The Graphical User Interface for RAP ILP System

Nelson Prazeres Costa

Thesis submitted for the degree of
Integrated Master in Electrical and Computers Engineering
Telecommunications major

Orientator: Luboš Popelínský (Ph.D)

July 2008
Abstract

Data Mining is a new, fast-growing research area with great importance nowadays. Discovering knowledge in databases requires computational support and Machine Learning applications is a good approach to find previous unknown patterns in data. The complexity of the algorithms allied with compound parameters and high-level results makes Data Mining a complex task requiring normally a Machine Learning expertise to work with these applications.

With the implementation of a Graphical User Interface to RAP Inductive Logic Programming system, a Machine Learning application for mining maximal frequent patterns in first-order logic, life may be facilitate to the normal user. The goal is to build an interface that can get the parameters and present the results of the RAP ILP algorithm in a user-friendly way. Developing such an interface that can improve the application potentiality and visibility is essential to improve the applicability and success of this system.

This thesis reports all the theoretical work performed as a background of the project implementation. A brief description of Data Mining, Machine Learning and Inductive Logic Programming is presented and RAP ILP system is introduced. The Graphical User Interface is presented with the application requirements, architecture, design, implementation and test.
Acknowledgements

It was with a great pleasure and delight that I performed this project. The challenge was well received and a great dedication was taken in the project. I thank all involved, in a direct way, my professors and colleagues and indirectly my family and friends.

From all, a special reference to professor and coordinator Luboš Popelínský, for the incredible kindness, support and presence, received since my arrival at Czech Republic and during my stay. The availability, help and guidance in the whole project was vital for the success of the project.

I thank professor Rui Camacho, for his help in the Master’s thesis text and for all the assistance in the initial phase of the project. The discussion of the Master’s thesis subject between the three parts, student and coordinators in FEUP - Porto, Portugal and Masaryk University in Brno, Czech Republic was conducted in a perfect way.

I thank Jan Blaťák, creator of RAP ILP system, always available for my questions and doubts that occurred and I thank all the Knowledge Discovery Group at Faculty of Informatics of Masaryk University. At last I thank João Conceição, a colleague that followed my work and helped me in the development of the project giving me much support.

As it was my first contact with the new developing area called Data Mining and Knowledge Discovery, I thank the assistance I received in these subjects as well as with Prolog logic programming language basis.

Doing the Master’s thesis under the Erasmus Programme was a very enriching and gratifying experience. The culture in which I was absorbed in fascinated me as well as all the people I have the pleasure to personally meet. Doing the thesis in a foreign country and faculty could seem, at first sight, a tough task but because of the outstanding support that I received it was a very pleasant work that I enjoyed every moment. I learned a lot with the experience and once more, thanks everyone.

Nelson Prazeres Costa
"All men by nature desire knowledge."

Aristotle (384 BC - 322 BC)
Contents

Abstract  iv
Acknowledgements  vi

1 Introduction  1
1.1 Motivation  1
1.2 Work Proposal  2
1.3 Report Structure  2

2 Data Mining  3
2.1 Why use Data Mining?  4
2.2 The Knowledge Discovery in Databases Process  5
2.3 Techniques in Data Mining  6
2.4 Text Mining  7
2.5 Machine Learning  8
2.6 Inductive Logic Programming  9

3 The RAP ILP System  11
3.1 Description of RAP  12
3.2 RAP Main Algorithm  13
3.3 RAP Input Files  14
3.4 RAP Output Files  17
3.5 RAP Settings File  18
3.6 Prolog  19

4 The Graphical User Interface  21
4.1 Requirements  21
4.2 Architecture  22
4.3 Design  23
4.4 Implementation  24
4.5 Testing  25
4.6 Java Documentation  25
4.7 Wrapper Pattern  26
4.8 Limitations  28

5 Conclusions  29
5.1 Future Work  29
5.2 Final Considerations  30
CONTENTS

A  RAP Settings  31
Bibliography  36
List of Figures

2.1 The knowledge discovery process ........................................ 5
3.1 An example of generation of two maximal patterns .................. 12
4.1 The general project architecture ........................................ 22
4.2 Class diagram for the main package ................................... 24
4.3 The project UML package diagram ..................................... 25
4.4 A particular object diagram ............................................. 26
LIST OF FIGURES
Abbreviations

AI  Artificial Intelligence
API  Application Programming Interface
DM  Data Mining
FOL  First Order Logic
GUI  Graphical User Interface
HTML  HyperText Markup Language
ILP  Inductive Logic Programming
KD  Knowledge Discovery
KDD  Knowledge Discovery in Databases
ML  Machine Learning
NLP  Natural Language Processing
RDB  Relational Database
SQL  Structured Query Language
UML  Unified Modeling Language
XML  Extensible Markup Language
Chapter 1

Introduction

In this Master's thesis I present the Graphical User Interface to the RAP ILP system. RAP is an ILP system for mining maximally frequent patterns in first-order logic. Besides the existence of a command line interface to execute RAP, with the construction of this project, a Machine Learning user will be able to build and run a RAP project in an easier way with this more friendly and simple environment.

The development of the interface assumes a complete understanding of the RAP ILP system and general know-how about Data Mining, Machine Learning and Inductive Logic Programming. Therefore in a first stage, the work was based in research and information gather in these fields, with a particular attention in RAP system operation. The second stage consisted in the analysis of existent graphical user interfaces for Inductive Logic Programming systems and in some background knowledge on user interfaces developing and software development process fields. The third and final stage consisted in the application construction. Several phases were here performed, such as the requirements gathering, the architecture and design process, the implementation (source code writing), and final test and debug of the application. This stage, due it’s complexity, consumed most of the project time. The final stage of the work was the documentation process, that included the creation of the user’s manual and the creation of the thesis report.

1.1 Motivation

Analyzing large collections of data is a complex task. Machine Learning based algorithms have been developed recently with the intention to help in the task of Data Mining. One drawback of such algorithms is that they usually require a ML expertise to use them adequately. There are usually a lot of parameters to tune and it is required advanced logic programming knowledge. This drawback strongly limits the algorithms applicability.

To increase the use of such algorithms, proper interfaces must be created in order to hide the algorithm’s parameters from the user and to present the algorithm results in a simple and intuitive form. By doing this, the minimal users know-how about logic
programming and about the particular ML application is reduced, allowing a more wide range of users to use the application.

1.2 Work Proposal

The work consists in the design and development of a Graphical User Interface capable of making data mining tasks more easy with RAP ILP system. The interface will hide the details of the data mining tool, will assist the user in the data mining experiments and will provide an improved presentation of the results. The data mining tool will be the RAP system. The interface will help the user in: i) the data preparation stage of data mining tasks (the processing of the necessary input data); ii) the development of the required background knowledge; iii) automatically tune the parameters (settings) of the RAP ILP algorithm and; iv) showing the results (knowledge extracted) in a comprehensive form. It is intended, by performing these steps, to abstract the user of logic programming knowledge and focus his attention into the algorithm results.

