in Computer Science*, pages 175–186. Springer, 2010.

[SA12] Andreia Silva and Cláudia Antunes. Mining patterns from large star schemas based on streaming algorithms. In Roger Lee, editor, *Computer and Information Science 2012*, volume 429 of *Studies in Computational Intelligence*, pages 139–150. Springer Berlin Heidelberg, 2012.

[SBC+13] Oliver Schulte, Bahareh Bina, Branden Crawford, Derek Bingham, and Yi Xiong. A hierarchy of independence assumptions for multi-relational bayes net classifiers. In *CIDM*, pages 150–159. IEEE, 2013.

[SMB92] Ashwin Srinivasan, Stephen Muggleton, and Michael Bain. Distinguishing exceptions from noise in non-monotonic learning. In *In Proceedings of the 2nd International Workshop on Inductive Logic Programming*, pages 97–107, 1992.

[SP91] Glenn Silverstein and Michael J. Pazzani. Relational cliches: Constraining constructive induction during relational learning. In L. Birnbaum and G. Collins, editors, *ML91*, pages 203–207. MK, 1991.

[Sri03] Ashwin Srinivasan. The Aleph Manual, 2003.

[Sta93] Irene Stahl. Predicate Invention in ILP - an Overview. In *Proceedings of the European Conference on Machine Learning (ECML'93)*, volume 667 of *Lecture Notes in Computer Science*, pages 311–322. Springer-Verlag, 1993.

[Sta96] Irene Stahl. Predicate invention in inductive logic programming. In Luc De Raedt, editor, *Advances in Inductive Logic Programming*, pages 34–47. IOS Press, 1996.

[SVRH05] Douglas J. Slotta, John P. Vergara, Naren Ramakrishnan, and Lenwood S. Heath. Algorithms for feature selection in rank-order spaces, 2005.

[TKS02] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right interestingness measure for association patterns. In *KDD*, pages 32–41, 2002.

[TWM+02] Masahiro Terabe, Takashi Washio, Hiroshi Motoda, Osamu Katai, and Tetsuo Sawaragi. Attribute generation based on association rules. *Knowledge and Information Systems*, 4(3):329–349, 2002.

[VD04] Giorgio Valentini and Thomas G. Dietterich. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. *Journal of Machine Learning Research*, 5:725–775, December 2004.

[WF99] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[WM94] Janusz Wnek and Ryszard S. Micbalski. Hypothesis-driven constructive induction in aq17-hci: A method and experiments. *Machine Learning*, 14:139–168, 1994.

[WZ05] Geoffrey I. Webb and Songmao Zhang. K-optimal rule discovery. *Data Mining Knowledge Discovery*, 10(1):39–79, 2005.

[YH05] Xiaoxin Yin and Jiawei Han. Efficient classification from multiple heterogeneous databases. In *Knowledge Discovery in Databases: PKDD 2005, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, Lecture Notes in Computer Science*, volume 3721, pages 404–416, Porto, Portugal, 2005. Springer.

[YHA03] Xifeng Yan, Jiawei Han, and Ramin Afshar. CloSpan: Mining Closed Sequential Patterns in Large Datasets. *SDM*, pages 166–177, 2003.

[YHYY04] Xiaoxin Yin, Jiawei Han, Jiong Yang, and Philip S. Yu. Crossmine: Efficient classification across multiple database relations. In *Proceedings of the 20th International Conference on Data Engineering*, pages 399–411, Boston, MA, USA, 2004. IEEE Computer Society.

[YL03] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *20th International Conference on Machine Learning*, pages 856–863, 2003.

[YL04] Lei Yu and Huan Liu. Efficient feature selection via analysis of relevance and redundancy. *Journal of Machine Learning Research*, 5:1205–1224, 2004.

[Zak00] Mohammed J. Zaki. Sequence mining in categorical domains: Incorporating constraints. In *CIKM*, pages 422–429, 2000.

[Zak01] Mohammed J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 1(42):31–60, 2001.

[Zhe98] Zijian Zheng. Constructing x-of-n attributes for decision tree learning. In *Machine Learning*, pages 1–43. Kluwer Academic, 1998.

[ZL06] Filip Železný and Nada Lavrač. Propositionalization-based relational subgroup discovery with rsd. *Machine Learning*, 62(1-2):33–63, 2006.

Appendix A

DNF Formulae

dnf0: $(x_1 \wedge x_2) \vee (x_3 \wedge \bar{x_4} \wedge x_5)$;

dnf1: $(x_1 \wedge x_2 \wedge x_3 \wedge x_8) \vee (x_3 \wedge x_5 \wedge x_6 \wedge \bar{x_7}) \vee (x_1 \wedge \bar{x_2} \wedge \bar{x_6} \wedge \bar{x_7}) \vee (\bar{x1} \wedge x_4 \wedge x_2 \wedge \bar{x_6}) \vee (x_1 \wedge x_2 \wedge x_3 \wedge x_4) \vee (\bar{x_5} \wedge \bar{x_6} \wedge x_7 \wedge \bar{x_8})$;

dnf2: $(x_1 \wedge x_2 \wedge x_3 \wedge x_8 \wedge \bar{x_9} \wedge \bar{x_{10}}) \vee (x_2 \wedge x_3 \wedge x_5 \wedge \bar{x_6} \wedge \bar{x_7} \wedge x_9) \vee (x_1 \wedge \bar{x_4} \wedge \bar{x_8} \wedge \bar{x_9} \wedge x_{10}) \vee (x_2 \wedge x_5 \wedge \bar{x_6} \wedge x_8) \vee (x_5 \wedge \bar{x_7} \wedge x_8 \wedge x_9 \wedge \bar{x_{10}}) \vee (x_3 \wedge \bar{x_4} \wedge \bar{x_5} \wedge \bar{x_7})$;

dnf3: $(\bar{x_1} \wedge \bar{x_2} \wedge x_5 \wedge x_8) \vee (\bar{x_4} \wedge \bar{x_6} \wedge \bar{x_7}) \vee (\bar{x_3} \wedge x_6 \wedge \bar{x_8}) \vee (x_1 \wedge x_7 \wedge x_8) \vee (\bar{x_1} \wedge \bar{x_2} \wedge \bar{x_3} \wedge x_8) \vee (x_5 \wedge x_6 \wedge \bar{x_7} \wedge x_8)$;

Appendix B

Datasets

In this section we present the datasets that we used to evaluate the algorithms and frameworks that we develop in this thesis. We present the Hepatitis, the Financial, a balanced version of the Financial, and the Protein multi-relational datasets.

Both the Hepatitis and the Financial datasets are available to download at the PKDD challenge web page[1]. The Protein dataset was already used in [MBFE11].

