Summary

The first aim of this project is to develop a new Knowledge Discovery (KD) Workbench for Structured data. This is a new platform that provides an integrated framework for multi-relational learning methods (such as Inductive Logic Programming, First Order Decision Trees) and provides interface to existing data mining workbenches and systems. The second aim is to develop a common file format to allow interoperability of the different systems and their proprietary file formats.

The objectives of the project as following:

- Studying relevant literature on KD and Logical Programming (LP) and focus on some Structured data that is the foundation of studying these different file formats
- Learning the different input files, which cover all integrated key state-of-the-art methods, is helpful to develop common file
- Creating a common file that is used to store data from other file formats and also can transfer data to these file formats
- Studying previous workbenches can provide a structure of developing a new workbench, which could maximize the advantages and minimize the drawback
- Designing a friendly interface that can make a better work or study environment

This project is mainly Type I (software development), as 70%, and also involves Type II (Investigatory), as 30%. At the beginning of the project, the background knowledge of structured data and each key state-of-the-art method have been investigated, which is the foundation for developing workbench. After that, the mainly work is programming to realize the functions of interface and background of the workbench.

JSP +Web Standard methods have been used to develop the workbench. After users upload available files, common file would be generated, and at the same time other systems' files would also be created by the common file. In the second page, users can amend data with the prompts in the accessory panel of the page. Thirdly, the result can be seen after users choose one of the algorithms to run. Finally, the results, the amended data file and common file can be downloaded. The minimum criterion in Research Review has been completed and partial second and third criterions also have been done.

The following list is the important elements that have been accomplished:

- Common file can store all information from any available input files to make a common file format where these input files can share their data and also keep their own specific parameters
- Web Standard workbench that makes the connection convenient in any operating systems and without install any software
- A friendly interface of the workbench based on the technique of HTML+CSS+JavaScript
- Giving correlative prompts when amending data
- Successfully running the systems in the workbench

Abstract

The project introduces an integrated Knowledge Discovery Workbench for Structured Data, which includes multiple data mining systems such as Progol, ACE, 1BC, Prism and Weka. The workbench not only can run each system successfully, but also provide a common file, which can help client transfer their data files into other file formats automatically. During client operate the workbench, a list of useful prompts will be provided to amend data efficiently. In addition, J2EE has been used to develop the workbench, which means this workbench can be provided online. Any client can use this workbench through Internet without install these data mining systems. For analyzing a data file, there are basically three steps that are uploading, amending and results need to be operated in the workbench. For changing data format, basically four steps are needed, which are uploading, amending, saving and result. It is always easy to be operated. Therefore, based on the reasonable control panels design, this workbench is absolutely useful, helpful and friendly for users to analyze data.

Acknowledgements

I am especially grateful to my supervisor Oliver Ray for his guidance, support and encouragement throughout the project.

Thanks to Bin Li for providing free J2EE Education Video.

I thank to Jun He for her many sacrifices during the project.

I am grateful to my family for their constant support and encouragement.

Contents

List of Figures

1 INTRODUCTION

1.1 Background

Information has played an important role in our lives today. Hundreds of thousands of new information are generated in every second and include a majority of connotative but more effective knowledge for people, which can help us to make a giant improvement. Data Mining (DM) methods then are used in the process, which we called Knowledge Discovery (KD), to reveal new pieces of connotative knowledge from enormous databases [7]. Because the amount of information was huge, we need to analyze the data with the help of computers instead of labour power. Therefore, structured data or suitable representation of data, which can be used by computers, will be required before KD. Attribute-value Learning (AVL) is one of the most common structured learning representations. Learning representation also encompasses Boolean Logic Learning (BLL), Multi-Instance Learning (MIL), Multi-Relational Learning (MRL) and Inductive Logic Programming (ILP). In this project, structured data mainly indicate the five data learning. As shown in Figure 1.1, it can generally describe the process of KD and some tools will be used during the process, which means users could have interaction with the system. There is also a loop, which is shown in wider line, to make a dynamic feedback loop [7]. Nowadays, the similar structure of KD process is still used, like Weka [9].

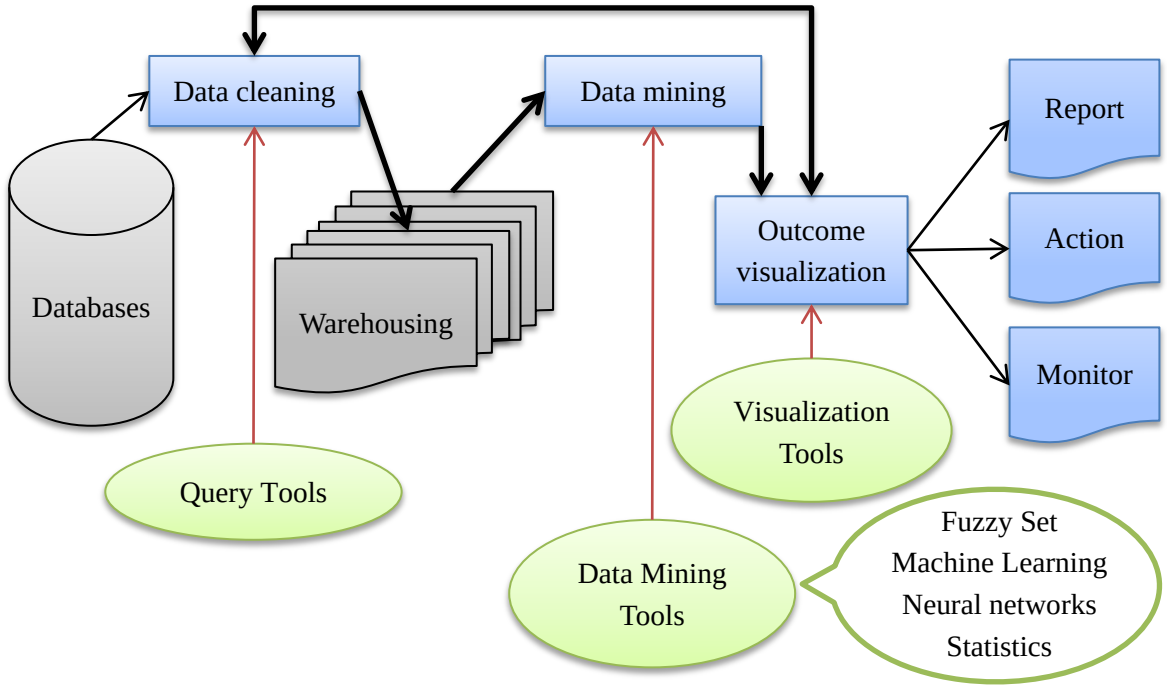


Figure 1.1 Main functional phases of the KD process [7]

One of the sufficient methods for KD is AVL, even if it has some limitations when they represent relationships between items. A number of algorithms have been developed and applied in both academia and business for wide variety structured learning. If most of these algorithms can be integrated into a single cohesive piece of software, it would be very helpful to analyze complex data. Workbench could be the best approach to do this. There are some subsistent workbenches for AVL that bring a wide variety of different AVL techniques into a single environment. One of the most famous workbenches is

the Waikato Environment for Knowledge Analysis (Weka), which was developed and distributed by the University of Waikato [9]. In addition, ACE is another successful data mining workbench that can be used at MRL and ILP level [2]. It also provides a common interface to a number of data mining systems, like Tilde and Warmr. There are several advantages of workbench. First reason is that using workbench can unify the learning representation. Secondly, workbench can supply a visual interface, which can provide an intuitive way to execute different functions of the workbench, rather than using command line interface. Thirdly, the workbench could be assembled by many algorithms.

1.2 Motivation

After learning couple of data mining systems, some disadvantages can easily be found. For example, all of them have different file format, even some of them deal with same type of data. In addition, some data mining system still use command line to operate, like Progol and ACE. It is very inconvenient. Based on such drawbacks, although these workbenches or systems are already successful, the more powerful workbench is still required.

This project not only has been considered using them, but also using other kinds of structured learning representations. Hence, more famous workbenches or systems have been studied in details, like 1BC and Prism. Currently, there is no such workbench that can provide an integrated platform to suit all learning representations and to convert data format between different systems or algorithms. Basically, the new workbench is considered to fix the drawbacks in this project. At the same time, it should keep these advantages as many as possible.

1.3 Challenge

First of all, the new workbench needs a new structure instead of the traditional one. Progol, ACE, 1BC, Prism and Weka are integrated in this workbench. Therefore, finding a suitable structure of the workbench is the first challenge. Secondly, for meeting the demands of integrating many systems, the workbench needs having a most general file format that can store the data from different file format, and also can convert data into different file format. Finding a common file format would be the one of the hardest challenge in this project. Thirdly, after the common file has been defined, the next challenge would be to convert data. 100 present of accuracy rate should be guaranteed during the conversions. Fourthly, for more convenient, the workbench should provide more detail prompts to users during their uploading and amending files. How the workbench checks the warnings and errors of their files is also a challenge. Finally, the last challenge in this project would be running the data file using specific systems. These systems require different input before users can get result. How to handle the interactivity between the users and the systems is the last challenge in this project.