1.3 Report Structure

The present master’s thesis report is structured in three major parts. The first part called Data Mining (2) introduces the field in which this work is inserted, explaining the concepts of Data Mining, Machine Learning and Inductive Logic Programming and their importance nowadays in data analysis. The second part, RAP ILP System (3), explains in detail the system who is the base of the project, with the presentation of it’s structure, algorithm, input and output files, settings and features. An introduction to logic programming language is here presented due to it’s impact in RAP and other logic programming systems processing. At last, the Graphical User Interface (4) is the part where the effective project performed is explained. As any software application, the software development process is described in detail in the chapter as well as the requirements and limitations of the application. Conclusions are at last presented with the introduction of RAP settings complete list as appendix.
Data Mining

Data Mining (or Knowledge Discovery on Databases) is the process of extracting knowledge implicit in large volumes of raw data. Raw data (sometimes called source data) is data that has not been processed for use\(^1\). The knowledge extracted must be new, not obvious, valuable and useful. Note that data is different from information as information is the end product of data processing. But data mining extracts knowledge and knowledge is different from information. As it’s still controversial to establish the definitions and therefore the differences between these two terms, knowledge is defined as ”a fluid mix of framed experience, values, contextual information and expert insight that provides a framework for evaluating and incorporating new experiences and information” (1). Information consists of facts and data organized to describe a particular situation or condition. Knowledge is applied to interpret information about the situation and to decide how to handle it.

Data Mining can be also defined as ”the nontrivial extraction of implicit, previously unknown, and potentially useful information from data (2), ”mechanized process of identifying or discovering useful structure in data” (3) and the science of extracting useful information from large data sets or databases (4).

The term ”Data Mining” was introduced in the 1990s, formed by the conjunction of three fields: Classical Statistics, Artificial Intelligence and Machine Learning. Classical Statistics has a long development history and it’s the main background of Data Mining. Regression and factor analysis, standard distribution, deviation and variance, cluster analysis and confidence intervals are classic statistical techniques used in the data mining process. Artificial Intelligence, built upon heuristics, attempts to apply human-thought-like processing to problems. Data Mining growing is strongly attached with the evolution of AI. Machine Learning joins the classical statistics concepts and AI in order to let computer programs ”learn” about the data they study and extract knowledge through

\(^1\) Although Data Mining is one step in KDD process it is often found in the literature the interchange use of Data Mining and KDD. It will be used the term Data Mining to refer both KDD and the Data Mining step of the KDD process
Data Mining algorithms can be divided into two categories, propositional (classical) and (multi-)relational data mining. While in the first case data is stored in a single table and patterns involve intra-tuple relations, in the second, data is stored in multiple tables with patterns involve inter-tuple or inter-table relations. While relational data mining systems execute a huge number of queries on the database, consuming much CPU time, in propositional systems, data is converted into a single table, where each row describes a specific instance with a fixed set of features or attributes. This conversion process is called propositionalization or feature construction.

The main drawback of propositional systems is that the data has to be converted into a single table, where each row describes a specific instance with a fixed set of features or attributes. Propositionalization (conversion process), is difficult, may require non-trivial human interaction and results in many cases in loss of information. Relational data mining systems do not require this complex conversion step because they can learn from the data in its original representation.

In recent years the most common types of patterns and approaches considered have been extended to the relational case where it’s possible to build models directly from a relational database. The advantages presented by relational data mining systems are that these systems can learn from the data in its original representation and the output can be formulated in terms of the original representation, which makes interpretation of the induced models easier for the domain expert. Inductive Logic Programming (ILP), that is further presented, is a research area of relational data mining.

Propositional and relational systems build two types of data mining models: predictive and descriptive models. In predictive data mining, the goal is to build a model that maps an instance to a valid prediction (as example, a predictive model can predict the mutagenicity of a molecule). Descriptive data mining algorithms are designed to discover interesting knowledge from the data, such as frequently occurring patterns.

2.1 Why use Data Mining?

As the amount of data stored in databases, data warehouses and other information repositories is rapidly growing and this growth by far exceeds the human capabilities to analyze the data contained in those data storage it’s more and more important to discover the knowledge contained in the data. Despite the data can be inserted in a random, unorganized form, the data, whose propose is to be analyzed, is normally set in organized databases by automated data collection tools. Huge databases are nowadays implemented containing enormous volumes of data and a classical human analyst approach no longer makes sense.

*We are drowning in data, but starving for knowledge*
2.2 The Knowledge Discovery in Databases Process

Because databases contain too much data and too less perceptible information and there is a great need to extract knowledge from the data, data mining processes can find relationships and patterns in raw data and the results can be either used in an automated decision support system or accessed by a human analyst. Data mining does not need to analyze all the data contained in the databases but only the specified by the user or algorithm. In some cases, mining all the data in a database would be an impossible task to be performed due to the high computer processing involved.

The powerful multiprocessor computers that are available, the efficient data mining algorithms created and the improvement performed in data collection and management, makes DM the solution for both companies and users willing to extract knowledge through data.

The challenges to be overcome are still several. The fact that most patterns found may not be interesting and may be inexact is common. Another fact is that information can be completely spurious when noisy data is present. To tackle these problems they should be a refinement both in the data mining algorithms and by the user. A pattern is interesting if it is easily understood by humans, valid on new or tested data with some degree of certainty, potentially useful or if it validates some hypothesis that a user seeks to confirm.

2.2 The Knowledge Discovery in Databases Process

Discovering knowledge explicit in raw data is a major challenge. Involving diverse steps knowledge is extracted from data. Data is diverse. It can be a collection of HTML or XML documents, a set of DNA sequences or a number of molecule descriptions. Nowadays a natural way to store data is in a RDB. Figure 2.1 describe the Knowledge Discovery in Databases (KDD) generic process (Fayyad 1996).

![Figure 2.1: The knowledge discovery process](image)
Data selection is the first step performed on the data. A selection is necessary because data is vast and data mining algorithms may be interested in handling with just some sections of the data. The next step is (pre-)processing, it is responsible for displaying the data in a adequate form to the data mining algorithm. This step, in general, is broke through two phases: a pre-processing and processing phase. Transformation produces formatted data, (i.e. normalised and smoothed data, ready for being processed by data mining algorithms). After the data mining process (where a specific data mining algorithm operates on the transformed data) is executed, the information interpretation and evaluation process is performed making this the most important step of all. The KDD process is conducted in a dynamic way with each particular process receiving some feedback from previous processes.

2.3 Techniques in Data Mining

Processing data in order to extract knowledge from it, is a complex process. To obtain structured descriptions from examples, class identification, classification, dependency analysis and deviation detection are the main steps taken. For these processes, the data mining techniques that can be applied are numerous. To build models based on large data sets, six major classes can aggregate these techniques: statistics, nearest neighborhood methods, clustering, decision trees, neural networks and association rules (rule induction). The first three classes are classified as classical techniques and the others as "next generation" techniques. Rule sets, decision trees, instance based representations, neural networks and support vector machines are frequently used predictive models (where the goal is to map an instance to a valid prediction). The different techniques applied for the four major processes in data mining are now described:

Class identification, often the first process used, is based on statistical analysis in order to group database objects into similarity subclasses. Association techniques discovers interesting associations between attributes contained in a database in a multi-dimensional or single-dimensional form.