B.1 Hepatitis

The Hepatitis dataset records long term monitoring, from 1982 to 1990, of patients having hepatitis B and C subtypes. One table provides personal data about patients. The other tables record blood and urinalysis examinations. The task that we address in this work is to discriminate between patients having hepatitis subtype B and C. In these experiments we select a subset of tables and performed limited feature selection. We explore table *pt_e030704*, that records basic information about the patients; table *bio_e030704*, that records the results of biopsy made by each patient; table *ilab_e030704* that records information about measurements in in-hospital examinations; table *labn_e030704*, that records information about measurements in in-hospital examinations and table *hemat_e030704*, that contains results on hematological analysis. All the frameworks and algorithms presented in this work explore temporal data from tables *ilab_e030704* and *hemat_e030704*. Regarding the feature selection, we explore all information available in tables *pt_e030704* and *hemat_e030704* and select

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

only the GOT, GPT, TTT, ZTT, T-CHO, CHE, ALB, TP, T-BIL, D-BIL, I-BIL, ICG-15, PLT, WBC and HGB features of the *ilab_e030704* table. We discretize each numerical feature using medical knowledge available in table *labn_e030704*. We use three bins: low, normal and high. The results registered in table *bio_e030704* were used to label each patient. The label of each patient is the result obtained in the first biopsy registered for each patient. After pre-processing, in the subtype problem we end up with 206 hepatitis-B patients and 297 hepatitis-C patients. In the illustrative examples presented throughout this thesis we identify the table that joins information from table *pt_e030704* and table *bio_e030704*, the one used to label patients, as *ptBioData*. The tables *ilab_e030704*, *hemat_e030704* and *labn_e030704* appear as *inHospRes*, *hematRes* and *labDesc*, respectively. We use these non-original names to improve readability.

B.2 Financial

The Financial dataset is a relational dataset that was provided by a bank, aimed at understanding certain mechanisms related with their product offer and to improve their services. This dataset register data about clients and their accounts by using eight tables. Among these tables we can find some tables that store static information about accounts, clients and district demographic data. The fact tables register information concerning credit card types, payments, transactions and loans for a given account. We use all examples, but we do not use all tables. We use tables *account*, *transaction*, *loan*, *card* and *disposition*. We built, again, a target table, using the link information in the *disposition* table, by joining information from table *account* and table *loan*.

Like in the Hepatitis dataset, the Financial dataset has many open problems. In this work we address the problem of predicting good or bad loans, i.e., discriminating between contracts finished without problems and contracts finished without the loan being payed. The class distribution is 203 positive examples and 31 negative examples.

B.3 FinancialB

The FinancialB dataset is a balanced version of the Financial dataset. The only difference is that we eliminate part of the examples available in the majority class to balance the dataset. The eliminated examples were randomly chosen. With this procedure we end up with a dataset where 60% of the examples are from the positive

Table B.1: Number of relations and class distribution of each database

Database	Number of Relations	Class Distribution
Hepatitis	7/**5**	**297/206**
Financial	8/**5**	**203/31**
FinancialB	8/**5**	**47/31**
Protein	**2**	**721/360/274/441/290**

class and 40% are from the negative class.

B.4 Protein

The third dataset is concerned with protein secondary structure classification. The dataset consists of logical sequences describing the secondary structure of protein domains. The task in this dataset is a multi-class classification problem and the goal is to predict one of the five most populated SCOP folds of alpha and beta proteins (a/b): TIM beta/alpha-barrel (721 examples), NAD(P)-binding Rossmann-fold domains (360 examples), Ribosomal protein L4 (274 examples), Cysteine hydrolase (441 examples), and Phosphotyrosine protein phosphatases I-like (290 examples). Inside the parentheses we present the class distribution in the dataset.

In Table B.1 we present the number of relations and the class distribution of each database. In bold we present the number of tables and class distributions used in this thesis.

Appendix C

RUSE-WARMR Results

Table C.1: Peak memory usage of the *RUSE-WARMR* framework (in kilobytes)

			Warmr		Filters		Class. induction	
Datasets	Support	ChiValue	Part A	Part B	Exclusive	Chi-Square	C45	SVM
Hepatitis	0.9	*No-Chi*	118.9 (15.8)	241.3 (7.6)	241.3 (5.8)	–	29.8 (1.5)	61.5 (1.5)
		10	118.9 (15.8)	241.3 (7.6)	241.3 (5.8)	29.8 (0.6)	27.9 (0.9)	60.7 (1.9)
		5	118.9 (15.8)	241.3 (7.6)	241.3 (5.8)	29.5 (0.9)	28.2 (0.7)	60.1 (3.6)
		Standard	243.5 (7.0)		–	–	29.7 (1.6)	62.3 (2.7)
	0.8	*No-Chi*	118.9 (15.4)	241.2 (7.0)	241.8 (6.4)	–	34.0 (0.2)	59.5 (1.0)
		10	118.9 (15.4)	241.2 (7.0)	241.8 (6.4)	31.9 (0.6)	28.1 (0.8)	60.7 (1.7)
		5	118.9 (15.4)	241.2 (7.0)	241.8 (6.4)	31.7 (0.8)	28.1 (1.0)	59.6 (3.9)
		Standard	242.5 (6.6)		–	–	33.9 (0.3)	59.9 (1.5)
Financial	0.9	*No-Chi*	125.6 (1.4)	125.2 (1.1)	125.2 (1.4)	–	40.6 (2.7)	63.5 (2.1)
		10	125.6 (1.4)	125.2 (1.1)	125.2 (1.4)	43.2 (3.2)	32.8 (2.3)	52.4 (3.0)
		5	125.6 (1.4)	125.2 (1.1)	125.2 (1.4)	41.3 (1.9)	30.2 (3.0)	47.6 (4.0)
		Standard	125.3 (1.6)		–	–	33.8 (0.4)	58.9 (0.5)
	0.8	*No-Chi*	125.3 (1.0)	125.9 (1.5)	125.9 (1.5)	–	70.9 (12.4)	96.3 (14.2)
		10	125.3 (1.0)	125.9 (1.5)	125.9 (1.5)	62.8 (9.8)	40.7 (6.5)	61.8 (6.7)
		5	125.3 (1.0)	125.9 (1.5)	125.9 (1.5)	57.1 (5.5)	34.2 (4.1)	55.1 (5.4)
		Standard	125.6 (1.8)				47.5 (2.0)	73.9 (3.3)
FinancialB	0.9	*No-Chi*	125.1 (2.0)	125.4 (1.3)	125.6 (1.4)	–	41.8 (2.0)	64.1 (1.1)
		10	125.1 (2.0)	125.4 (1.3)	125.6 (1.4)	43.7 (2.5)	33.1 (1.0)	52.6 (2.4)
		5	125.1 (2.0)	125.4 (1.3)	125.6 (1.4)	42.2 (2.4)	32.2 (2.1)	50.0 (3.1)
		Standard	125.9 (1.4)		–	–	33.9 (1.0)	58.3 (1.6)
	0.8	*No-Chi*	126.1 (1.3)	125.6 (1.1)	125.2 (1.1)	–	58.1 (5.0)	85.0 (5.7)
		10	126.1 (1.3)	125.6 (1.1)	125.2 (1.1)	55.1 (4.4)	37.9 (3.7)	59.2 (3.4)
		5	126.1 (1.3)	125.6 (1.1)	125.2 (1.1)	53.1 (3.5)	35.4 (2.3)	57.5 (2.7)
		Standard	126.2 (1.7)		–	–	46.4 (1.2)	74.0 (1.7)
Protein	0.9	*No-Chi*	24.1 (1.1)	18.7 (0.7)	28.7 (1.0)	–	31.4 (0.9)	55.3 (1.4)
		10	24.1 (1.1)	18.7 (0.7)	28.7 (1.0)	29.4 (0.8)	31.4 (1.0)	55.4 (1.2)
		5	24.1 (1.1)	18.7 (0.7)	28.7 (1.0)	29.5 (0.7)	31.5 (0.9)	55.4 (1.1)
		Standard	30.9 (1.0)		–	–	32.7 (0.7)	55.8 (1.2)
	0.8	*No-Chi*	23.2 (1.3)	18.2 (1.0)	27.9 (1.4)	–	30.3 (2.7)	56.5 (3.6)
		10	23.2 (1.3)	18.2 (1.0)	27.9 (1.4)	29.5 (2.0)	30.2 (2.6)	56.4 (3.6)
		5	23.2 (1.3)	18.2 (1.0)	27.9 (1.4)	29.7 (1.9)	30.4 (2.5)	56.6 (3.7)
		Standard	31.1 (0.8)		–	–	32.9 (0.9)	57.1 (1.7)