1.4 Aims and Objectives

The first aim of this project is to develop a new Knowledge Discovery (KD) Workbench for Structured data. This is a new platform that provides an integrated the key state-of-the-art methods, such as Progol, ACE, 1BC, Prism and Weka. Therefore, the workbench will run with different Inductive Logical Programming (ILP) and Attribute Value (AV) learning systems. In addition, since each system or

algorithm has their own data format, the second aim is to realize the conversion between different file formats based on common file.

In order to fulfill the aims above, the objectives of the project as following:

- Studying relevant literature on KD and Logical Programming (LP) and focus on some Structured data that is the foundation of studying these different file formats
- Learning the different input files, which cover all integrated key state-of-the-art methods, is helpful to develop common file
- Creating a common file that is used to store data from other file formats and also can transfer data to these file formats
- Studying previous workbenches can provide a structure of developing a new workbench, which could maximize the advantages and minimize the drawback
- Designing a friendly interface that can make a better work or study environment

1.5 Outline

Chapter 1 introduces the background of the KD and structured data. Some successful data mining systems is described to elicit the motivation of the project. Based on the new workbench, some challenges are listed to show the level of difficult of developing the workbench. Finally, aims and objectives are given.

Chapter 2 explains the machine learning representations and the hierarchy of them and also introduces the propositionalization. Some systems are described focus on their data format.

Chapter 3 illustrates some famous previous works on KD workbench.

Chapter 4 demonstrates the methodology, which includes Needs Analysis and Detailed Design.

Chapter 5 illustrates all the result found by the project, which include the interface and background of the workbench. A number of figures explain the results.

Chapter 6 summaries the main conclusion of the thesis based on the common file format, structure of the workbench and evaluations.

Chapter 7 lists several future works for improving the workbench.

2 BACKGROUND

The general knowledge of structured data learning is introduced in this chapter, which includes machine leaning representations and logical programming. In addition, there are five learning systems, Progol, ACE, 1BC, Prism and Weka, are described that include input and output file formats.

2.1 Logical Foundation

2.1.1 Concepts of Clausal Logic

First of all, the logical notation employed throughout this report, is given in the following examples, which is based on [16].

Example 2.1.1. The information about publications, authors and citations, stores in the database. The sentence $\text{authorOf}(\text{lloyd}, \text{logic\_for\_learning}) \leftarrow$ for the relation $\text{authorOf}/2$ represents the fact that lloyd is the author of the publication logic_for_learing. Similarly, if we want to represent that foundations_of_lp is the bibliography of logic_for_learing , we can use $\text{reference}(\text{logic\_for\_learning}, \text{foundations\_of\_lp}) \leftarrow$. [16]

The whole database is shown below for *Example 2.1.1*:

$\text{authorOf}(\text{russell}, \text{ai\_a\_modern\_approach}) \leftarrow$
$\text{authorOf}(\text{quinlan}, \text{learning\_logical\_definitions\_from \_relatons}) \leftarrow$
$\text{authorOf}(\text{lloyd}, \text{logic\_for\_learning}) \leftarrow$
$\text{authorOf}(\text{lloyd}, \text{foundations\_of\_lp}) \leftarrow$
$\text{authorOf}(\text{muggleton}, \text{ilp\_theory\_and\_methods}) \leftarrow$
$\text{authorOf}(\text{norvig}, \text{ai\_a\_modern\_approach}) \leftarrow$
$\text{authorOf}(\text{deraedt}, \text{ilp\_theory\_and\_methods}) \leftarrow$
$\text{reference}(\text{logic\_for\_learning}, \text{foundations\_of\_lp}) \leftarrow$
$\text{reference}(\text{logic\_for\_learning}, \text{learning\_logical\_definitions\_from \_relatons}) \leftarrow$
$\text{reference}(\text{logic\_for\_learning}, \text{ai\_a\_modern\_approach}) \leftarrow$
$\text{reference}(\text{ai\_a\_modern\_approach}, \text{foundations\_of\_lp}) \leftarrow$
$\text{reference}(\text{ai\_a\_modern\_approach}, \text{ilp\_theory\_and\_methods}) \leftarrow$
$\text{reference}(\text{ilp\_theory\_and\_methods}, \text{learning\_logical\_definitions\_from \_relatons}) \leftarrow$
$\text{reference}(\text{ilp\_theory\_and\_methods}, \text{foundations\_of\_lp}) \leftarrow$

Even though the database just contains a small data, it still could be used to introduce some key concepts of knowledge representation formalism. In logic, a relation is called *predicate p/n* ($\text{authorOf}/2$) where *p* denotes the name of the predicate (*authorOf*) and *n* indicate the number of arguments the predicate has (*2*). *Constants* always start with lowercase character, which are a particular type of terms, such as lloyd, logic_for_learning. All data in the database in Example 2.1.1 are *facts*, which are represented by $p(t_1,\cdots,t_n) \leftarrow$, where p/n is predicate and t_i are terms.

Example 2.1.2. The query $\leftarrow \text{authorOf}(\text{lloyd}, \text{logic\_for\_learing})$ asks whether lloyd is the author of logic_for_learing or not. A theorem prover, such as Progol (will be introduced in the next section), would return the answer yes, when it is true. Basically, the facts are true. [16]

When the arguments in the query start with an uppercase character, such as $\leftarrow \text{authorOf}(\text{lloyd}, \text{Publication})$, the term Publication is called a *variable*. The theorem proves would return the answer yes and would also return the substitutions $\{\text{Publication}/\text{logic\_for\_learning}\}$ and $\{\text{Publication}/\text{foundations\_of\_lp}\}$ (the definition of substitutions explained later), which are true under the original query.

Example 2.1.3. If the query is required to find out whether there is an author who wrote the books of logic_for_learing and foundations_of_lp , we can use the query $\leftarrow \text{authorOf}(\text{Author}, \text{logic\_for\_learing}), \text{authorOf}(\text{Author}, \text{foundations\_of\_lp})$. The single substitution is $\{\text{Author}/\text{lloyd}\}$.

Therefore, the "," between the two atoms in the query corresponds to *and*. The variable much be instantiated to the same constant in the substitutions. If we want to find the authors who cite one another, the following query can be used:

$$\leftarrow \text{authorOf}(\text{A}, \text{P}), \text{reference}(\text{P}, \text{Q}), \text{authorOf}(\text{B}, \text{Q})$$

Many answers would be generated for this query, like:

$\{\text{A}/\text{lloyd}, \text{P}/\text{logic\_for\_learning}, \text{Q}/\text{foundatoins\_of\_lp}, \text{B}/\text{lloyd}\}$

$\{\text{A}/\text{lloyd}, \text{P}/\text{logic\_for\_learning}, \text{Q}/\text{ai\_a\_modern\_approach}, \text{B}/\text{russell}\}$

Example 2.1.4. Instead of the query above, we can define a new predicate $\text{cites}/2$ for instead of the above query, like $\text{cites}(\text{A}, \text{B}) \leftarrow \text{authorOf}(\text{A}, \text{P}), \text{reference}(\text{P}, \text{Q}), \text{authorOf}(\text{B}, \text{Q})$. The clause indicates that A cites B if A is the authorOf P , P references Q, and B is the authorOf Q [16]. A theorem prover, such as Progol, these kinds of predicate are called *rules*.

A *term t* is a constant lloyd, a variable Publication or a compound term $f(\text{t}_1, \cdots, \text{t}_\text{n})$ composed of a function symbol f/n, *f* is the name, *n* is terms t_i. For instance, logic_for_learning (constant), Publication (variable). [16]

An *atom* is a formula of the form $p(\text{t}_1, \cdots, \text{t}_\text{n})$, where p/n is the symbol, *p* is then predict, *n* is terms t_i. For instance, authorOf(lloyd, foundations_of_lp), reference(P, foundations_of_lp). [16]

A *substitution* $\theta = \{\text{V}_1/\text{t}_1, \cdots, \text{V}_\text{n}/\text{t}_\text{n}\}$ is an assignment of terms $\text{t}_1, \cdots, \text{t}_\text{n}$ to variables $\text{V}_1, \cdots, \text{V}_\text{n}$. The formula is $\text{F}\theta$, where F is a term, atom or clause (explain later). For instance, the formula $\text{authorOf}(\text{X}, \text{Y})\theta$, where $\theta = \{\text{X}/\text{ lloyd}, \text{Y}/\text{logic\_for\_learing}\}$.