Classification process is focused in a description of data in a more compact way. Given a database of records, each with a class label, a classifier generates a concise and meaningful description for each class that can be used to classify subsequent records. A set of attribute values defines each record. They are a vast range of techniques proposed for classification including Bayesian classification, neural networks, genetic algorithms and tree-structured classifiers. But the classification technique most used in data mining applications are decision tree based classifiers as they offer a more intuitive representation, easy to assimilate and translate to standard database queries. The efficiency and accuracy of decision trees were superior to the other classification techniques in experimental tests performed (5).
As a classification process, prediction is sometimes used with the existence of unknown or missing data. Clustering techniques, also often employed in classification, is a kind of unsuppressed knowledge discovery or unsupervised learning, that finds appropriate groups of elements for a set of data based on their locality and connectivity within the n-dimensional space. Its principle is maximize the intra-class similarity and minimize the inter-class similarity.

Dependency analysis is the prediction of values of some attributes if knowing the values of others. Decision trees are very popular as they perform classification by constructing a tree based on training instances with leaves having class labels. The tree is traversed for each test instance to find a leaf, and the class of the leaf is the predicted class. This is a directed knowledge discovery in the sense that there is a specific field whose value we want to predict.

At last, deviation detection discovers deviations from the expectations. The use of neural networks is a popular technique explained as a layered set of interconnected processors. The processor nodes are frequently referred as neurodes so as to indicate a relationship with the neurons of the brain. Each node has a weighted connection to several other nodes in adjacent layers. Individual nodes take the input received from connected nodes and use the weights combined together to compute output values.

The results presented can come from an objective and subjective form. Objective measures are based on statistics and the structures of patterns. Subjective measures are not less important and are based on the user’s belief on data. Besides not an easy task, it’s the user responsibility to confirm the results due the fact that absence of critical data or data with errors may lead to wrong results.

Because data is so diverse and can be presented in many forms, with each form having a particular way to handle, numerous sub-fields emerged. Fields like Geographic data mining, Text mining, Web mining and DNA mining are specialized in analyzing and extracting knowledge of its particular data. In the project, text data has an important role and therefore text mining is presented.

### 2.4 Text Mining

Text expresses a vast, rich range of information, but encodes this information in a form that is difficult to decipher automatically (6). The use of text directly to discover heretofore unknown information is a difficult challenge but a evermore appealing area. Note that 90% of the world’s data is hold under unstructured text format.

Text Mining refers to ”the discovery by computer of new, previously unknown information, by automatically extracting information from different written resources” (7) or ”the extract of useful information from data sources through the identification and exploration of interesting patterns” (8). Text Mining uses techniques from data mining,
Data Mining involves the pre-processing of document collections (text categorization, information extraction, term extraction), the storage of the intermediate representations, the techniques to analyze these intermediate representations (such as distribution analysis, clustering, trend analysis and association rules), and visualization of the results (8). Therefore classification, filtering, clustering and association of the text information are the main applications of text mining.

Although rapidly growing in size and importance, this area has great limitations implicitly attached to text mining processing. The majority of information is in a complete unstructured textual form, as a very large number of possible dimensions, inherit to the nature of our languages. Word ambiguity, context sensitivity and noisy data are important challenges to be overcome.

Processing text data is performed by three major steps: i) modeling semi-structured data, ii) information retrieval from unstructured documents and iii) text mining, with the classifying, clustering and patterns finding across documents. These are several techniques on classifying documents, such as decision trees or naive Bayes classifier, on clustering (finding groups of similar documents) and patterns finding. These techniques may be applied independently or together in order to optimize the results.

### 2.5 Machine Learning

Machine Learning is a sub-field of Artificial Intelligence concerned with the design and development of algorithms and techniques that allow computers to "learn". A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance on tasks in T, as measured by P, improves with experience E (9).

Two types of learning can be defined: Inductive and deductive learning. Deductive learning is an approach to language teaching in which learners are taught rules and given specific information about a language. Then, they apply these rules when they use the language. This may be contrasted with inductive learning in which learners are not taught rules directly, but are left to discover (or induce) rules from their experience of using the language.

The main methods in machine learning are supervised and unsupervised learning. In supervised learning, the model defines the effect on one set of observations, called inputs, has on another set of observations, called outputs. In other words, the inputs are assumed to be at the beginning and outputs at the end of the causal chain. The models can include mediating variables between the inputs and outputs and their quality depend closely from the quality of data.

In unsupervised learning, all the observations are assumed to be caused by latent variables, that is, the observations are assumed to be at the end of the causal chain. In
practice, models for supervised learning often leave the probability for inputs undefined. This model is not needed as long as the inputs are available, but if some of the input values are missing, it is not possible to infer anything about the outputs. In unsupervised learning, if the inputs are also modelled, then missing inputs cause no problem since they can be considered latent variables. With unsupervised learning method, theoretical, it will be more easy to find non-obvious patterns in data.

2.5.1 Application Success

Machine learning has already be applied to numerous fields and with successful results. In applications including natural language processing, syntactic pattern recognition, search engines, medical diagnosis, detecting credit card fraud, stock market analysis, classifying DNA sequences, speech and handwriting recognition, object recognition in computer vision, game playing and robot locomotion.

Data mining and relational learning can be applied to many fields like informatics, medicine, engineering and management through different forms such as text, web and spatial mining (10).

2.5.2 Machine Learning Software

There are several tools available in the market that are orientated to data mining. The analytic tools WEKA (Waikato Environment for Knowledge Analysis) (19) and YALE (Yet Another Learning Environment), also known as RapidMiner (20), are open-source software that provides powerful mining algorithms and complete visualization tools to extract knowledge through analysis.

It’s critical emphasize the importance of the visualization tools in the extraction of knowledge from the patterns. Without the ability to see a high-level view of the data, it is easy to get bogged down in the details of a very small part of the data set.

2.6 Inductive Logic Programming

Inductive Logic Programming (ILP) is a research area formed at the intersection of Machine Learning and Logic Programming (11). Machine Learning, as presented, is the acquisition of structural descriptions from examples (12), providing the technical basis of Data Mining with the extraction of information from the raw data in databases (12). Logic Programming refers to a variety of computer languages and execution models based on the traditional concept of Symbolic Logic (13).

Inductive Logic Programming, term introduced by Stephen Muggleton in 1991, is therefore defined as a class of machine learning algorithms where the agent learns a first-order theory from examples and background knowledge (14), proceeding by proof search.
according to a fixed strategy (15) (the particular algorithm). It explores inductive learning in First Order Logic (FOL).

ILP systems construct predicate descriptions from examples and background knowledge. The use of FOL programs as the underlying representation makes ILP systems more powerful and useful than the conventional empirical machine learning systems (14). The examples, background knowledge and final descriptions are all described as logic programs. The advantages of ILP are the possibility to obtain data and hypothesis in first-order logic, the introduction of new information into data and the possibility to the user to incorporate very complex domain specific background knowledge.

The theory of ILP is based on proof theory and model theory for the first order predicate calculus. Inductive hypothesis formation is characterised by techniques including inverse resolution, relative least general generalisations, inverse implication and inverse entailment. The ability of ILP systems to accommodate background knowledge is fundamental.