Table C.2: Run-time of the *RUSE-WARMR* framework (in seconds)

			Warmr		Filters		Class. induction	
Datasets	Support	ChiValue	Part A	Part B	Exclusive	Chi-Square	C4.5	SVM
Hepatitis	0.9	*No-Chi*	20.9 (0.9)	41.2 (1.5)	18.8 (1.5)	–	0.2 (0.4)	0.1 (0.3)
		10	20.9 (0.9)	41.2 (1.5)	18.8 (1.5)	0.2 (0.4)	0.1 (0.3)	0.1 (0.3)
		5	20.9 (0.9)	41.2 (1.5)	18.8 (1.5)	0.3 (0.5)	0.2 (0.4)	0.1 (0.3)
		Standard	50.2 (2.9)		–	–	0.1 (0.3)	0.6 (0.5)
	0.8	*No-Chi*	26.6 (0.7)	62.7 (2.6)	18.3 (0.7)	–	0.2 (0.4)	0.2 (0.4)
		10	26.6 (0.7)	62.7 (2.6)	18.3 (0.7)	0.3 (0.5)	0.3 (0.5)	0.3 (0.5)
		5	26.6 (0.7)	62.7 (2.6)	18.3 (0.7)	0.3 (0.5)	0.1 (0.3)	0.3 (0.5)
		Standard	66.5(1.8)		–	–	0.3 (0.5)	0.5 (0.5)
Financial	0.9	*No-Chi*	80.8 (8.4)	104.6 (22.4)	10.6 (0.7)	–	0.2 (0.4)	0.5 (0.5)
		10	80.8 (8.4)	104.6 (22.4)	10.6 (0.7)	0.2 (0.4)	0.1 (0.3)	0.4 (0.5)
		5	80.8 (8.4)	104.6 (22.4)	10.6 (0.7)	0.5 (0.5)	0.0 (0.0)	0.2 (0.4)
		Standard	69.8 (9.0)		–	–	0.2 (0.4)	0.4 (0.5)
	0.8	*No-Chi*	223.8 (12.1)	357.2 (92.7)	28.1 (5.4)	–	0.4 (0.5)	0.4 (0.5)
		10	223.8 (12.1)	357.2 (92.7)	28.1 (5.4)	0.4 (0.5)	0.1 (0.3)	0.5 (0.5)
		5	223.8 (12.1)	357.2 (92.7)	28.1 (5.4)	0.1 (0.3)	0.3 (0.5)	0.3 (0.5)
		Standard	243.2 (17.3)		–	–	0.4 (0.5)	0.6 (0.5)
FinancialB	0.9	*No-Chi*	168.2 (16.2)	104.8 (22.4)	12.3 (0.8)	–	0.1 (0.3)	0.5 (0.5)
		10	168.2 (16.2)	104.8 (22.4)	12.3 (0.8)	0.3 (0.5)	0.0 (0.0)	0.4 (0.5)
		5	168.2 (16.2)	104.8 (22.4)	12.3 (0.8)	0.2 (0.4)	0.4 (0.5)	0.2 (0.4)
		Standard	106.6 (23.7)		–	–	0.1 (0.3)	0.4 (0.5)
	0.8	*No-Chi*	364.9 (32.5)	352.8 (87.7)	37.3 (8.5)	–	0.2 (0.4)	0.5 (0.5)
		10	364.9 (32.5)	352.8 (87.7)	37.3 (8.5)	0.2 (0.4)	0.3 (0.5)	0.4 (0.5)
		5	364.9 (32.5)	352.8 (87.7)	37.3 (8.5)	0.2 (0.4)	0.3 (0.5)	0.3 (0.5)
		Standard	323.2 (19.2)		–	–	0.1 (0.3)	0.5 (0.5)
Protein	0.9	*No-Chi*	2.0 (0.3)	1.1 (0.4)	1.3 (0.4)	–	0.2 (0.4)	0.2 (0.4)
		10	2.0 (0.3)	1.1 (0.4)	1.3 (0.4)	0.1 (0.3)	0.2 (0.4)	0.3 (0.5)
		5	2.0 (0.3)	1.1 (0.4)	1.3 (0.4)	0.2 (0.3)	0.2 (0.4)	0.3 (0.5)
		Standard	1.3 (0.3)		–	–	0.1 (0.3)	0.3 (0.5)
	0.8	*No-Chi*	2.1 (0.4)	1.1 (0.4)	1.3 (0.3)	–	0.1 (0.3)	0.4 (0.5)
		10	2.1 (0.4)	1.1 (0.4)	1.3 (0.3)	0.2 (0.4)	0.2 (0.4)	0.3 (0.4)
		5	2.1 (0.4)	1.1 (0.4)	1.3 (0.3)	0.2 (0.3)	0.1 (0.3)	0.4 (0.5)
		Standard	1.3 (0.2)		–	–	0.2 (0.4)	0.2 (0.4)

Table C.3: Generalization accuracy of the *RUSE-WARMR* framework selecting the top ten and the top twenty most interesting patterns

			Generalization Accuracy	
Datasets	Support	NumTop	C4.5	SVM
Hepatitis	0.9	10	64.63 (8.20)	62.50 (6.18)
		20	64.63 (8.20)	62.50 (6.18)
	0.8	10	63.42 (6.92)	63.50 (6.09)
		20	63.42 (6.92)	63.50 (6.09)
Financial	0.9	10	86.90 (1.45)	86.90 (1.45)
		20	86.90 (1.45)	86.90 (1.45)
	0.8	10	86.90 (1.45)	86.90 (1.45)
		20	86.90 (1.45)	86.90 (1.45)
FinancialB	0.9	10	61.40 (1.90)	61.40 (1.90)
		20	61.40 (1.90)	60.20 (4.05)
	0.8	10	54.20 (14.26)	60.10 (20.47)
		20	54.20 (14.26)	62.60 (16.93)
Protein	0.9	10	63.33 (0.01)	63.33 (0.01)
		20	63.33 (0.01)	63.33 (0.01)
	0.8	10	63.18 (0.01)	63.23 (0.01)
		20	63.18 (0.01)	63.23 (0.01)

Appendix D

MuSer Results

Table D.1: Peak memory usage in each step of the *MuSer* framework (in kilobytes)