Although terms and atoms have a similar syntax, the meanings of them are quite different. When looking at ground terms and atoms (ground means using only constant), it can be easy to distinct. Ground terms represent objects in the domain, whereas ground atoms represent a relationship between the objects, which is denoted by the terms. Therefore, ground terms do not possess truth-values, but ground atoms do. For instance, it does not make sense to have the truth-value of constant lloyd, or authorOf(lloyd). However, the ground atoms $\text{authorOf}(\text{lloyd}, \text{foundations\_of\_lp})$ (for predicate authorOf/2) would be true. [16]

2.1.2 The Syntax of Clausal Logic

Secondly, we will introduce the definition of clauses, which are the key constructs in clausal logic.

Definition 2.1.5. A clause is an expression of the form $h_1;\cdots;h_n \leftarrow b_1,\cdots,b_m$ where the h_i and b_j are logical atoms. [16]

The symbol "," stands for conjunction (and) and the symbol ";" for disjunction (or), and "←" for implication (if). $h_1;\cdots;h_n$ is referred to as the head of the clause, $b_1,\cdots,b_m$ as the body of the clause. Formally, a literal is either a logic atom b (a positive literal with atomic part A) or a negated atom $\neg b$ (a negative literal with atomic part A). Therefore, we can represent the clause in the *Definition 2.1.5* as the set $\{h_1,\cdots,h_n,\neg b_1,\cdots,\neg b_m\}$. In addition, the set notation also can be written as a disjunction $h_1 \vee \cdots \vee h_n \vee \neg b_1 \vee \cdots \vee \neg b_m$. [16]

We can set the n and m in different values to generate different types of clauses. There are now four special types:

- Horn clauses, where n=1 or n=0;
- denials, where n=0.
- facts, where n=1 and m=0;
- definite clauses, where n=1;

The facts and definite clauses are most often employed in logical and relational learning. Horn clauses include facts and definite clauses. Denials represent negative information and be used as queries and goals [16], such as $\leftarrow \text{country(bristol)}$. Specifies that Bristol is not a country, the clause country(bristol) is false. Typically, the databases are consisted by a set of clauses. All of the clauses in the set are true.

2.1.3 The Semantics of Clausal Logic

Finally, the semantics of clausal logic can be introduced after the syntax of clausal logic has been defined.

The semantics of clausal logic are based on the concept of an interpretation *I* and is usually restricted to the class of so-called Herbrand models, which are especially amenable to symbolic manipulation [17]. Interpretations are defined by using notion of domain. Herbrand interpretations is a special class of interpretation that will be focused on this report. Therefore, the Herbrand domain represents a set of all ground terms, which is constructed with constants and function symbol. A Herbrand interpretation then maps all ground terms to itself [16]. For a program $\mathcal{P}$ with language $\mathcal{L}$, the Herbrand universe $H_{\mathcal{L}}$ is the set of all ground terms in $\mathcal{L}$, and the Herbrand base $B_{\mathcal{L}}$ is the set of all ground atoms in $\mathcal{L}$. A Herbrand interpretation is an interpretation that is any subset of $B_{\mathcal{L}}$. An interpretation *I* satisfies an expression *E* if and only if:

- E is a positive ground literal "a" such that $a \in I$
- E is a negative ground literal "not a" such that $a \notin I$
- E is a ground rule *R* such that *I* satisfies a head atom of *R* or fails to satisfy at least one body literal of *R*.
- E is a non-ground rule *R* such that *I* satisfies each ground instance of *R*

- E is a program *P* such that *I* satisfies every clause in *P*.

Some theories have multiple models. There might be a model can be a subset of another model. If there is no subset of the Herbrand model, the model is minimal. The minimal modal is unique and is called the least Herbrand Model (LHM). [16]

2.1.4 Logic Programming

In the previous section, we have seen what clausal logic is and how the logic represents simple knowledge about a particular domain. Logic programming is a particular way to achieve programming, which is the use of logic as both a declarative and procedural representation language [12, 13]. In a purely declarative programming language, it would not involve in expressing procedural knowledge. Therefore, logic programming cannot ignore procedural programming. Prolog is not a purely declarative language, which is also involved procedural language [5]. Prolog is one of the first logic programming languages.

Logic programming is based on the fact that a backwards reasoning theorem proves (as mentioned in 2.1.1) applied to declarative sentences in the form of implications like:

$$\text{if } A_1 \text{ and } \cdots \text{ and } A_n \text{ then H}$$

The sentence means if $A_1 \cdots A_n$ are true, then H becomes true, or false otherwise. If we use prolog to deal with the sentences above, the form of the language is:

$$H: -A_1, \cdots, A_n$$

There are four types of clauses we have mentioned in 2.1.2 with expression of the form $h_1; \cdots; h_n \leftarrow b_1, \cdots, b_m$, which are Horn clauses, denials, facts and definite clauses. We now demonstrate each type clauses in prolog based on the Example 2.1.1:

- Horn clauses, where n=1 or n=0:
  These two clauses would indicate rules (n=1) and query (n=0). Here, we discuss the Horn clauses where $m \neq 0$.
  For instance, $\text{cites}(A, B) \leftarrow \text{authorOf}(A, P), \text{reference}(P, Q), \text{authorOf}(B, Q)$ is a rule, which can be represented by $\text{cites}(A, B): -\text{authorOf}(A, P), \text{reference}(P, Q), \text{authorOf}(B, Q)$. When n=0, the clause $\leftarrow \text{authorOf}(A, P), \text{reference}(P, Q), \text{authorOf}(B, Q)$ would be a query, which can also represented by $?-\text{authorOf}(A, P), \text{reference}(P, Q), \text{authorOf}(B, Q)$.
- denials, where n=0:
  This is included by Horn clauses. Usually, denials represent negative information in the database. $:-\text{authorOf}(\text{russell}, \text{logic\_for\_learning})$. is the representation of a negative information, because the author of logic_for_learning is not Russell.
- facts, where n=1 and m=0:
  For instance, $\text{authorOf}(\text{lloyd}, \text{logic\_for\_learning}) \leftarrow$ is a fact in the database, which can be represented by prolog as $\text{authorOf}(\text{lloyd}, \text{logic\_for\_learning})$.
- definite clauses, where n=1:
  If $m \neq 0$, definite clauses are rules. If $m = 0$, they are facts.

Therefore, Progol can be used to represent any kind of clauses, and also can be used a declarative language. In the section 2.3, we can learn that the logic programming can be used in any ILP systems or workbench, such as Progol, Tilde, Warmr and Prism. All of the information can be read as prolog language.

2.2 Machine Learning Representations

As introduced in the Chapter 1, there are five type representations for machine learning, BLL, AVL, MIL, MRL and ILP. In this project, there are only two machine Learning representations have been used in the workbench. Therefore, in the following 2 sections, only AVL and ILP are described.

2.2.1 Attribute Value Learning

AV representation is the most popular class of representation languages for data analysis so far and is employed in many academic and business areas. The reason why it is very popular is that the data can be represented in tabular form. It is very convenient to analyze or transfer data.

Consider a playtennis example from [14] is shown in Table 1 (partial):

Table 1 Quinlan's playtennis example

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	no	negative
sunny	hot	high	yes	negative
overcast	hot	high	no	positive
rain	mild	high	no	positive
rain	cool	normal	no	positive
rain	cool	normal	yes	negative
overcast	cool	normal	yes	positive
…	…	…	…	…

AVL has a similar character with BLL. We just change the boolean value into attribute value. Such as $\mathrm{class}(\mathrm{positive}) \leftarrow \mathrm{playtennis}(\mathrm{overcast}, \mathrm{Temperature}, \mathrm{Humidity}, \mathrm{Class})$.

These two representations are essentially propositional, which are based on Boolean or propositional logic. Propositional representations cannot represent multiple entities as well as the relationships amongst them. Ryszard Michalski and Gordon Plotkin soon realized the limitations and started to utilize more expressive knowledge representation frameworks for learning. They focused on representing a variable number of entities using frameworks, which have relationships amongst them. Such representations are called relational. They are grounded in and derived from first-order logic, logical representations are found soon [16]. The targets of it are to learn problem involving multiple entities and the relationships amongst them. Then the rest of representations that are logic representations will be illustrated.

2.2.2 Inductive Logic Programming

ILP is the intersection of machine learning and logic programming. The examples, background knowledge and descriptions of ILP systems are all described within logic programs. The goal of the ILP is to find the hypothesis, which is covered all examples except negative examples, with background knowledge and concept description language. [18]

This section has covered a fraction of prolog. Prolog is a powerful representation language. Now, there

is a more complex example shown below:

Example 2.2.1. Consider the facts $\text{sort}([1,3,4],[1,3,4]) \leftarrow, \text{sort}([3,5,4],[3,4,5]) \leftarrow$ and $\text{sort}([5,4],[4,5]) \leftarrow$. When learning them as positive example from entailment setting [16]. The program below can cover all of these examples:

```
insertsort(List, Sorted):-isort(List,[],Sorted).
isort([], Acc, Acc).
isort([H|T], Acc, Sorted):-insert(H,  Acc, NAcc),isort(T, NAcc, Sorted)

insert(X,[],[X]).
insert(X,[Y|T],[Y|NT]):-X>Y, insert(X,T,NT).
insert(X,[Y|T],[X,Y|T]):-X=<Y.
```

These prolog codes are based on [16]. As we can see from the codes, the positive examples are covered by the programs. The negative example would state a wrong output for the given input.