ILP applicability is huge. Successful results were already achieved from bio-informatics and chemi-informatics (predicting mutagenesis, learning drug structure-activity, predicting protein secondary structure, etc.) to engineering, environmental sciences and NLP. ILP techniques in natural language processing contribute in an important way to the resolution of generic text mining problems, such as the representation of whole sentences in a structured way and the incorporation of additional information (morphology, morphosyntactic relations). Because of many combinations that can appear in NLP, protein databases and XML documents it’s sometimes difficult to analyze the complex data with attribute-value learning systems. Distributed and partitioned ILP systems is the solution and are increasing in number relative to the non-distributed systems.

### 2.6.1 ILP Systems

Several ILP systems were developed and applied to various problem domains. Applications such as Duce, CIGOL, Linus, Golem, Progol and Aleph are successful implementations of ILP.

While Aleph (A Learning Engine for Proposing Hypotheses, Ashwin Srinivasan, 1993) integrates numerous functionalities of other ILP Systems in a way to provide easy and personalized knowledge to the user, Linus (Lavrač and Dzeroski, 1994) incorporates attribute-value learning systems in order to transform a restricted class of ILP problems into propositional form and solve the transformed learning problem with an attribute-value learning algorithm. Several other systems are available and well documented.
Chapter 3

The RAP ILP System

The RAP System was created in 2002 by Jan Blaťák with the collaboration of Luboš Popelínský and Miloslav Nepil. Inserted in the projects developed by the Knowledge Discovery Group (KD Group) at Faculty of Informatics, Masaryk University in Brno, Czech Republic, the main goal of the project was to create a system capable to "mine" maximal patterns in data.

Oriented to be efficient with text data containing long patterns and for feature mining (if the new features are expected to be long), the goal of RAP is to implement a system, capable to find some or all maximal patterns fast (i.e. faster than level-wise algorithms). RAP main advantage is that it usually generates less number of candidates then the systems that employ the apriori-like architecture. There are several fields were the application can be useful, such as analyzing biochemical and/or molecular data and in hyper or normal text data.

RAP is an Inductive Logic Programming system for mining maximal frequent patterns in First Order Logic (FOL). FOL is understood as a formal deductive system that allows a proposition to be expressed as a predicate about (one or more) objects. RAP is oriented to find patterns in multi-relational data.

RAP system implements either partial or complete (with/without loss of information respectively) propositionalization. Propositionalization, consists in the conversion of data to an attribute-value form, followed by a propositional learner function (16). This conversion permit to achieve efficiently frequent patterns that can be successfully used as new boolean features (16).

Parallel and distributed data mining have become very important in the recent time. It can overcome many problems of serial algorithms, particularly the lack of operation memory for storing data or hypothesis, or immoderate time requirements. Both problems play an important role in ILP. Almost all ILP systems are designed to run in main memory, therefore it is difficult to use them for mining large volumes of data because there are too many candidates need to be processed. RAP exploits distributed architecture through dRAP system.
dRAP, a framework for distributed mining first-order frequent patterns extends the RAP system for running on parallel shared-nothing architecture. It utilizes several well-known methods for parallel mining propositional frequent patterns and a new algorithm that minimizes communication overhead. The new algorithm require significantly smaller number of messages passed than the earlier methods.

RAP revealed successful results in propositional feature construction, propositionalization of mutagenesis and carcinogenesis domains, mining in the STULONG database and NLP(18).

3.1 Description of RAP

RAP uses a generate-test paradigm for finding first-order clauses which cover at least N examples (17). The value N is called minimal frequency threshold \( \text{minfreq} \) and it’s an attribute given by the user (17). For a given minimal frequency \( \text{minfreq} \), RAP returns conjunctions of literals from background knowledge that cover at least \( \text{minfreq} \) examples (16).

Three types of search are implemented: random, depth-first search and best-first search (heuristics based search on entropy) (16). RAP may store the information about infrequent patterns and class distribution of patterns to optimize the system performance (16). There is an example 3.1 of a random search by RAP System:

```
Examples = \{ [a,b,c,d],[a,b],[b,c,d],[c,d],[a,c],[b,c],[b,c,d],[a,d],
            [a,c],[c,d],[d],[c],[c,d],[b,c,d],[a,b,c,d],[d] \}
```

![Tree Diagram](image)

- **1st run**
  - \( a_6 \) \( b_7 \) \( c_12 \) \( d_1 \)
  - \( a_3 \) \( c_6 \) \( d_4 \)
  - \( a_2 \) \( c_3 \)

- **2nd run**
  - \( a_5 \) \( b_7 \) \( c_12 \) \( d_1 \)
  - \( a_4 \) \( b_3 \) \( d_4 \)
  - \( b_2 \) \( d_2 \)

- **Infreq** = \{ [b,a],[b,d,a] \}
- **Known** = \{ [b],[b,d],[b,d,c] \}

Figure 3.1: An example of generation of two maximal patterns

Each path in a tree represents a pattern. The number in a node is equal to the frequency of the pattern. Solid lines represent the maximal pattern. Dashed lines identify possible candidates and dotted lines identify known candidates.
3.2 RAP Main Algorithm

Written in Prolog logic programming language, RAP algorithm is supported by three input files and a settings file. The input files needed are: learning set file (filestem.kb), containing the database, domain knowledge (filestem.bg) and a specification file (filestem.s) which contains the language bias definition.

RAP settings file provides general operation options to the user. The main options are filestem (the project name), minfreq (minimal frequency threshold), maxruns (the number of patterns to be generated) and maxlength (maximal pattern length). These and all the settings are forward detailed in the Appendix entitled "RAP Settings".

The RAP algorithm can run in two regimes, GENERATE and COMBINE. While in the first case a new query is built step-by-step employing some of the refinement operators implemented, COMBINE regime consists in two or more maximal queries used for building a prefix of the next maximal query.

As a result, RAP inductive database consists on three predicates: frequent, in the form freq(Query, Vars, Types, Instances), infrequent as infreq(Query, Vars, Types) and maximal as max(Query, Vars, Types, Instances). These predicates refer to the frequent/infrequent patterns known and known maximal queries. As infrequent predicates can be quite huge they can be suppressed.

3.2 RAP Main Algorithm

RAP main algorithm can be very simple described by the following sequence of instructions. Note that this is an abstract presentation of the RAP ILP algorithm (written in Prolog language):

1. \(Q = \{\text{initial query}\}\)
2. while(\(Q\) is not maximal)
3. \(\text{generate}\) some of refinements NewQ of \(Q\)
4. remove all \(Q'\) from NewQ that are infrequent
5. remove all \(Q'\) from NewQ that are known
6. \(Q = \) one of the frequent queries in NewQ

As the above representation is in a very abstract form, it is presented its implementation in more detail. The algorithm outlined below describes the computation of one maximal pattern:

**Input:** Database \(r\), the key predicate used to address examples key, the threshold minimal frequency minfreq and maximal pattern length maxlength

**Structures:** set of infrequent patterns Infreq and set of known patterns Known

**Output:** maximal pattern \(Q\)
1. $Known = \{\}$

2. make initial pattern $Q$ (default $Q = key(Key)$)