			Sequence Miner			Filter Chi-Square	Classifier	
Datasets	Support	Top-K	Part A	Part B	Mapping		C4.5	SVM
Hepatitis	0.9	10	613.2 (39.1)	1203.2 (37.1)	196930.4 (54680.0)	42759.6 (2435.1)	35544.4 (611.5)	55045.2 (213.0)
		20	612.4 (39.4)	1204.8 (37.6)	196312.0 (53586.2)	44692.4 (2312.5)	36026.4 (1174.2)	56818.4 (268.4)
	0.8	10	875.6 (22.4)	2006.0 (93.9)	3773773.2 (3341.2)	1263896.8 (495713.9)	36351.6 (713.5)	55445.2 (298.3)
		20	874.4 (23.8)	2006.0 (93.2)	3774758.4 (3256.2)	1270796.4 (502124.9)	35174.0 (788.1)	57100.8 (654.0)
Protein	0.9	10	916.4 (4.9)	752.7 (3.3)	636752.9 (12406.4)	442126.1 (48448.6)	36515.4 (681.5)	58082.6 (848.3)
		20	916.2 (4.7)	752.6 (3.4)	636936.5 (12417.7)	443263.8 (47970.7)	37865.6 (570.9)	60600.5 (908.3)
	0.8	10	1010.9 (7.0)	830.6 (6.9)	787832.9 (1047.4)	3347801.2 (288293.0)	36362.2 (561.2)	58195.1 (871.9)
		20	1011.1 (7.5)	830.6 (7.2)	787873.7 (879.3)	3202965.5 (367953.5)	37245.2 (1202.4)	59632.5 (1715.7)

Table D.2: Run-time of each step of the *MuSer* framework (in seconds)

			Sequence Miner			Filter Chi-Square	Classifier	
Datasets	Support	Top-K	Part A	Part B	Mapping		C4.5	SVM
Hepatitis	0.9	10	0.0 (0.0)	0.0 (0.0)	3.2 (0.8)	0.3 (0.5)	0.2 (0.4)	0.3 (0.5)
		20	0.0 (0.0)	0.0 (0.0)	3.0 (0.5)	0.3 (0.5)	0.2 (0.4)	0.2 (0.4)
	0.8	10	0.0 (0.0)	18.6 (6.4)	911.9 (316.2)	10.4 (4.0)	0.1 (0.3)	0.3 (0.5)
		20	0.0 (0.0)	18.4 (6.4)	900.7 (306.8)	10.5 (4.2)	0.2 (0.4)	0.4 (0.5)
Protein	0.9	10	1.8 (0.3)	0.0 (0.1)	142.2 (9.4)	4.8 (1.1)	0.2 (0.4)	0.3 (0.5)
		20	1.8 (0.2)	0.0 (0.1)	141.2 (9.0)	4.1 (0.7)	0.2 (0.3)	0.4 (0.5)
	0.8	10	11.2 (0.8)	0.2 (0.3)	1186.3 (62.0)	39.4 (4.2)	0.2 (0.4)	0.3 (0.4)
		20	11.2 (0.9)	0.2 (0.2)	1190.8 (62.8)	41.5 (5.0)	0.2 (0.4)	0.5 (0.5)

Appendix E

XMuSer Results

Table E.1: Peak memory usage in each step of the *XMuSer* framework (in kilobytes)

				ILP learning		
Datasets	Support	cSpade	Mapping	Frequent	Closed	Maximal
Hepatitis	0.9	2307.6 (50.2)	185636.4 (6215.2)	29477.2 (989.2)	29476.8 (990.4)	29064 (649.3)
	0.8	3623.6 (223.2)	218436.8 (1360.1)	28867.2 (7.3)	29069.6 (647.4)	29070 (647.2)
	Aleph	–	–	28860 (3.3)		
Financial	0.9	17202 (181.7)	1133536.4 (19778.9)	101424.8 (11360.2)	101426.4 (11358.1)	86680.8 (10464.5)
	0.8	17982.4 (319.6)	2067001.2 (49106.1)	101426 (11359.5)	101426.8 (11358.9)	95897.6 (13761.3)
	Aleph	–	–	100364.8 (11140.7)		
FinancialB	0.9	8302.8 (123.5)	1152971.2 (40671.4)	63067.6 (3958.4)	63066.8 (3957.2)	62862.8 (3619)
	0.8	10312.8 (171.8)	2258075.2 (65185)	63066.8 (3957.2)	63066.4 (3956.4)	58766 (5114.5)
	Aleph	–	–	62645.6 (3371.3)		
Protein	0.9	2704.3 (207.5)	42566 (18818.5)	21989.8 (228.7)	22527 (118.5)	22589 (111.7)
	0.8	3923.9 (1526.3)	150543.5 (142006.4)	21139.2 (2311.4)	21262.8 (2403.8)	21176.9 (2270.5)
	Aleph	–	–	17832.5 (67.9)		

Table E.2: Run-time of each step of the *XMuSer* framework (in seconds)

				ILP learning		
Datasets	Support	cSpade	Mapping	Frequent	Closed	Maximal
Hepatitis	0.9	0.0 (0.0)	5.6 (0.3)	3.9 (1.8)	3.8 (1.9)	3.1 (1.7)
	0.8	0.6 (0.5)	6.3 (0.3)	5.0 (0.7)	4.9 (0.6)	4.8 (0.6)
	Aleph	–	–	1.1 (0.3)		
Financial	0.9	61.7 (2.7)	960.7 (446.5)	2.8 (0.4)	2.9 (0.3)	2.6 (0.5)
	0.8	102.0 (4.1)	4438.8 (2093.8)	2.8 (0.4)	2.9 (0.3)	2.7 (0.5)
	Aleph	–	–	1.8 (0.4)		
FinancialB	0.9	31.1 (2.3)	1804.8 (833.9)	1.8 (0.4)	1.8 (0.4)	1.5 (0.5)
	0.8	49.7 (1.9)	7972.2 (3694.2)	1.8 (0.4)	2.0 (0.0)	1.4 (0.5)
	Aleph	–	–	1.3 (0.5)		
Protein	0.9	0.7 (0.3)	1.6 (1.9)	24.4 (2.3)	20.8 (2.4)	19.9 (2.6)
	0.8	6.1 (6.9)	110.9 (143.5)	14.9 (10.4)	12.4 (9.0)	10.5 (7.9)
	Aleph	–	–	7.0 (0.3)		

Table E.3: Generalization accuracy of the *XMuSer* framework using a propositional learner in the last step