Storing prolog into database, many researchers did a lot of work. However, there is still no suitable approach to store them. In this project, finding a way to store such programs is needed, one way can do it now is to store the whole program in one table with the name of the program's function.

2.3 Learning Systems

In the section, four famous systems of structured data will be demonstrated. We will focus on the input and output data format of each system. For studying these systems, further knowledge about how the different machine learning representations work can be learned.

2.3.1 Progol

Progol is a state-of-the-art ILP system, which was first described in [10]. Since then a number of advances have been made, the last system has been informed by feedback from experiments on a variety of real-world applications [11]. There is no graphical interface in the software, but the form of this progol is easy to be understood. Progol constructs logic programs from examples and background knowledge. In addition, examples are divided into positive examples and negative examples. All information is stored in a single file by the expanded-name 'pl', which also include settings and mode declarations. In the project, expanded name always be used to indicate files, like .pl.

Input file

For instance, input file of animals can be given as following [11]:
We just show a part of the data for saving space.
Background knowledge of animals can be given as following:

```
has_covering (dog, hair).
has_legs (dog,4).
has_milk (dog).
Homeothermic (dog).
```

Habitat (dog, land).

and positive examples can be given like:

class (dog, mammal).
class (trout, fish).
class (eagle, bird).
class (lizard, reptile).

and then there are several negative examples given:

:- class (penguin, mammal).
:- class (dog, fish).
:- class (penguin, reptile).
:- class (dog, bird).

Furthermore, the categories of objects, such as numbers, lists, names etc., should be described by types in the world under consideration.

animal (dog).
class(mammal).
class(bird).
covering(scales).
covering(feathers).
habitat(land).

After that, the relations between objects of given types are described by modes. They can be used either in head (modeh) or body (modeb) of hypothesized clauses. For the head of a general rule, we might give the following head mode declarations

:- modeh(1,class(+animal, #class))?

The declaration states that head atom has predicate symbol class and has two variables of type animal and class as arguments. The '+' sign indicates that an argument is input variable. The '-' sign indicates output variable. And '#' sign indicates that a constant should be sited in the hypothesis. Therefore, the declaration above indicates

Class(X, mammal).

For atoms in the body, modeb can be declared as following:

:- modeb(1,has_milk(+animal))?
:- modeb(1,has_covering(+animal, #covering))?
:- modeb(1,not has_milk(+animal))?
:- modeb(*,habitat(+animal, #habitat))?

The number in the modes is called the recall. This is used to bind the number of optional solutions for instantiating the atom. '1' means the predicate gives a unique answer (yes or no) when given suitable arguments. '*' indicates no limit to the number of instantiations, but progol will set its upper limit of 100.

Finally, we need describe some parameters as setting. For instance,

set(c,4)?
set(posonly)?

The first declaration means the maximal length of clause. And another one indicates learn from positive examples only.

Output file

The output file include final clauses and intermediate steps which enclosed with '[' and ']'. Moreover, there are four numbers which are preceded in each clause. These are, in turn,

f = Number of positive examples covered -
Number of negative examples covered -
Number of literals in body of clause -
Optimistic estimate of literals needed;
p = Number of positive examples covered;
n = Number of negative examples covered;
h = Optimistic estimate of literals needed.

The progol will find the one of the most specific clause depends on the highest f with lowest n. Finally, at the bottom of the output file the final output are listed as:

```
class(snake, reptile).
class(A, mammal) :- has_milk(A).
class(A, fish) :- has_gills(A).
class(A, bird) :- has_covering(A, feathers).
class(A, reptile) :- has_covering(A, scales), has_legs(A,4).
```

These are the final hypothesis. The progol codes and the output information are based on [11].

Progol has a simple way to collect data. However, many other ILP systems choose to use three to store different information separately. Aleph is another ILP system, which is much similar with progol, and also has three files. The study between them will help us to improve the converting between one file systems and three files systems. Studying Aleph will start at the beginning of the project.

2.3.2 ACE (Tilde and Warmr)

Tilde is one of the key algorithms in the ACE Dataming. There are three different file types in ACE system as input files, which are *app.kb*, *app.bg* and *app.s* where *app* is the name of the application [2]. The usage of each file shown below:

- *app.kb*: this file contains the examples. Both training and test data are included in this file.
- *app.bg:* this file contains the background knowledge, which is an optional file.
- *app.s*: this file is a setting file, which include certain parameters of the algorithms.

We also can see the Figure 2. It has two format of example, key format and model format, key format is much similar with progol. If we use model format, we have to set model for each submodels. Background knowledge is much similar with progol. However, in the setting file, Tilde and progol have a big different.

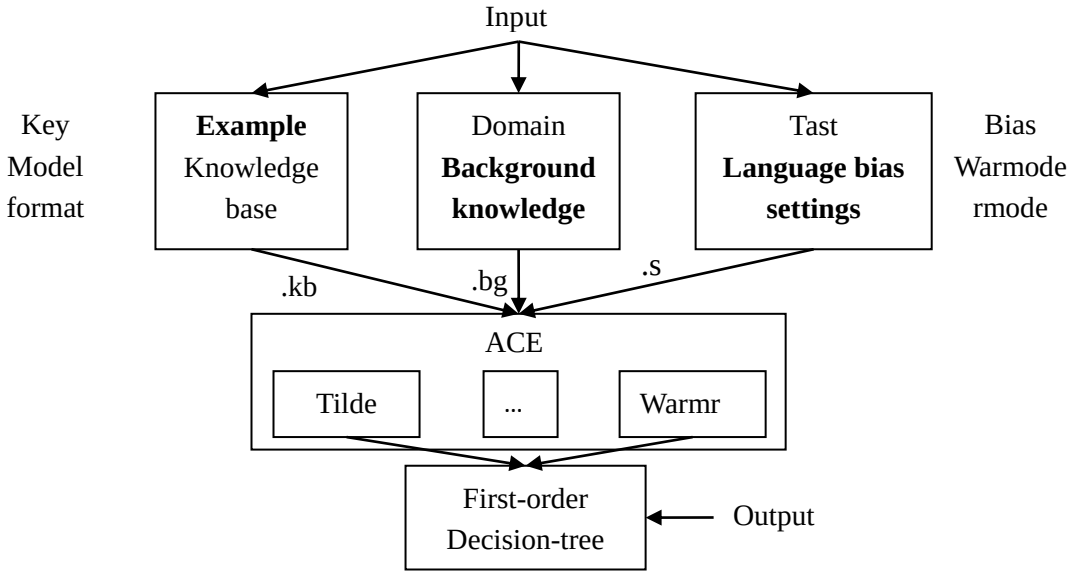
In Tilde setting file, the refinement operator (hypothesis language) is using *warmode* as following form [2]:

$$\text{warmod}(\text{pred}(t_{m_1}, t_{m_2}, \ldots t_{m_n})$$

there is other form like:

$$\text{warmode}(\text{N}: \text{pred}(t_{m_1}, t_{m_2}, \ldots t_{m_n})$$

where, N is the times of literal can be added, and N is infinite by default [2]. The t_{m_i} are type and mode declarations for variables.



http://www.cs.bris.ac.uk/Teaching/Resources/COMSM0301/lsd-10-11-lecs67-ace.pdf

Figure 2 ACE system

At lower level, the refinement operator is using *rmode* predicate as:

rmode(conj).

It also has other form like:

rmode(N: conj).

where N is a natural number or infinite. conj means that the refinement step can consist of adding the conjunction of literals to the query. rmode also can contain some constant and symbols [6]. The setting file of Tilde input is quit complex. We do not

The output files include '.out' and '.ptree' and some other files. 'bongard.ptree' is shown here as a typical output of first-order decision-tree as Figure 3.

bongard.ptree

```
1 triangle(-A) ?
2 +--yes: in(A,-B) ?
3 |       +--yes: triangle(B) ?
4 |       |       +--yes: [pos] 82.0 [[pos:82.0,neg:0.0]]
5 |       |       +--no:  circle(-C) ?
6 |       |               +--yes: in(C,-D) ?
7 |       |               |       +--yes: [neg] 28.0 [[pos:0.0,neg:28.0]]
8 |       |               |       +--no:  [pos] 34.0 [[pos:34.0,neg:0.0]]
9 |       |               +--no:  [neg] 36.0 [[pos:0.0,neg:36.0]]
10 |      +--no:  circle(-E) ?
11 |              +--yes: in(E,-F) ?
12 |              |       +--yes: [neg] 79.0 [[pos:0.0,neg:79.0]]
13 |              |       +--no:  [pos] 12.0 [[pos:12.0,neg:0.0]]
14 |              +--no:  [neg] 34.0 [[pos:0.0,neg:34.0]]
15 +--no:  [neg] 87.0 [[pos:0.0,neg:87.0]]
16
```

Figure 3 Tilde output file - bongard.ptree

From Figure 3, the decision tree is clearly built based on 'triangle', 'in' and 'circle', which are required in the rmode, such as:

mode(triangle(+-S)).

rmode(square(+-S)).

rmode(circle(+-S)).
rmode(in(+S1,+-S2)).