3. while pattern $Q$ is shorter than $maxlength$ do

4. generate set $Q_{SPECS}$ of new refinements of $Q$

5. discretize continuous arguments in added literals

6. if $Q_{SPECS} \neq 0$ then

7. select one refinement $Q_e$ from $Q_{SPECS}$ such that $Q_e \notin Infreq$ and $Q_e \notin Known$

8. else

9. pattern $Q$ is maximal

10. end if

11. compute support of $Q_e$ on database $r$

12. if support of $Q_e$ is great or equal than $minfreq$ then

13. $Q = Q_e$ and update structure $Known$ (add $Q_e$ to $Known$)

14. else

15. add pattern $Q_e$ to $Infreq$ and go to Step 5

16. end if

17. end while

3.3 RAP Input Files

The RAP system uses three input files (the language bias definition file (filestem.s), the background knowledge file (filestem.bg) and the database file(filestem.kb)). As information, in a normal setting of Inductive Logic Programmes it is common to have positive and negative example files and a background knowledge file. RAP differs from this structure with the database file compiling the positive and negative examples and the background knowledge separated into the language bias definition and background file. Each input file is next described:
3.3 RAP Input Files

3.3.1 filestem.s

The file filestem.s (where filestem should be replaced by the project name) contains the language bias definition of the RAP project. The user specifies the modes and types of variables and defines constants in the predicates. The file consists on the definition of modes (rmode) type, types and predicates. Classes, lookahead and theory definitions can also appear in this file but its use is for particular cases (requiring a RAP expertise to manage them). In detail:

3.3.1.1 Main specifications:

The specification file must contain the following specifications next described. The rmode and type specifications provide comprehensive information of the predicate structure and types and predicates specifications provide a general view of the project structure.

- rmode(recallNumber, predicate(m1, ..., mN)).

  recallNumber is an integer between 1 and \( \infty \) (no limit) (represented as *). predicate is the predicate name and arguments \( m1 \) to \( mN \) are the arguments of the predicate. Each argument is built in the form mode + variable where mode can be input (+), output (-), input/output (+-) or constant (atom or \#[atom1, ..., atomN] in case of more than one atom) and variable is the name of variable (always in the upper case form). As example:

  - Example: rmode(3, name(+V, -V, #[c1,c2], A, +-U)).

  defines three "calls" to predicate name with five arguments: variable V as input, variable V as output, a constant c1 or c2 as argument three, an atom A and variable U as input/output. The variables should be always represented in an upper case form.

- type(predicate(t1, ..., tN)).

  type is built with predicate name predicate, and its arguments (with the variables in a lower case form). Note that it must be in accordance with the specification made in rmode. As example:

  - Example: type(key(v, u, w)).

  refers to predicate key with three arguments: variables v, u and w.

- types([t1, ..., tN]).

  types organizes all the different types that were inserted in the predicates.

- predicates([pred1/N1, ..., predN/Nn]).

  predicates also organizes the information inserted in the specification file by compiling all the predicate heads followed by the number of arguments.
3.3.1.2 Another specifications:

There are another specifications used in the specification file (filestem.s) but not so often implemented as they usual refer to some particular cases. Classes, lookahead and theory specifications are presented:

- classes([c1, . . . , cN]).

  *classes* specification is necessary in the case the classified data is mined and best-first search is used (eval, consistency and classdiff). In that case the heuristic measures are computed for classes listed in this predicate. There has to be defined class/2 predicate in the data. The predicate has to return for each key identifier a class identifier from a given list.

- lookahead(predicate(m1, . . . , mN)).

  *lookahead* specification allows RAP refinement operator add several literals in one step. Arguments *m1* to *mN* (the body of the predicate) may have an extra type (=) referring to all the variables consumed by the literals.

- theory(predicate1(m1, . . . , mN), predicate2(m1, . . . , mN)).

  *theory* specification refers to an attribution between two predicates. The rule (in general described in the background knowledge file) can be also specified in this file.

3.3.2 filestem.bg

Consists of some appropriate background knowledge (e.g. definition of literals specified in filestem.s). In order to build appropriate background knowledge it’s important to have some experience with ILP systems and programming logic.

- key(X) :- customer(X).

  In the example it is stated that key is referred to customer. Argument association can also be performed in this file.

3.3.3 filestem.kb

Contains the database (examples). As its a goal to work with large sets of data the user usual loads an existent database file (previous processed to be read by the algorithm).

- parent(carol, diana).
  buys(allen, wine).

  The code presented refers to some data that is an pre-processed form and ready to be run by RAP algorithm. The arguments of parent and buys are displayed as shown in the above example.
3.4 RAP Output Files

In this section, RAP output files that are generated are presented. At total, six RAP output files can be generated (but usually less are generated depending on the settings defined): rap.stream, rap.out, rap.successno, rap.disc, rap.entropy and rap.statistics provides specific information about the results achieved. Each file will be next described:

3.4.1 rap.stream

RAP output file "rap.stream" contains all found patterns. The patterns are the following with the respective types and arguments:

- maximal patterns: \( \text{max}(\text{Spec, Query, Vars, Types, Keys}) \)
- sub-maximal patterns: \( \text{submax}(\text{Spec, Query, Vars, Types, Keys}) \)
- frequent patterns: \( \text{freq}(\text{Spec, Query, Vars, Types, Keys}) \)
- infrequent patterns: \( \text{infreq}(\text{Spec, Query, Vars, Types}) \)

This file contains all patterns found. The information contained in this file is huge and it is the result of the RAP algorithm, in a non-refined form. If the user only disposes this file, it is necessary to be a RAP system expertise to manage this information and obtain results.

3.4.2 rap.out

File rap.out can be considered the most important file of all as it contains the maximal patterns displayed in a human readable form. By analyzing the information contained in the "rap.stream" file it provides the most relevant information contained in that file. This allows user’s with less knowledge about Data Mining and Inductive Logic Programming systems to analyze the results.

Query: key(A)
vars: [A]
types: [k]
type: max pattern
length: 0
frequency: 1.0
support: 3

As described above the information is displayed in a simple and comprehensive form with the presentation (for each query) of the variables (vars), types, type (maximal pattern or not), length of the pattern (length), frequency and support.
3.4.3 rap.successno

RAP output file, rap.successno, is not always displayed (the user has the option, in the RAP settings file, to set the displaying or not of this file. In a general way, rap.successno contains a coverage of literals which are specified in the filestem.s input file. As in the next example:

```
successno(parent(.,.), 2).
```

says that the predicate parent (with two arguments) did not "succeed" in establishing parent as a maximal pattern.

3.4.4 rap.disc

The file "rap.disc" is responsible for displaying the information related to discretization. It contains the auxiliary structures used for discretization. Depending on the settings set by the user before running the project, its information may be relevant or not to the user.

3.4.5 rap.entropy

As "rap.disc", "rap.entropy" is responsible for displaying the information related to entropy. It contains the auxiliary structures used for the candidate selection (best-first search) and depending on the settings set by the user before running the project, its information is relevant to the user.

3.4.6 rap.statistics

The statistics file contains some definitions about the execution of the algorithm, such as the particular running time dispensed in each section of RAP algorithm process (select refine, known patterns, infrequent patterns, coverage, data, refine and discretize).