			C4.5 learner			SVM learner		
Datasets	Support	Top-k	Frequent	Closed	Maximal	Frequent	Closed	Maximal
Hepatitis	0.9	10	62.10 (6.01)	62.10 (6.01)	62.10 (6.01)	64.10 (6.33)	64.10 (6.33)	64.70 (6.18)
		20	62.10 (6.01)	62.10 (6.01)	62.10 (6.01)	63.70 (6.07)	63.70 (6.07)	65.10 (6.61)
	0.8	10	66.20 (8.11)	67.20 (8.55)	67.50 (7.17)	66.40 (7.85)	66.80 (7.90)	67.20 (8.09)
		20	66.70 (7.89)	67.10 (7.89)	67.40 (6.74)	67.50 (7.09)	68.30 (6.34)	68.00 (6.90)
Financial	0.9	10	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)
		20	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)
	0.8	10	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)
		20	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)	86.90 (1.45)
FinancialB	0.9	10	61.40 (1.90)	61.40 (1.90)	65.50 (18.17)	61.40 (1.90)	61.40 (1.90)	64.10 (18.30)
		20	61.40 (1.90)	61.40 (1.90)	65.50 (18.17)	61.40 (1.90)	61.40 (1.90)	67.70 (20.82)
	0.8	10	61.40 (1.90)	61.40 (1.90)	66.90 (15.23)	61.40 (1.90)	61.40 (1.90)	64.30 (17.22)
		20	61.40 (1.90)	61.40 (1.90)	64.30 (14.67)	61.40 (1.90)	61.40 (1.90)	72.60 (13.37)
Protein	0.9	10	61.52 (0.03)	68.83 (0.04)	70.13 (0.04)	59.89 (0.03)	67.49 (0.04)	68.26 (0.03)
		20	69.94 (0.04)	71.96 (0.03)	74.26 (0.03)	68.89 (0.04)	70.19 (0.04)	71.62 (0.05)
	0.8	10	63.72 (0.03)	71.26 (0.04)	73.52 (0.05)	63.53 (0.03)	69.39 (0.04)	69.33 (0.03)
		20	75.10 (0.04)	77.30 (0.04)	78.78 (0.03)	73.12 (0.04)	73.46 (0.04)	73.93 (0.03)

Table E.4: Theory size: number of nodes

			Tree Size		
Datasets	Support	Top-k	Frequent	Closed	Maximal
Hepatitis	0.9	10	6.4 (2.5)	6.4 (2.5)	6.4 (2.5)
		20	6.4 (2.5)	6.4 (2.5)	6.4 (2.5)
	0.8	10	30.8 (11.7)	30.6 (12.0)	30.0 (9.9)
		20	30.6 (7.5)	33.8 (8.4)	31.2 (5.9)
Financial	0.9	10	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)
		20	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)
	0.8	10	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)
		20	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)
FinancialB	0.9	10	1.0 (0.0)	1.0 (0.0)	3.0 (0.0)
		20	1.0 (0.0)	1.0 (0.0)	5.0 (2.7)
	0.8	10	1.0 (0.0)	1.0 (0.0)	5.0 (2.1)
		20	1.0 (0.0)	1.0 (0.0)	6.8 (2.6)
Protein	0.9	10	6.6 (2.4)	10.9 (3.6)	14.9 (4.4)
		20	16.9 (5.4)	19.2 (5.4)	27.6 (6.3)
	0.8	10	6.2 (2.7)	10.5 (4.4)	18.1 (4.5)
		20	15.2 (4.2)	18.3 (5.0)	25.5 (5.2)

Appendix F

ConFOLding Results

Table F.1: Peak memory usage of the *ConFOLding* framework (in kilobytes)

				Aleph - ILP Learning		
Datasets	BeamSize	BeamSouL	Mapping	Frequent	Closed	Maximal
Hepatitis	0	11280503.2 (1174.0)	14779.2 (818.9)	102455.2 (2905.7)	102658.4 (102658.4)	102448.4 (2981.6)
	5	11292461.6 (3388.3)	16344.0 (1585.6)	104511.2 (2070.1)	104515.2 (104515.2)	104706.0 (2380.0)
	20	11409307.6 (216260.8)	28552.8 (7517.2)	104068.8 (2181.1)	103665.2 (103665.2)	104304.0 (2275.2)
	35	11385082.0 (28170.2)	43365.6 (8393.9)	103810.8 (2083.9)	104219.6 (104219.6)	103537.6 (1579.5)
	50	11437983.6 (39307.6)	59826.8 (12376.9)	104430.8 (2162.5)	103783.2 (103783.2)	103935.2 (2036.4)
	MineSeqLog	11273648.0 (1594.2)	15391.2 (1459.9)	–	–	–
Financial	0	57950.4 (658.9)	19707.2 (23.8)	44403.6 (3.0)	44403.6 (44403.6)	44404.4 (1.3)
	5	58771.2 (1084.8)	19697.6 (37.6)	44405.2 (1.9)	44403.2 (44403.2)	44404.8 (3.7)
	20	61841.6 (26.9)	19642.4 (26.8)	44404.8 (1.7)	44404.8 (44404.8)	44404.0 (2.7)
	35	66169.6 (1823.7)	23495.2 (1350.2)	44404.8 (3.2)	44403.6 (44403.6)	44403.6 (3.5)
	50	70373.6 (1174.8)	24032.0 (21.2)	44402.8 (2.7)	44404.0 (44404.0)	44403.6 (3.0)
	MineSeqLog	86746.4 (36.9)	24184.0 (21.0)	–	–	–
FinancialB	0	36350.4 (64.0)	14702.0 (2.8)	44399.2 (2.5)	44400.4 (44400.4)	44399.6 (3.0)
	5	36351.6 (65.8)	14704.0 (3.8)	44399.6 (2.3)	44401.6 (44401.6)	44399.6 (3.0)
	20	37559.2 (1046.3)	14705.6 (2.1)	44400.0 (2.7)	44400.0 (44400.0)	44400.4 (1.3)
	35	39606.8 (1092.0)	14706.8 (1.9)	44399.6 (2.3)	44400.4 (44400.4)	44400.4 (2.3)
	50	42499.2 (2064.1)	14705.6 (2.1)	44401.2 (1.9)	44398.8 (44398.8)	44399.2 (2.5)
	MineSeqLog	47998.0 (1118.6)	14698.8 (3.3)	–	–	–
Protein	0	24945.2 (419.9)	13288.6 (2.3)	20104.6 (561.7)	20001.0 (20001.0)	20022.2 (543.6)
	5	27617.6 (264.1)	13288.7 (2.0)	22753.7 (143.1)	22727.1 (22727.1)	22676.5 (70.0)
	20	36626.4 (891.6)	13910.8 (4.5)	24423.3 (621.0)	24505.7 (24505.7)	24297.2 (427.8)
	35	46226.0 (1310.7)	14023.5 (110.0)	26007.8 (405.5)	26089.5 (26089.5)	25795.4 (387.7)
	50	56250.1 (581.2)	14312.9 (188.8)	26203.0 (732.1)	26174.6 (26174.6)	25983.1 (650.0)
	MineSeqLog	26662.7 (215.6)	14332.4 (2.3)	–	–	–