Tilde induces first-order logical decision tree (FOLDT) from data [3] and the input and output files have been introduced above. It will help us to gain general information about the characters of ACE Data mining for studying other systems.

2.3.3 Prism

Programming in Statistical Modeling (Prism) is a logic-based language like prolog, which integrates logic programming and probabilistic reasoning with parameter learning [19]. Both of the Prism and prolog execute programs in a top-down left-to-right manner. The input of the file is the rules and facts. When people visit prism, the terminal will be a control panel that uses can input what they want to predict of calculate the possibility of a set data. After that the result will be listed in the terminal such as the result in Progol. In addition, the Prism offers two other command, call upprism and mpprism. “upprism” is named as uni-process Prism which is used as a batch execution, that means there is no interactive execution when the prism run the data files. Moreover, if users want to run the upprism instead of prism, the query for the batch execution must be specified in the body of prism_name/0-1. “mpprism” is named as multi-process Prism which need to be set some additional settings for a parallel computing environment [19]. The output file would depends on what users want to predict or calculate. One of the result, which is used the example file in Prism software, is list in the Chapter 5.

2.3.4 Weka

Weka is the most popular workbench for AV learning as mentioned in the introduction. It provides a comprehensive collection of machine learning algorithms and data preprocessing tools [9]. Weka can be a previous work of my project. However, only data format of Weka is studied in this section. For example, a typical file is named “weather.arff” can be seen in Figure 4.

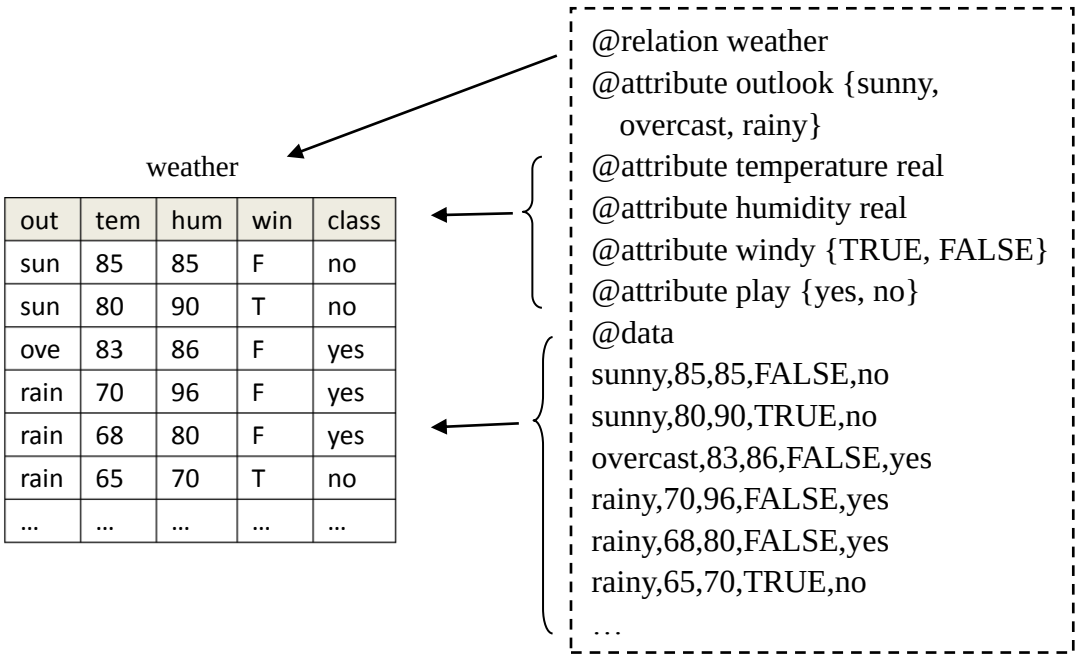


Figure 4 weather.arff

Figure 4 shows that all ARFF format files can be converted into tables. The data can be easily managed by databases. ARFF is the data format of Weka. Any ARFF format file can be passed on the Weka.

Information in the ARFF file can be divided into three parts: declaring relation, declaring attribute and listing data. AVL can be shown in a single table. ‘Relation’ is the name of the table. ‘Attribute’ is the meaning of each column. ‘data’ store the whole values.

This kind file format can be used in the workbench and allowed to use every function in the workbench, like regression, classification, clustering, association rule mining and attribute selection. However, some of the data attributes need to be amend slightly to suit specific algorithms. The output of Weka is comprehensive for AV learning. Different outputs are based on what parameters you set and what kind of data you analyze.

2.3.5 1BC

1BC is a first-order Bayesian Classifier. A Bayesian Classifier is simple probabilistic classifier that can be shown as:

$$argmax_c \frac{P(d|c)P(c)}{P(d)} \quad or \quad argmax_c P(d|c)P(c) \qquad [8]$$

However, the first-order rules not be used in the 1BC, it generate a set of first-order conditions instead of those rules, which are used classical attributes in a naïve Bayesian classifier. It is seem like a propositionalization approach [8]. In the rest of the section, the input file and output file are introduced.

Input File

There are mainly two input files, call .prd and .fct, to store the hypotheses language in first one and store all ground facts only in another. For instance the eastwest.prd [6] file:

```
--INDIVIDUAL
train 1 train cwa
--STRUCTURAL
car 2 1:train *:car cwa li
load 2 1:car 1:load cwa li
--PROPERTIES
class 2 train #class * cwa
shape 2 car #shape 1 cwa
length 2 car #length cwa
wall 2 car #wall cwa
roof 2 car #kind cwa
wheels 2 car #nb_wheels cwa
object 2 load #object cwa
number 2 load #number cwa
```

There are three parts in the file, “--INDIVIDUAL” part as a molecule that can be a kind of objects. “--STRUCTURAL” are used to refer to relate objects as setting new variables, “--PROPERTIES” define the predicts as consuming variables. Each line corresponds to one predicate that is defined by name, arity, type of arguments and whether the closed world assumption (cwa) is used for the predicate. It there is no cwa, the negative ground facts of that predict must be given in the .fct file. The line under the “--PROPERTIES”, “*” is inserted into, which mean the number of repetitions of the predicate in the hypothesis would be running. [6]

Output File

The output file is call .res. The results are stored into this file. More information can be stored here if the other file called .sta is used with the verbose option –v is set. [6]

2.4 Conclusion

In this chapter, the logic foundation has been demonstrated firstly, which include clausal logic, logic programming. Then two machine learning representations are illustrated in details. Finally, five ILP and AV data mining system have been explained based on the input and output files. The knowledge is very important. The workbench in the project will integrate the five systems that are mainly used ILP and AV Learning. Learning the knowledge in this chapter is helpful to understand what these systems need (as input) and what kind results we will have (as output), which also can help us to make a general structure of how this workbench works? In the next Chapter, some famous previous works on the KD workbench are introduced.

3 PREVIOUS WORKS ON KD WORKBENCHES

Some previous works on KD workbench and their limitations are described in this chapter, which involve Weka, ACE Datamining, Orange Data Mining, Oracle Data Mining and BET. These works are references of the project to create a new workbench for KD, which will inherit most advantages of these previous works, and avoid the limitations of them.

3.1 WEKA

Weka is one of the most famous workbenches for AV learning, which is a collection of state-of-the-art machine learning algorithms and data preprocessing tools. Weka can be used easily to access to the techniques in machine learning by researchers without becoming familiar with the technical details of the software and algorithms. Weka is enormously popular, because of its powerful, and is recognized as a landmark system in DM and machine learning [9].

There are several advantages of having a workbench. Firstly, using workbench can unify the representation. One reason for Weka is powerful is the simplicity of its file format. Any AV data can be encoded in the ARFF file format, and then passed on any available functions, such as classifiers and clusters. Secondly, this workbench supplies a visual interface, which can provide an intuitive way to execute different functions of the workbench rather than using command line interface. Finally, Weka combines a lot of AV algorithms. Users just need to click the mouse, and amend the parameters, and then the result will be display in the panel. In this project, we will consider assembling as many algorithms as well for supporting a better workbench.