3.5 RAP Settings File

The RAP settings or parameters are vast and are explained in detail in appendix. Here just some considerations are presented:

**Refinement generation:** The refinement operator uses language bias definition to refine the pattern $Q$. There are two possible settings, which drive its function. The system can generate all possible extensions (candidates) of pattern $Q$ or it can generate (randomly) only one possible refinement. It is useful in the case, where system RAP uses depth-first or random strategy.

**Discretization:** In RAP integrated unsupervised discretization method is used. It is based on equal frequency intervals algorithm. This method is not so good as supervised
methods, but it is suitable for frequent pattern discovery task, because it often processes unclassified data.

**Selecting refinement:** System RAP can use one of three available methods, random choice, full depth-first search and best-first search for selecting one refinement from set $Q_{SPECS}$. The first method is suitable for domains with two main classes but it may be used on other datasets too. The second method uses the entropy measure. The system compute entropy for all candidates from $Q_{SPECS}$ and selects pattern with minimal one. The last method uses the confidence measure. The user specifies minimal confidence and the system selects the first pattern whose confidence is greater than given value. The confidence is computed for association rule which has the class identifier in the consequent and Qe in the antecedent.

**Check infrequent patterns:** The system can store information about known infrequent patterns. For some kind of data it is inefficient to use this information because the structure can be quite huge. Using a specified setting, the saving infrequent patterns can be suppressed.

**Check known patterns:** Similar situation can appear when testing whether the new pattern is equal or subsumed by some of known patterns. However, the number of the known patterns is not usually so large. There are three main methods how the patterns are checked. RAP checks if Qe is not subset of or if it is not equal to some known maximal or frequent pattern. This checking can be suppressed for the patterns shorter than a given length.

### 3.6 Prolog

RAP is written in the Prolog language. Prolog is a logic programming language designed by Alain Colmerauer in 1972. It’s a simple, general purpose language often associated for artificial intelligence and computational linguistics implementations. It is based on first-order logic, also known as predicate logic and it stands away from machine logic to approach to human logic. As being a based logical language it’s a declarative and relational style of programming. Prolog differs from the most common programmings languages because it is declarative language. Traditional programming languages are procedural, meaning that the programmer specify how to solve a problem. In declarative languages the programmers only give the problem and the language find the way to solve the problem.

Prolog was one of the first logic programming languages to appear and still remains the most popular. While initially aimed to natural language processing, the language has since then stretched far into other areas like theorem proving, expert systems, games, automated answering systems, ontologies and sophisticated control systems, and modern Prolog environments support the creation of graphical user interfaces, as well as administrative and networked applications.
RAP is implemented in SICStus Prolog engine. SICStus is a Prolog implementation that uses the full virtual memory space and is efficient and robust for large amounts of data and large applications. In SICStus features it’s included the bi-directional interfaces to C & C++, .NET and Java, and Tcl/Tk.

3.6.1 Data Types

Prolog files are represented with the (*.pl) extension. Prolog can be simple written in any text editor with the introduction of a set of clauses describing relations. The clauses contains all the relations that make the program. There are two types of clauses: Facts and Rules. Clauses with empty bodies are called facts (having no subgoals means that the clause is always true/successful). Both facts and rules can contain arguments and it can be any legal Prolog term. Terms are either atoms, numbers, variables or compound terms. Constants start with a lower case letter and differ from variables (that starts with a upper case letter). Pure Prolog is restricted to Horn clauses, a Turing-complete subset of first-order logic.
Chapter 4

The Graphical User Interface

The main task in the Master’s thesis work was the development of the Graphical User Interface to the already existent RAP ILP System. The goal was to produce a friendly interface, intuitive, clean, easy to manage in order to provide the user either the parameters and results of the RAP algorithm (the existent command line interface for RAP system was no more the suitable for this ILP System).

To do such a complex task it’s necessary to structure the development process and in order to do that a standard software development process was taken as model.

Written in Java Programming Language it inherited the advantages of being written in this language. As an object oriented application the code writing was much more simplified. Also, the fact of the interface is based on Java, allows the application to be runnable in all platforms that have a Java Runtime Environment installed.

The project was built underlining the main procedures in the construction of a generic graphical user interface. In the further sections those procedures are described in detail.

4.1 Requirements

The project requirements consisted, in a general form, in providing the user RAP ILP algorithm parameters in a friendly form, providing comprehensive information about those parameters and also displaying the results in a perceptible form. By presenting a friendly interface in which RAP ILP project could be built, it’s expected to reach all level (beginner’s and expertise’s) Machine Learning users and therefore dismistify the fact that it’s required an expertise to run ILP applications. With the performing of the interface, some more particular requirements where gathered in a dynamical way through discussion with ILP expertise’s.
4.2 Architecture

The architecture of a system is its top-level description. The architecture comprises all the software components, their properties and the relationships between them. Sometimes defined as strategic design, it provides the developer a high-level perspective of the application.

![Figure 4.1: The general project architecture](image)

Figure 4.1 displays the general architecture of the project. Given (or setting) a specific workspace directory all the project is here designed. The only external application attached is the Java Virtual Machine, that is called from the workspace when the application is executed. The main component in the workspace is the file "main.jar". This file is in an executable form and it's where the user should execute the application. They are four folders attached to the project correct execution: RAP ILP folder, SICStus Prolog folder, the examples and library folders. RAP ILP folder contains RAP ILP system, displayed in a set of Prolog files it's in this folder that the RAP algorithm is performed and the results are created. SICStus Prolog is the compiler for RAP. As it is necessary SICStus Prolog engine to run RAP system, it must be attached in the project. In the folder examples, they are several project implementations that serve as a support for new projects and as
4.3 Design

a tutorial for beginner user's. Library folder contains application components, such as images and icons, responsible for the correct functionality of the software.

The bottom components, input and output files and the RAP settings file (inserted in a dotted box due they are only temporary files), are created with the project creation (input and RAP settings files) and, if the project is correctly executed, the output files are created by the ILP algorithm.

4.3 Design

Before the implementation of the interface, comprehensive models were designed. The objective was to acquire some background to the further interface implementation. As all projects, the initial design was object of modifications and the final structure of the system was quite different from the initial study. In the next section several models/diagrams are presented containing the system architecture through different perspective supported by other models with use case examples, sequence charts, etc. These diagrams were built under Unified Modeling Language (UML) notation. UML is the standard for modeling object oriented programs.

4.3.1 Class Diagram

The Class diagram (see Figure 4.2) describes the static structure of the project by presenting the classes built and the relations between them:

The complexity of the project demanded a complex project structure. Five packages and twenty nine classes were built with a complex chain of relationships between them. Therefore, in figure 4.2 it’s only presented the class diagram for the main package as it’s the core of all packages and contains the main structure of the project.

4.3.2 Package Diagram

A top-level structure is presented in the package diagram. The objective is to organize elements into groups and therefore reduce the dependencies between packages. The package diagram obtained (as the result of the project implementation) contains five packages with the following relationship between them described in 4.3.

As the classes are not described, it’s not possible to see in a more specific way how the packages interact. However with this diagram it’s shown that the packages ensure a certain independence from each other and have very distinct roles in the project.