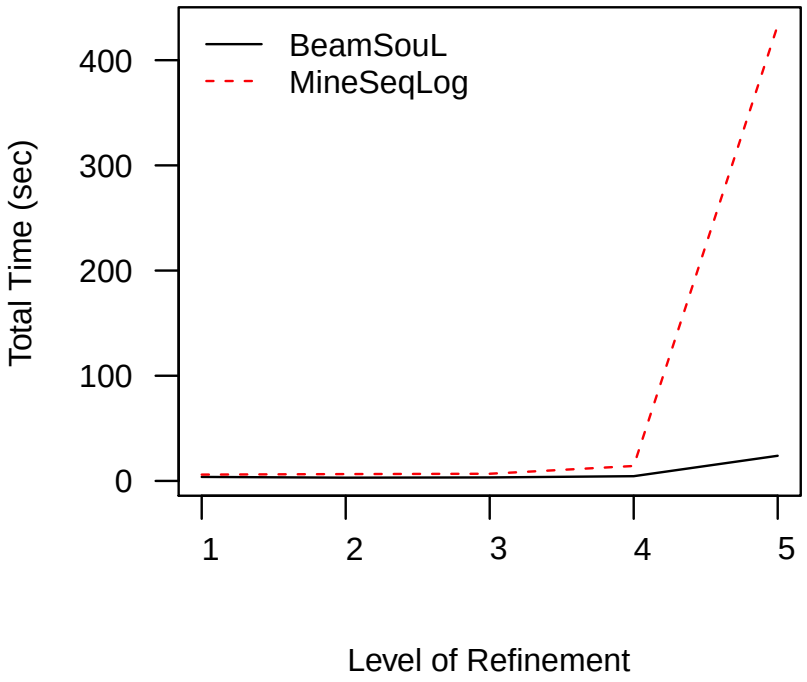


Figure F.1: Run-time needed by the *BeamSouL* and the MineSeqLog algorithms to find patterns in minority partition of the Hepatitis dataset. We present the run-time of these algorithms for different refinement levels

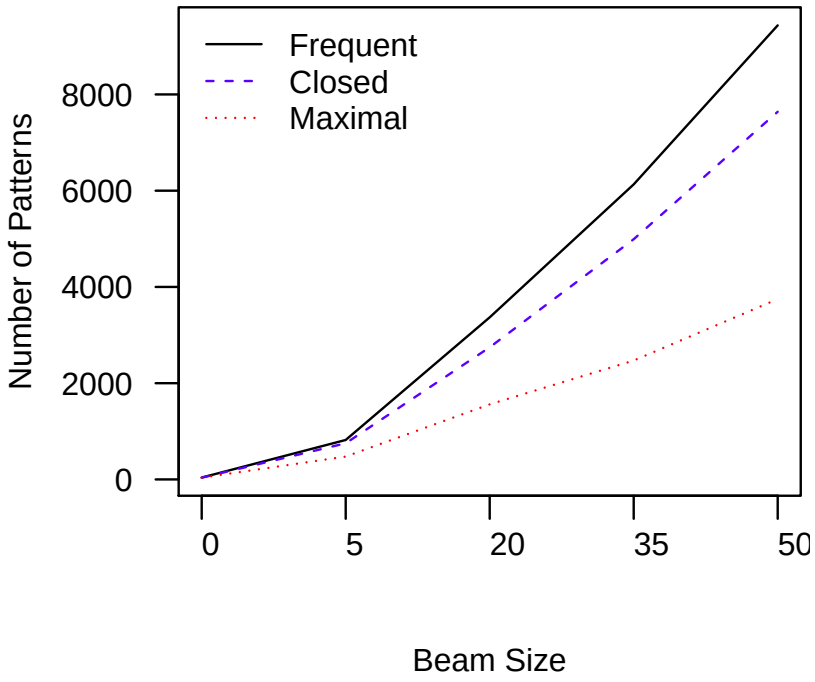


Figure F.2: Number of patterns found by the *BeamSouL* algorithm in the minority partition of the Hepatitis dataset. We present the number of frequent, closed and maximal patterns found for a range of beam sizes

Table F.2: Run-time of the *ConFOLding* framework (in seconds)

Dataset	BeamSize	BeamSouL Seq. Miner		Aleph - ILP Learning		
		Positive-Run	Negative-Run	Frequent	Closed	Maximal
Hepatitis	0	9.2 (6.6)	6.8 (5.5)	14.4 (1.7)	14.4 (1.8)	14.2 (2.2)
	5	31.1 (8.8)	6.5 (1.4)	15.5 (2.0)	14.3 (1.8)	14.2 (2.6)
	20	331.6 (277.3)	13.2 (10.6)	15.4 (2.0)	14.4 (1.9)	14.2 (2.0)
	35	925.0 (481.6)	24.9 (28.8)	15.5 (1.6)	14.5 (1.9)	14.3 (1.5)
	50	2050.5 (1018.1)	44.0 (45.1)	15.2 (1.9)	14.4 (1.9)	14.3 (1.8)
	MineSeqLog	15.5 (1.2)	14.2 (1.4)	–	–	–
Financial	0	397.4 (11.0)	120.0 (16.5)	1.0 (0.0)	1.0 (2.3)	1.0 (0)
	5	410.9 (9.8)	120.4 (15.7)	1.1 (0.3)	1.1 (3.2)	1.0 (0.0)
	20	453.4 (11.9)	123.2 (16.2)	1.0 (0.0)	1.1 (1.7)	1.0 (0.0)
	35	457.8 (16.5)	126.9 (16.1)	1.0 (0.0)	1.0 (3.0)	1.1(0.3)
	50	476.7 (17.2)	131.4 (15.7)	1.1 (0.3)	1.0 (2.7)	1.0 (0.0)
	MineSeqLog	686.7 (22.2)	617.2 (16.6)	–	–	–
FinancialB	0	144.3 (5.0)	48.2 (3.7)	1.0 (0.0)	1.0 (2.3)	1.0 (0.0)
	5	145.8 (4.4)	49.6 (3.8)	1.0 (0.0)	1.1 (2.1)	1.0 (0.0)
	20	171.2 (6.5)	51.8 (3.7)	1.2 (0.4)	0.9 (2.7)	1.0 (0.0)
	35	184.5 (7.8)	54.5 (3.8)	1.1 (0.3)	1.1 (2.3)	1.0 (0.0)
	50	759.5 (491.6)	56.9 (4.4)	0.9 (0.3)	1.0 (2.7)	1.1 (0.6)
	MineSeqLog	303.2 (34.1)	264.7 (8.2)	–	–	–
Protein	0	1.9 (0.5)	1.8 (0.5)	5.1 (0.6)	5.0 (0.5)	5.1 (0.5)
	5	2.4 (0.4)	1.9 (0.3)	5.9 (0.6)	5.8 (0.2)	5.0 (0.6)
	20	4.2 (0.5)	3.0 (0.4)	9.5 (2.9)	7.7 (0.6)	7.0 (2.0)
	35	8.0 (1.1)	4.1 (0.4)	15.8 (2.1)	13.1 (0.4)	11.5 (1.6)
	50	11.2 (0.8)	6.1 (1.0)	15.5 (3.0)	12.7 (0.6)	11.1 (1.4)
	MineSeqLog	7.8 (0.5)	7.3 (0.5)	–	–	–