Weka is powerful in AV learning, even in BL. However, a lot of machine learning problems are based on other kind representations, like RL and ILP. The most important weakness of Weka is that only AV data can be used in this workbench. The new workbench will be focus on integrating more algorithms for any kind of representations to fix the limitation of Weka in this area.

3.2 ACE Data Mining

ACE (A Combined Engine) Datamining is a relational datamining system developed at the Katholieke Universiteit Leuven, which bring several relational algorithms together, such as Tilde, Warmr, ICL, relational reinforcement learning, and some incremental learning systems. The data format in ACE is principally based on Prolog. This is a famous workbench for ILP.

ACE is a data mining system that provides a command interface to a number of relational data mining algorithms. Relational data mining is the process of finding possible patterns in a relational database, which consist multiple tables [2]. Currently, ACE encompasses nine algorithms, which are Tilde [15], Warmr [1], ICL, RegRules, KBR, NLP, RIB3, TG and RRL [4]. These systems in the ACE are described in [2]. An advantage of the workbench is that it uses three files to contain three different data, examples, background knowledge and settings. The methods will be considered to use in the new workbench. In addition, this workbench has a number of relational data mining algorithms. We can use any machine learning representations in this workbench.

Nevertheless, as we known, ACE only has a command interface. Users need to learn the operation of command, which is not very easy to be used for beginners. Therefore, a better interface and an easier operation should be needed for analyzing relational data mining. In addition, ACE cannot connect database, users need to create datasets manually from their database, and then run the data with ACE. This is very inconvenient. The new workbench should have a function of connecting database and can converting the data into specific datasets automatically.

3.3 Oracle Data Mining

Oracle Data Mining (ODM) is a priced option to the Oracle Database 11g Enterprise Edition. The workbench runs directly in the oracle database. ODM has been design for fitting well with SQL language. It leaves data in its natural form instead of transposing and transactional form to a flat, and mines data directly. Such approach is simplicity and improved performance.

ODM also has a friendly interface for users. However, when we consider analyzing some relational data mining, some data would be transacted or transposed. For example, there are several rules in the ILP. ODM cannot parse them and translate into database. The new workbench would have this ability.

3.4 Orange Data Mining

Orange is a component-based data mining and machine learning software suite, has a friendly interface and powerful visual programming front-end for explorative data analysis and visualization. It includes comprehensive set of components for data preprocessing, modeling and exploration techniques. It is maintained and developed at the Bioinformatics Laboratory of the Faculty of Computer and Information Science, University of Ljubljana, Slovenia. [20]

There are five main features in orange data mining.

- Visual programming
- Visualization
- Interaction and data analytics
- Large toolbox
- Scripting interface

Orange data mining is good data mining software. Since there is limited time to study, it is hard to find a way to integrate this workbench in the project.

3.5 BET System

BET is implemented in Java an Object Oriented Language that is easy to extend. Many other applications can use the ILP system's API and do not need to worry about the implementation details. They are developing a successful emulation for all the existing systems (FOIL, GOLEM, PROGOL, etc) and users can use interface to set the parameters of each system. By the integration of many different systems, users can easily compare the systems' performs and find a better results. In addition, varied list of operators can search the search space. [18]

This workbench has the main idea of this project. We would also like to create such workbench for different ILP systems. Whereas, we will also consider integrate Weka or other AV learning and convert all representations into AV format. Moreover, database will be utilized in our workbench. In addition, BET is software for ILP systems. We do not think it is a convenient method for users to use. A new structure of the workbench will be attempted based on Brower/Server Structure.

3.6 Conclusion

From the studying of the previous works, the advantages and disadvantages are clearly shown. The new workbench should minimize the limitations and maximize the advantages of the existing works. One of the possible methods is using website to preprocess and run data, which have been accomplished in this project. In the next Chapter, the methodology of the project is discussed.

4 METHODOLOGY

After studying the background and previous works on this field, the strong background knowledge is formed on what are the AV and ILP learning, how do the previous workbenches work? This chapter describes the developing language, needs analysis and workbench design based on the background knowledge.

4.1 Developing Languages

Lots of developing languages can be chosen to develop the workbench as a website. JSP is finally used in the project. Java is one of the post popular and powerful languages to develop J2EE and provide a wide and convenient interface with Linux system and WEKA. In addition, HTML is the main website developing languages with CSS and JavaScript. How good and friendly the workbench interface is depends on the manipulation of the HTML+CSS+JavaScript. Java is the main developing language that is used to convert data, calling the systems and display the information dynamically. Compare with other developing languages, Java has portability and security, and many open sources can be used to develop the workbench, like SmartUpload methods -- one of the most famous methods to upload, delete and download files. In this project, shell languages are also used to handle the batch for running the different systems. Partial of these codes are listed in the Appendix.

4.2 Needs analysis

Needs analysis is one of the most important processes when new software or workbenches will be developed. A good need analysis can directly affect the quality of the following developing parts. In general, the needs of the workbench are always changed. A very details analysis cannot be confirmed. There are, however, some essential parts of the workbench like Inputs, middle results and outputs need to be described. In the section, these three parts and files management are represented.

4.2.1 Common File

Creating a common file is one of the project aims. It is used to collection the information from different system data formats and also can sharing the data with the different systems. This file contains a new structure of data that any ILP learning systems' file has not been used before. The detail design of the common file is explained in the section 4.3.1.

4.2.2 Inputs

Input files are required at the beginning of the operations in the workbench. However, no uploaded files are also allowed to come into the next operation control panel, since users can require a brand new data file to analyzing data like Figure 5 shown below. The diamond indicates the condition whether the new empty file should be created. The Middle results are described in the next section:

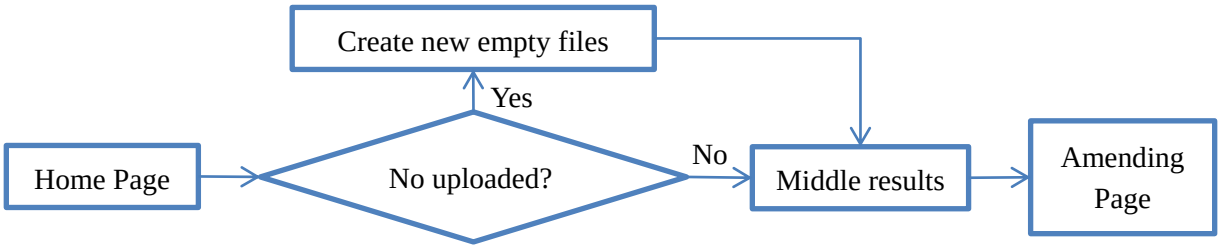


Figure 5 Input File Flow Chart

4.2.3 Middle Result

There are several middle results involved when the users in the amending Pages as Figure 6 shown. A common file will be converted from the uploaded file(s). A number of other systems' files will then be converted from common file. At the same time, some of results are created by calling their systems, like Progol and ACE for giving a prompts to users to help them amend the data efficiently, the other systems are also provided some prompts by the workbench based on their minimum requirements. After user update their data file, a new loop from converting common file to generating the systems' files starts again.

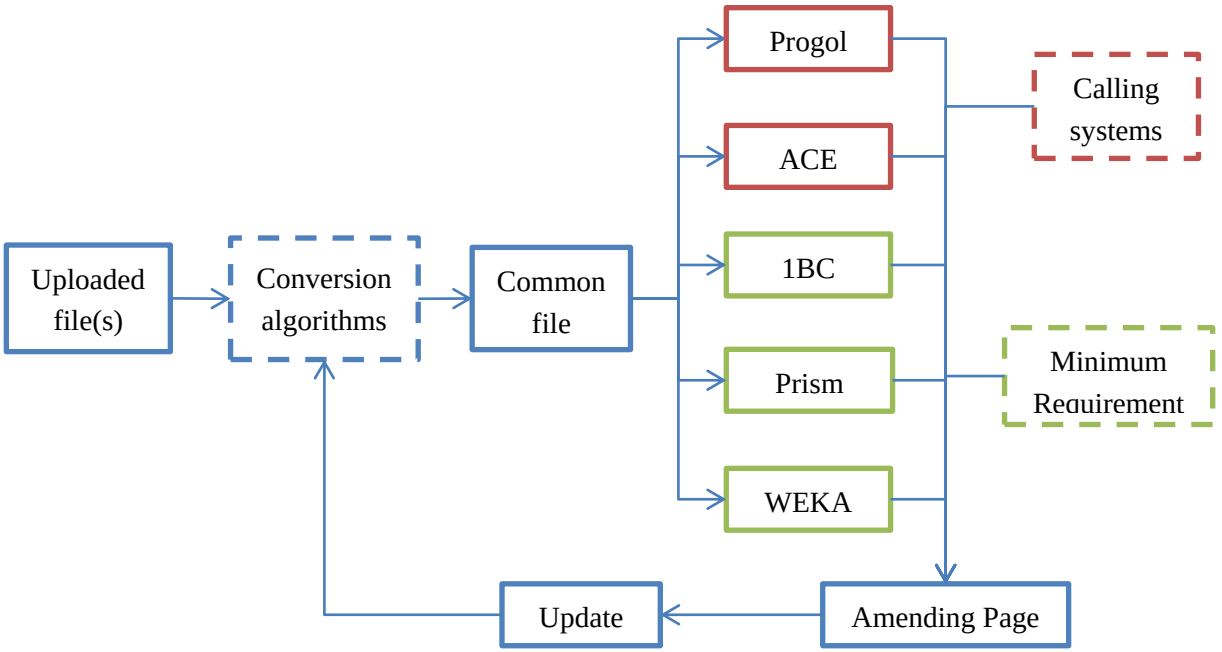


Figure 6 Middle Result Flow Chart

4.2.4 Outputs

There are three types of files need to be shown at the end of the workbench, which are the results, amended file and common file, as Figure 7 shown. These outputs need to be displayed in the Result Page, downloading files is required. Furthermore, as mentioned in Chapter 2, the results of ACE are store in the different files. If ACE cannot be run correctly, there is no results file that can be generated. Therefore, the outputs of the ACE has to be considered showing the suitable one based on which output can show more useful information to users.