4.3.3 Object Diagram

The Object diagram (see Figure 4.4) describes the static structure of a system at a particular time. As they are numerous states where the object diagram could be built,
the particular time analyzed in this diagram is in a random input operation by the user in the specification file (filestem.s).

As shown in figure 4.4, a simple text input in a project specification file produced changes in several application parameters. While the status bar provided comprehensive information about the particular operation, variables were changed and listeners were fired without the knowledge of the user.

## 4.4 Implementation

The implementation phase consumed the most time of the project. Eclipse development environment was chosen as the application where the project code should be written because of its great capabilities, functionality and internal Java Virtual Machine. The internal GUI builder from this Java development tool simplified the initial interface building and the component interaction. Also the JavaDoc builder induced this choice.
4.5 Testing

Software testing is the process used to assess the quality of the application. Testing software is done in several forms but the goal is always to correct software problems, also known as software bugs, so each line code is executed as it was supposed to. The project test phase consisted on a condensed experimental analyzing and dynamic debugging.

4.6 Java Documentation

The Java Documentation, also known as JavaDoc was performed, enabling all the application code, a documented support. JavaDoc is a tool for generating API documentation into HTML format from document comments in Java source code. For further developing or just for code analyze, in this documentation it is shown how the application operates and how the code is organized. As the code will be open-source, it’s intended also to help java developers, particular Graphical User Interface developers in the construction of their own applications.
4.7 Wrapper Pattern

The developing of a wrapper pattern, known also as an adapter pattern, was one of the crucial steps that were taken in the interface construction. A wrapper pattern is an adapter that allows two incompatible interfaces to work together. The wrapper is responsible for adapting one interface into another that is understood by the second interface, becoming itself a bi-directional interface where commands can be understood by both interfaces after a "wrapping" process. In this particular case, a wrapper was designed in order to allow the compatibility of prolog and java interface.

In order to built the adapter pattern, several ideas were analyzed. One consisted in the use of a java class that worked as an interface using pipelines. Pipelines are a set of processes, operated in Linux and Unix-like operating systems, that send the output of one program to another program for further processing. After analyzing and testing this possible implementation, the results that were obtain lead to an unstable configuration. However part of the idea was used in the final configuration further described. The second idea that was studied was the use of Jasper Java library. Jasper is an adapter pattern that was built by SICStus, a Prolog development system, focused in providing compatibility between SICStus Prolog system and Java. Because it is SICStus the prolog engine used by RAP ILP system, the applicability of Jasper is perfectly oriented to the project requirement. Jasper provides a Java library containing classes that can perform any operation done in SICStus prolog run-time system. The classes under Jasper library are adapter classes that transforms the parameters received Through Java into parameters
understood by SICStus prolog engine. After many experiments with Jasper library, unsuccessful results were obtained. The reason was an incompatibility between RAP algorithm and the existent Jasper commands responsible to run the algorithm. As the Jasper library is still in development and with strong limitations for particular operations, its use was declined.

Undertaking the idea of sending standard output as standard input to another program, the construction of the adapter was done in a simple form. Using java Runtime library, responsible for providing an interface with the standard environment (operating system terminal) in which the application is running, an auxiliary application is called in order to interact with SICStus prolog engine.

The auxiliary application is a simple shell file that is called by Java with the introduction of one parameter, the current directory path.

```
Runtime().getRuntime().exec("./file.sh " + current.path)
```

Java Runtime library just executes the file and the several commands inserted in the shell file are handled independently from Java. The information contained in this file is here presented:

1. `#!/bin/bash`
2. `export CUR_PATH=$1`
3. `export RAP_ILP_ROOT=$CUR_PATH/RAP/`
4. `export LD_LIBRARY_PATH=$CUR_PATH/sicstus-3.9.1/lib/`
5. `export SP_PATH=$CUR_PATH/sicstus-3.9.1/lib/sicstus-3.9.1`
6. `alias rap='sicstus -l $RAP_ILP_ROOT/rap.pl'`
7. `$CUR_PATH/sicstus-3.9.1/bin/sicstus -p $CUR_PATH/sicstus-3.9.1/bin/sp-3.9.1/sicstus-3.9.1/bin -l $CUR_PATH/RAP/rap.pl --goal rap.`
8. `*$`

The shell file above presented refers simple commands that are executed by the local console. Current path, containing the current workspace directory, receives its value as a standard input from the Java interface. RAP ILP root and SICStus library and SP paths are also defined and sicstus is executed with arguments -p and -l enabled. At last the process is killed.

Note that if the user changes any of the above paths, he should also change the paths inside the shell file as they are static instances.
With the use of this auxiliary application, the parameters that have to be send from Java to SICStus prolog dont need to be "wrapped" but simply passed through the command line interface, provided by Java Runtime library, to the auxiliary program, as standard input arguments. As the parameters dont need to be "wrapped" because of the use of an external interface, it is not correct to say that a wrapper pattern was implemented. But, as this solution revealed to be the most stable and simple, it was developed and concluded as better solution than the initial idea of building an internal Java adapter pattern to transform the Java output parameters into a perceptible form to be run in SICStus prolog engine.

4.8 Limitations

With the current version of the project, some limitations exist. Working with the project under the conditions under described may lead to problems and incorrect results. The interface was only tested in Linux and Unix-like operating systems and besides be designed to all applications, the results of the RAP algorithm are only obtained under these platforms. The reason for this limitation is known and can be easily solved. The reason is that SICStus prolog engine that was provided to the development of the work is oriented to work only under Linux platform. Besides the existence of SICStus prolog under other platforms, such as Windows, Solaris and Mac operating systems, it was not possible to obtain the binary distributions for these platforms and therefore to test the application in these operating systems.

The interface was tested with Java Runtime Environment Standard Edition 6 (Java SE 6). Using the interface under lower Java versions may lead to unknown problems and therefore it is not recommended. The several tests that were performed before deployment used RAP ILP system version 2. Although it is possible to use RAP system in version 1, the user must be aware of the commands and notation that the deprecated version accepts.

RAP system was developed to work under SICStus prolog version 4. Trying to obtain RAP results with or without the interface under SICStus prolog version 3 is also deprecated.
Chapter 5

Conclusions

After performing this challenge of developing an interface for data mining with RAP system, the main goal was achieved. With the result of this work it’s possible to any user develop an ILP Machine Learning project, based on the RAP algorithm, in order to find maximal patterns in first-order logic, in a user-friendly way.

As one of the requirements, RAP Inductive Logic Programming system now reaches more and more diverse users. Bringing a larger community of users to work with ILP and Machine Learning applications is very positive but it is still necessary a very complete understanding of the patterns extracted in order to analyzed them and get the necessary knowledge. Experience in data mining implementation is so, very important characteristic for any user.

The thesis development was conducted in a enjoyable way, however, as always, some obstacles had to be overcome. The language barrier, was sometimes a challenge. RAP documentation was written in czech language and therefore there had to be an extra effort to gather the necessary information required. Another hold back was related to the RAP system (that is the core of the interface) being currently in a running state. Not necessary an obstacle, it led to some problems in the parameters definition and compatibility.

This thesis covered the implementation of the Graphical User Interface to the RAP Inductive Logic System, describing also the RAP description and features. While the first chapter described the field in which the project is inserted in, the second chapter, RAP ILP System, is presented as the core of the interface.