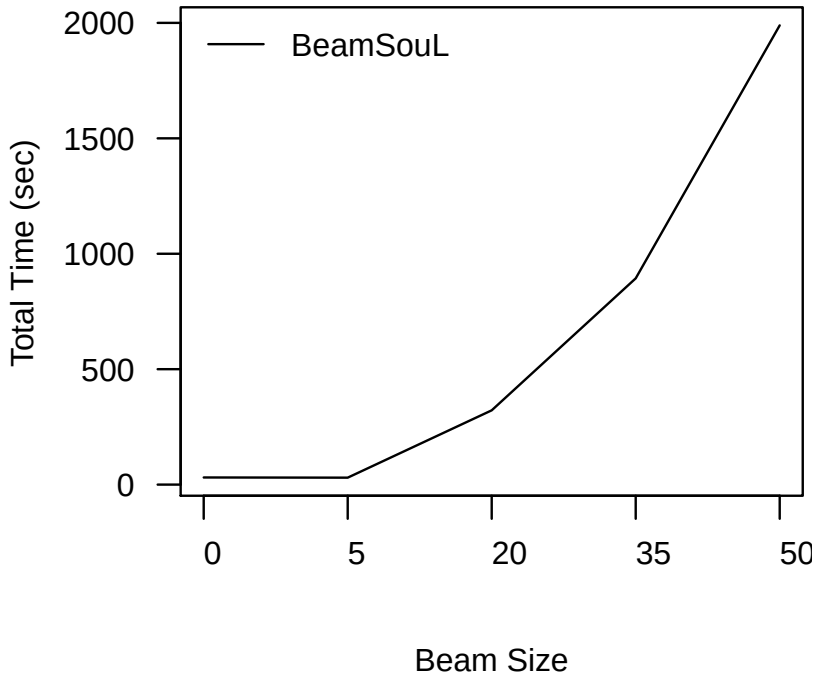


Figure F.3: Run-time needed by the *BeamSouL* algorithm to find patterns in the Hepatitis dataset. We present the run-time of the *BeamSouL* algorithm for different beam sizes and for the negative-run of the *ConFOLding* framework

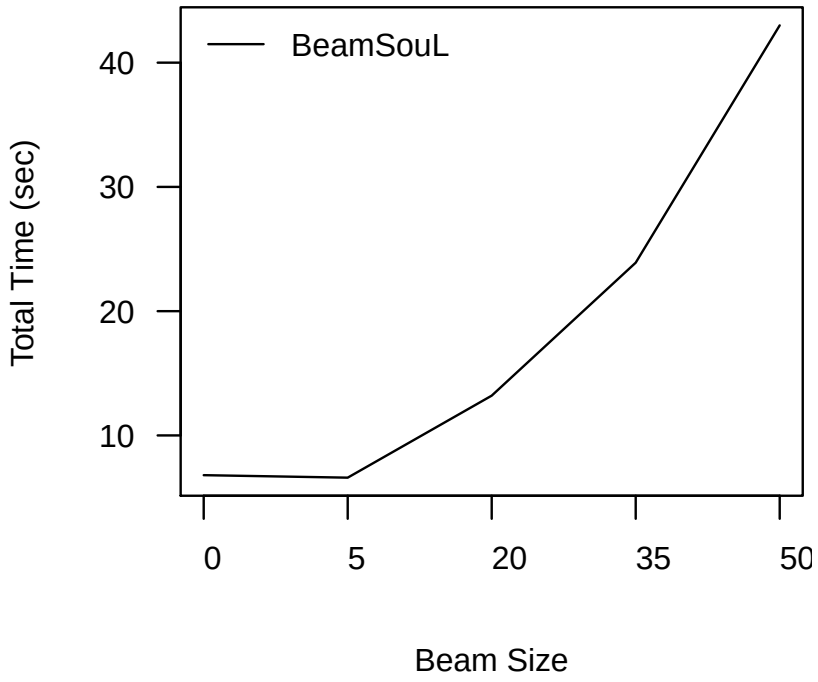


Figure F.4: Run-time needed by the *BeamSouL* algorithm to find patterns in the Hepatitis dataset. We present the run-time of the *BeamSouL* algorithm for different beam sizes and for the positive-run of the *ConFOLding* framework

Table F.3: Theory size: number of rules

Dataset	BeamSize	Number of Rules		
		Frequent	Closed	Maximal
Hepatitis	0	71.8 (2.7)	71.8 (2.7)	71.8 (2.7)
	5	85.6 (7.4)	85.5 (7.2)	85.3 (7.1)
	20	85.5 (7.2)	85.5 (7.2)	85.7 (7.2)
	35	85.4 (7.0)	85.4 (7.0)	85 (7.2)
	50	85.5 (7.1)	85.5 (7.1)	85.3 (7.1)
Financial	0	3.4 (0.7)	3.4 (0.7)	3.4 (0.7)
	5	4.9 (0.7)	4.3 (1.0)	4.3 (0.9)
	20	4.9 (0.7)	4.3 (1.0)	4.3 (0.9)
	35	5.0 (1.0)	4.6 (0.5)	4.3 (0.9)
	50	4.7 (1.4)	3.9 (0.9)	3.6 (0.5)
FinancialB	0	2.8 (0.4)	2.8 (0.4)	2.8 (0.4)
	5	4.6 (3.2)	3.8 (3.3)	3.8 (3.3)
	20	8.6 (3.5)	6.3 (3.9)	6.2 (3.8)
	35	8.4 (3.6)	6.6 (3.7)	6.4 (3.9)
	50	8.3 (3.5)	7.1 (3.4)	7.1 (3.4)
Protein	0	173.2 (9.7)	173.1 (9.7)	173.1 (9.7)
	5	165.5 (7.0)	164.9 (7.2)	163.0 (6.6)
	20	165.8 (8.3)	155.1 (11.2)	152.9 (10.6)
	35	151.1 (16.2)	129.8 (14.7)	120.6 (11.8)
	50	146.6 (21.4)	124.9 (15.4)	115.0 (13.1)