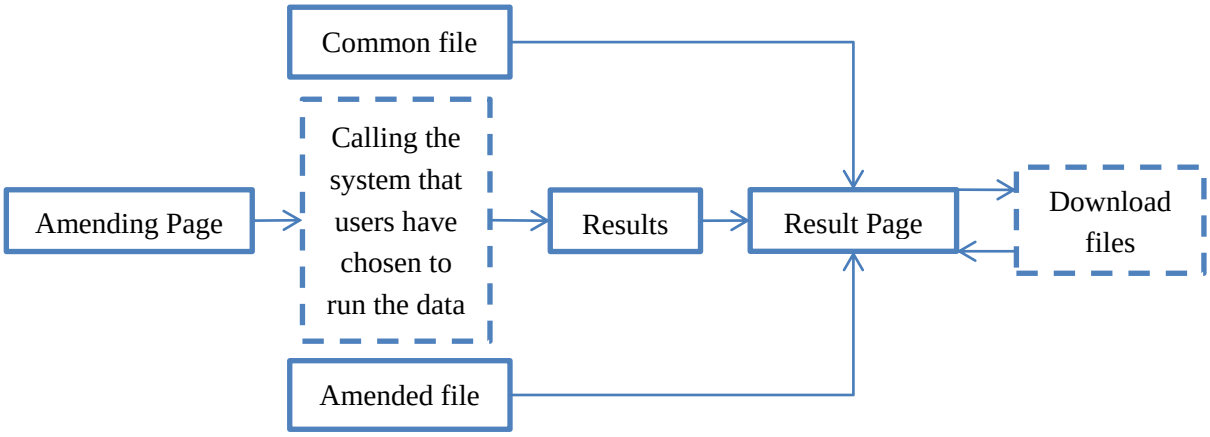


Figure 7 Output Files Flow Chart

4.2.5 Files Management

Five systems have been integrated in the workbench, good organization of files' directory is helpful to expand and maintain the workbench. In this project, entirely separate file system management is used to manage the files instead of storing the relative files into the same subdirectory. For example, there are three subdirectories that are used to store progol, called "progol", "tempprogol" and "resultprogol". The "progol" directory is used to store the final progol file which is converted from common file. The "tempprogol" directory is used to store the semi-manufactured progol file, which is still need to be clean up. The results of progol will store into the "resultprogol" directory. This approach might create more files when users operate the workbench. However, the needed time is acceptable. In addition, that could bring lots convenience for developing the workbench.

4.3 Workbench Design

Some of the flow charts have been described in section 4.2 for clearly explaining the requirements. In this section, common file format and the main flow chart of the workbench are illustrated. The figures used in 4.2 are needed here instead of the duplication of works.

4.3.1 Common File Format

Creating new data format is the core of this project. The data format would allow us to analyze the data with different algorithms. There are two available approaches to achieve this. One is Common Data Format (CDF); another one is Converting Data between Different Algorithms (CDDA), which are mentioned in the interim report. The CDF finally have been used in this project as shown in Figure 8:

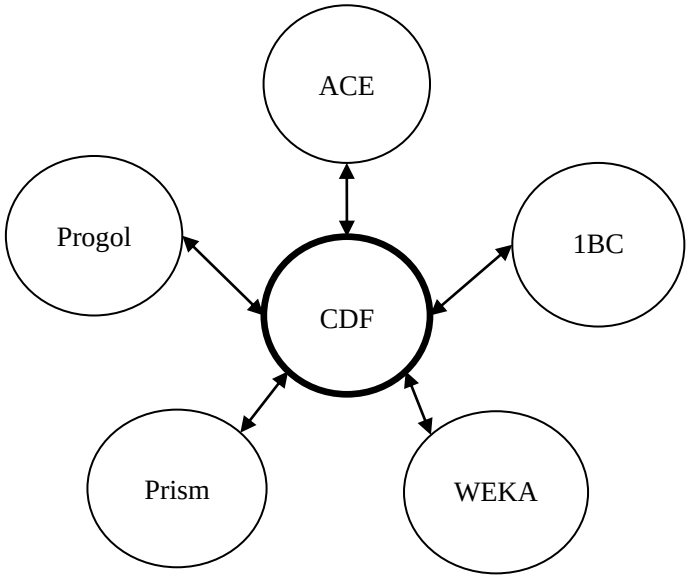


Figure 8 CDF

Based on the knowledge in section 2.3, a general character can be found for these ILP systems' learning. All of them have four main parts, which are settings, positive examples, negative examples, and background knowledge. In the setting part, it is always divided into mode declarations and settings. Therefore, as default, in any ILP system there are five elements, which are settings, mode declarations, positive and negative example and background knowledge. Some of the ILP systems just have two or three elements, but on one contains the element that is not included in the five.

As the train example shown in Chapter 2, when we want to know which way the trains more likely to go? Some positive and negative examples and background knowledge should be needed firstly. Some particular mode declarations and settings should be set for prediction. This is prevalent method of realize the datasets. Therefore, in the common file, abstracting the information separately is a possible approach to accomplish the aim. However, the common cannot just store the data based on the different parts. It should have their own structure to store the data to realize the data sharing between the different data formats. Actually, the positive examples and background knowledge can be shared between different systems directly. The part that can be shared directly need to be sorted and parsed before they can be stored in the common file.

In the interim report, we supposed that the common file(s) can have more than one files like ACE and 1BC. The settings and mode declarations can be stored and shared in one file, the facts and rules in another file. However, for convenient, the final common file have been created in a single file. The method used here to distinguish the data is to input a start symbol in front of each sentence such as:

$$\mathrm{from}(Parameter):-$$

This sentence looks like a head of a rule. The idea of the structure is from the prolog language. Even though Java is used here to parse the data, prolog actually can do that faster than Java does. This structure allows people to make an extension or improvement on common file with prolog language. In addition, this method is successful used in the workbench for storing data. Furthermore, having a good start of a sentence is not enough. To realize the data sharing, the rest of the sentence must be powerful. There is a list as following, which are the mode declarations in Progol, ACE and 1BC settings file:

$$:-\text{modeb}(1,\text{class}(+\text{train},\#\text{direction}))?$$
$$\text{warmode}(1:\text{class}(+\text{train},-\text{direction})).$$
$$\text{class 2 train \#class 1 cwa}$$

The meanings of each sentence have been explained in Chapter 2. Here, they all predict one thing – which directions the trains are going to leave? The first two lines are head mode declarations. In 1BC, different structures of the predictions are used. However, we can clearly see the similar part from the three sentences. Therefore, the following sentences have been created for store all these information:

- from(Progol):-body_name(modeb),body_int(1),not(no),arity($),body_predict(class),body_predict_name(+train,#direction),repetition($),cwa($).
- from(ACE):-body_name(warmode),body_int(1),not(no),arity($),body_predict(class),body_predict_name(+train,#direction),repetition($),cwa($).
- from(PROPERTIES):-body_name($),body_int($),not(no),arity(2),body_predict(class),body_predict_name(train,#class),repetition(1),cwa(yes).

As we can see, the three sentences have same structure. When we have any one sentence, the other two can be generated automatically by workbench. The process of the sentence conversion is illustrated in Chapter 5. The setting part (indicates the parameters in the systems) cannot be shared, because the type of algorithms is different, the meaning of the settings are also different. However, the rest data in the ILP learning can be shared by each other. For example:

- from(Progol):-rule{number_of_cars(T,N):- train(T), findall (M, has_car(T,M),Ms), length(Ms, N), integer (N). }.
- from(Progol):-fact{class(east1,eastbound).}.
- from(Progol):-negative{:-class(west5,eastbound).}.
- from(BC):-fact{class(t2,eastbound).}.
- from(ACE):-fact{begin(model(02)).}
- from(ACE):-fact{class(east1,eastbound).}.
- from(Dat):-fact{phenotype(ab).}.