The third chapter shows the implementation of the interface. By presenting the several steps that were taken it the development process it is explained the whole construction of the project in detail.

5.1 Future Work

Besides the results accomplished, the Graphical User Interface can be improved and refined. Because Data Mining is a complex task, the Machine Learning algorithms are also
complex. The future work will be focused in providing more assistance in the parameters specification and in the compilation process, allowing a higher probability of building a successful project (with successful results) by user’s with no or few experience in Machine Learning systems. As the current project is running only under Unix-like operating systems, it is a goal to extend to the other operating systems managing this way a bigger range of users.

5.2 Final Considerations

As this thesis was focused in a process often neglected in the development of Machine Learning applications (the creation of a visualization tool), it’s expected to serve as a base for further works done with these applications. Associating so distinct languages as it is Prolog and Java into one project was a difficult task, ceased into a original solution. It seems necessary to Java developers evolve libraries capable of providing a stable interface between Java and Prolog.
Appendix A

RAP Settings

RAP ILP system settings are described in detail in this appendix. For each setting it is presented it’s definition, the value type (such as boolean, string, integer), the possible options and an example. Through RAP settings file (file rap.settings in the project workspace) or from the graphical user interface, the user can view and change the values for the several settings.

- **filestem** - The project name where the algorithm will process.
  
  *Type:* string
  
  *Example:* :- set(filestem, projectName).

- **minfreq** - Minimal frequency threshold - number of examples covered (in %).
  
  *Type:* integer
  
  *Options:* 1 to 100
  
  *Example:* :- set(minfreq, 10).

- **maxlength** - Maximal number of literals that can appear in the pattern (length of the pattern).
  
  *Type:* integer
  
  *Example:* :- set(maxlength, 5).

- **maxruns** - Maximal number of patterns generated.
  
  *Type:* integer
  
  *Example:* :- set(maxruns, 1000).

- **select_ref** - Method used for selecting a pattern from a candidate set.
  
  *Type:* string
  
  *Options:* depth-first - depth-first search, random - performs a random search, eval - search based on the information gain, consistency - search based on confidence and classdiff - search based on accuracy
  
  *Example:* :- set(select_ref, random).
• **not_known** - Pruning method (tests known length).
  Type: integer
  Options: 1 to 4
  Example: :- set(not_known, 2).

• **discretise** - Enables discretization.
  Type: string
  Options: no - no discretization method and *equalfreq* - use discretization (discretization frequency must be defined at *disc_eqf_freq* setting)
  Example: :- set(discretise, equalfreq).

• **disc_eqf_freq** - Discretization frequency (in %) (relative size of intervals when equalfreq discretization method is applied.
  Type: integer
  Options: 1 to 100
  Example: :- set(disc_eqf_freq, 50).

• **use_freq** - Enables frequent predicate.
  Type: boolean
  Example: :- set(use_freq, yes).

• **save_freq** - Saves frequent predicate.
  Type: boolean
  Example: :- set(save_freq, yes).

• **use_infreq** - Enables infrequent predicate.
  Type: boolean
  Example: :- set(use_infreq, yes).

• **save_infreq** - Saves infrequent predicate.
  Type: boolean
  Example: :- set(save_infreq, yes).

• **maxgen** - Generates maximal predicates.
  Type: boolean
  Example: :- set(maxgen, yes).

• **save_max** - Saves maximal predicates.
  Type: boolean
  Example: :- set(save_max, yes).

• **save_disc** - Saves discretization (applies only when discretization method is enabled).
  Type: boolean
  Example: :- set(save_disc, yes).
• **save_entropy** - Saves entropy (applies only when `select_ref` has value `eval`).  
  *Type:* boolean  
  *Example:* :- set(save_entropy, yes).

• **consistency_assert** - Asserts consistency (applies only when `select_ref` has value `consistency`).  
  *Type:* boolean  
  *Example:* :- set(consistency_assert, yes).

• **consistency_save** - Saves consistency (applies only when `select_ref` has value `consistency`).  
  *Type:* boolean  
  *Example:* :- set(consistency_save, no).

• **disc_eqf_less** - Allows smaller intervals when discretization is enabled (discretise setting has value `equalfreq`).  
  *Type:* boolean  
  *Example:* :- set(disc_eqf_less, no).

• **testinfreq** - Tests infrequent predicates.  
  *Type:* boolean  
  *Example:* :- set(testinfreq, yes).

• **ordered** - Orders the patterns.  
  *Type:* boolean  
  *Example:* :- set(ordered, no).

• **save_settings** - Saves RAP settings information.  
  *Type:* boolean  
  *Example:* :- set(save_settings, true).

• **verbose** - Sets verbose level.  
  *Type:* integer  
  *Example:* :- set(verbose, 1).

• **verboseA** - Sets verbose level in context discEqF (discretize frequency).  
  *Type:* integer  
  *Example:* :- set(verboseA, (discEqF, 0)).

• **verboseA** - Sets verbose level in context disc (discretize).  
  *Type:* integer  
  *Example:* :- set(verboseA, (disc, 0)).

• **verboseA** - Sets verbose level in context main.  
  *Type:* integer  
  *Example:* :- set(verboseA, (main, 1)).
• **verboseA** - Sets verbose level in context refine.
  
  *Type:* integer
  
  *Example:* `:- set(verboseA, (refine, 0)).`

• **verboseA** - Sets verbose level in context not_known.
  
  *Type:* integer
  
  *Example:* `:- set(verboseA, (not_known, 0)).`

• **verboseA** - Sets verbose level in selectref and entropy context.
  
  *Type:* integer
  
  *Example:* `:- set(verboseA, ((selectref, entropy), 0)).`

• **version** - Refers the RAP system version that is used.
  
  *Type:* string
  
  *Options:* `v1` - version 1 (deprecated) and `v2` - version 2
  
  *Example:* `:- set(version, v2).`

• **refine_start_maxpatterns** - Generates first number of maximal patterns (#count).
  
  *Type:* integer
  
  *Example:* `:- set(refine_start_maxpatterns, 0).`

• **regime** - Sets the regime to be implemented.
  
  *Type:* string
  
  *Options:* `refine` - generates new patterns from scratch and `combine` - generates patterns from the known ones
  
  *Example:* `:- set(regime, refine).`

• **search** - Sets the search strategy to be implemented.
  
  *Type:* string
  
  *Options:* `depth_first`  
  
  *Example:* `:- set(search, depth_first).`

• **testknown** - Tests known in frequent predicate.
  
  *Type:* string
  
  *Options:* `beforefrequent`, `afterfrequent` and `no`
  
  *Example:* `:- set(testknown, beforefrequent).`

• **refine** - Generates operator references.
  
  *Type:* integer
  
  *Example:* `:- set(refine, 40).`

• **frequent** - Tests frequent method.
  
  *Type:* integer
  
  *Example:* `:- set(frequent, 3).`
• **save_successno** - Saves successno file.
  
  *Type:* boolean
  
  *Example:* :- set(save_successno, true).

• **not_known_pref_ln** - Sets not known prefix length.
  
  *Type:* integer
  
  *Example:* :- set(not_known_pref_ln, 2).
Bibliography