Table F.4: Generalization accuracy of the propositional *ConFOLding* framework

Datasets	Top-K	BeamSize	C4.5			SVM		
			Frequent	Closed	Maximal	Frequent	Closed	Maximal
Hepatitis	10	0	74.80 (5.47)	74.80 (5.47)	72.40 (5.27)	78.40 (4.72)	78.40 (4.72)	77.40 (5.48)
		5	73.50 (7.44)	72.60 (6.54)	71.30 (5.66)	70.20 (5.77)	70.90 (6.81)	71.30 (6.93)
		20	72.10 (7.31)	71.90 (7.03)	71.50 (6.49)	70.20 (6.18)	70.90 (6.89)	71.00 (6.88)
		35	72.50 (7.11)	72.30 (6.83)	71.20 (6.73)	70.40 (6.17)	70.90 (6.72)	71.00 (6.38)
		50	70.80 (5.94)	70.30 (5.25)	70.00 (5.31)	70.80 (6.84)	71.60 (6.17)	71.00 (6.80)
	20	0	76.80 (4.83)	76.60 (5.13)	74.60 (3.78)	76.60 (4.22)	77.10 (4.25)	76.40 (4.27)
		5	71.90 (6.01)	72.70 (5.74)	73.50 (8.10)	71.30 (6.67)	71.20 (5.31)	73.80 (7.79)
		20	71.70 (6.80)	71.10 (6.37)	71.50 (6.88)	69.80 (6.46)	71.30 (6.78)	71.60 (7.17)
		35	72.50 (7.11)	72.30 (6.53)	71.50 (5.84)	70.00 (5.85)	71.80 (6.61)	72.00 (7.04)
		50	71.30 (6.43)	71.70 (5.52)	71.10 (6.21)	70.70 (6.98)	72.50 (6.75)	72.40 (6.57)
Financial	10	0	96.40 (2.95)	96.40 (2.95)	96.40 (2.95)	96.40 (2.95)	96.40 (2.95)	96.40 (2.95)
		5	95.50 (4.20)	96.40 (2.95)	96.40 (2.95)	95.50 (4.20)	96.40 (2.95)	96.40 (2.95)
		20	95.50 (4.20)	96.40 (2.95)	96.40 (2.95)	95.50 (4.20)	96.40 (2.95)	96.40 (2.95)
		35	94.60 (4.88)	95.10 (4.72)	96.40 (2.95)	95.50 (4.20)	95.10 (4.72)	96.40 (2.95)
		50	95.00 (4.42)	95.50 (4.20)	96.40 (2.95)	95.50 (4.20)	95.50 (4.20)	96.40 (2.95)
	20	0	97.20 (2.70)	97.20 (2.70)	97.20 (2.70)	95.90 (4.01)	97.20 (2.70)	97.20 (2.70)
		5	96.40 (2.95)	97.20 (2.70)	97.20 (2.70)	96.40 (2.95)	97.20 (2.70)	97.20 (2.70)
		20	95.50 (4.20)	96.40 (2.95)	96.40 (2.95)	95.50 (4.20)	96.40 (2.95)	96.40 (2.95)
		35	95.10 (4.72)	96.40 (2.95)	96.40 (2.95)	95.10 (4.72)	96.40 (2.95)	96.40 (2.95)
		50	95.00 (4.42)	96.40 (2.95)	96.40 (2.95)	95.50 (4.20)	96.40 (2.95)	96.40 (2.95)
FinancialB	10	0	94.00 (6.32)	94.00 (6.32)	94.00 (6.32)	94.00 (6.32)	94.00 (6.32)	94.00 (6.32)
		5	94.00 (6.32)	90.10 (13.00)	94.00 (6.32)	94.00 (6.32)	90.10 (13.00)	94.00 (6.32)
		20	94.00 (6.32)	90.10 (13.00)	94.00 (6.32)	94.00 (6.32)	90.10 (13.00)	94.00 (6.32)
		35	94.00 (6.32)	90.10 (13.00)	90.10 (13.00)	94.00 (6.32)	90.10 (13.00)	90.10 (13.00)
		50	90.10 (13.00)	87.60 (13.28)	90.10 (13.00)	90.10 (13.00)	87.60 (13.28)	90.10 (13.00)
	20	0	94.00 (6.32)	94.00 (6.32)	94.00 (6.32)	94.00 (6.32)	94.00 (6.32)	94.00 (6.32)
		5	94.00 (6.32)	94.00 (6.32)	94.00 (6.32)	94.00 (6.32)	94.00 (6.32)	94.00 (6.32)
		20	94.00 (6.32)	90.10 (13.00)	94.00 (6.32)	94.00 (6.32)	90.10 (13.00)	94.00 (6.32)
		35	94.00 (6.32)	90.10 (13.00)	94.00 (6.32)	94.00 (6.32)	90.10 (13.00)	94.00 (6.32)
		50	94.00 (6.32)	90.10 (13.00)	91.40 (11.93)	94.00 (6.32)	90.10 (13.00)	91.40 (11.93)
Protein	10	0	54.51 (0.02)	54.61 (0.02)	54.61 (0.02)	53.94 (0.02)	53.70 (0.02)	53.70 (0.02)
		5	54.52 (0.02)	53.99 (0.02)	53.32 (0.02)	54.41 (0.02)	53.89 (0.02)	53.56 (0.02)
		20	56.83 (0.03)	58.27 (0.04)	58.70 (0.04)	57.07 (0.03)	57.69 (0.04)	57.64 (0.04)
		35	60.30 (0.03)	65.28 (0.04)	66.53 (0.03)	58.33 (0.03)	61.25 (0.03)	61.16 (0.03)
		50	60.82 (0.03)	66.33 (0.03)	67.02 (0.03)	59.77 (0.04)	62.55 (0.02)	60.82 (0.04)
	20	0	54.75 (0.02)	54.75 (0.02)	54.75 (0.02)	53.99 (0.02)	53.99 (0.02)	53.99 (0.02)
		5	54.85 (0.02)	53.27 (0.02)	53.46 (0.02)	54.80 (0.02)	53.70 (0.02)	53.89 (0.02)
		20	57.36 (0.04)	59.76 (0.05)	60.33 (0.05)	57.07 (0.04)	58.32 (0.04)	58.70 (0.04)
		35	62.70 (0.03)	67.01 (0.03)	67.64 (0.03)	59.72 (0.03)	63.51 (0.02)	61.54 (0.03)
		50	63.46 (0.02)	67.87 (0.02)	68.26 (0.02)	61.21 (0.02)	63.75 (0.02)	62.35 (0.03)

Table F.5: Theory size: number of tree nodes

			Tree Size		
Datasets	Top-K	BeamSize	Frequent	Closed	Maximal
Hepatitis	10	0	40.8 (10.4)	40.8 (10.4)	44.0 (10.6)
		5	13.0 (0.9)	14.6 (3.4)	16.2 (4.6)
		20	13.4 (3.7)	15.0 (5.5)	14.2 (4.3)
		35	13.0 (2.3)	14.2 (4.4)	14.2 (4.4)
		50	13.2 (2.6)	13.6 (2.8)	14.2 (3.7)
	20	0	41.4 (11.2)	42.4 (10.4)	43.0 (10.6)
		5	15.4 (5.2)	19.4 (7.9)	22.0 (7.6)
		20	12.4 (1.9)	15.4 (6.2)	15.2 (5.3)
		35	12.8 (2.4)	14.0 (4.4)	16.0 (6.1)
		50	13.0 (2.7)	14.8 (3.7)	18.6 (7.0)
Financial	10	0	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
		5	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
		20	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
		35	3.0 (0.0)	3.2 (0.6)	3.4 (0.8)
		50	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
	20	0	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
		5	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
		20	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
		35	3.0 (0.0)	3.2 (0.6)	3.0 (0.0)
		50	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
FinancialB	10	0	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
		5	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
		20	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
		35	3.4 (0.8)	3.4 (0.8)	3.0 (0.0)
		50	3.4 (0.8)	3.4 (0.8)	3.4 (0.8)
	20	0	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
		5	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
		20	3.0 (0.0)	3.0 (0.0)	3.0 (0.0)
		35	3.4 (0.8)	3.0 (0.0)	3.0 (0.0)
		50	3.4 (0.8)	3.2 (0.6)	3.0 (0.0)
Protein	10	0	7.3 (1.4)	7.3 (1.3)	7.3 (1.3)
		5	5.4 (1.1)	5.9 (1.4)	6.7 (1.2)
		20	4.4 (1.1)	6.2 (2.1)	7.5 (2.1)
		35	4.9 (1.2)	7.3 (2.1)	9.9 (2.9)
		50	4.9 (1.6)	7.6 (2.3)	10.1 (2.7)
	20	0	7.4 (1.3)	7.6 (1.2)	7.6 (1.2)
		5	6.5 (1.7)	6.5 (1.2)	6.7 (1.1)
		20	5.8 (2.0)	9.1 (2.7)	9.9 (3.1)
		35	6.5 (1.9)	10.6 (3.8)	13.4 (4.6)
		50	6.0 (2.2)	11.5 (3.5)	15.0 (3.8)