These data can be used by any ILP system, except some specific clause, like from(ACE):-fact{begin (model(02)).} is only used by ACE Model Format. Alternatively, this sentence can be stored with other name instead of *fact*. In this project, the above styles have been used in common file. They are easy to realize all the functions in the workbench. Furthermore, Weka is also integrated in this workbench. The common style of the data is like:

from(Weka):-weka_name(eastwest).
from(Weka):-weka_attribute(short),attribute_value({yes, no}).
from(Weka):-weka_data_start(@data).
from(Weka):-weka_data(yes,eastbound).

From this example, we can consider that the common can be extended for any other systems' file formats with just a separate parameter and name as "*from(Parameter):-name(value).*", and store the data which can be shared into the shared part of the common file.

4.3.2 Main Flow Chart of Workbench

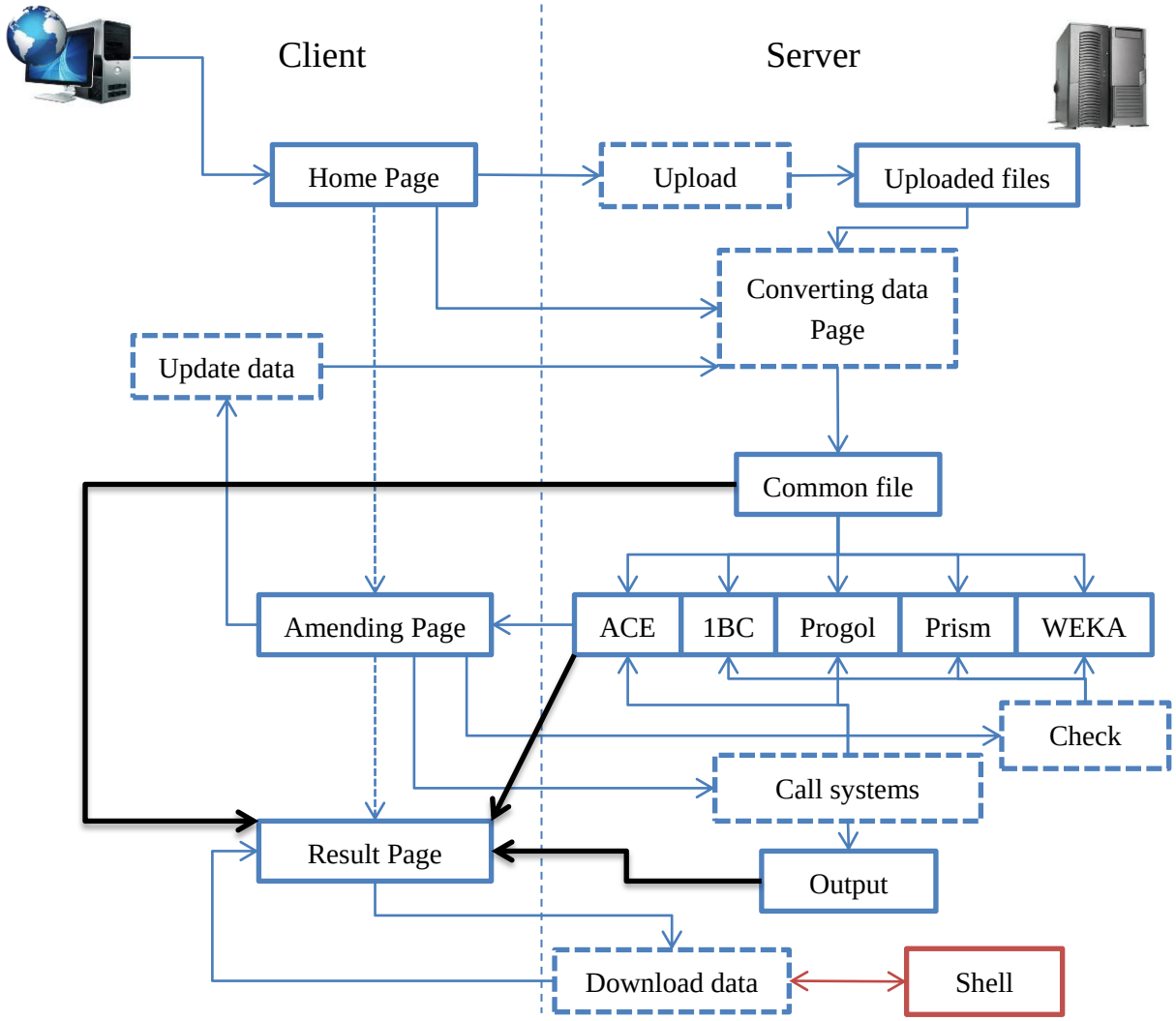


Figure 9 Main Flow Chart of Workbench

Figure 9 shows the main flow chart of this workbench, which is used Browser/Server (B/S) structure. The users can use browser to interview the workbench. The blue dashed box means the java methods are required, the blue solid box in Client part indicate the web pages, and shows the types of files in server part. The dashed line means the users could only visit the three web pages like they are connected. The solid line, however, shows the real path that the data followed.

First of all, the Home Page is displayed, the files uploaded by users is stored in the "Uploaded files" through the Upload methods. Secondly, the server will convert the data into common file and then convert the data into all of the systems. Thirdly, these data will be displayed in the Amending Page. At the same time, Call Systems methods are used to calculate the ACE and Progol results. Then the error results will be displayed as prompts. The rest systems need the input from users. Therefore, the prompts about error or warning prompts will be found by the Check methods. Fourthly, users can update their data, and then the same flow from second will be run again. Fifthly, when users have finish update data, they can choose an algorithm to see the result. At this time, the particular Call systems methods will be run, and send the outputs to the Result Page that include result, amended file and the common file as shown in thick line. Finally, users can download any files using download function. If there are several

results, a shell will be employed to compress these files in one .zip file.

4.3.3 Upload and Download File Flow Chart

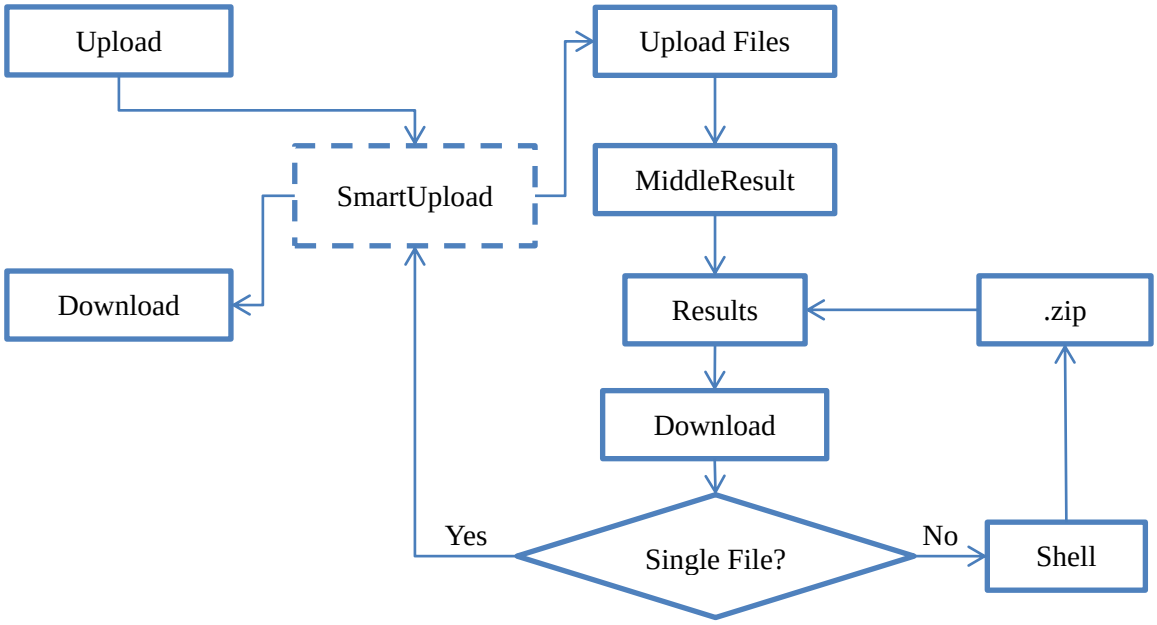


Figure 10 Upload and Download File Flow Chart

Figure 10 shows the details of upload and download in the workbench. SmartUpload methods are used here to upload and download files. (SmartUpload is an open free code). The explanation has been made in the 4.3.2.

4.3.4 Converting Data Flow Chart

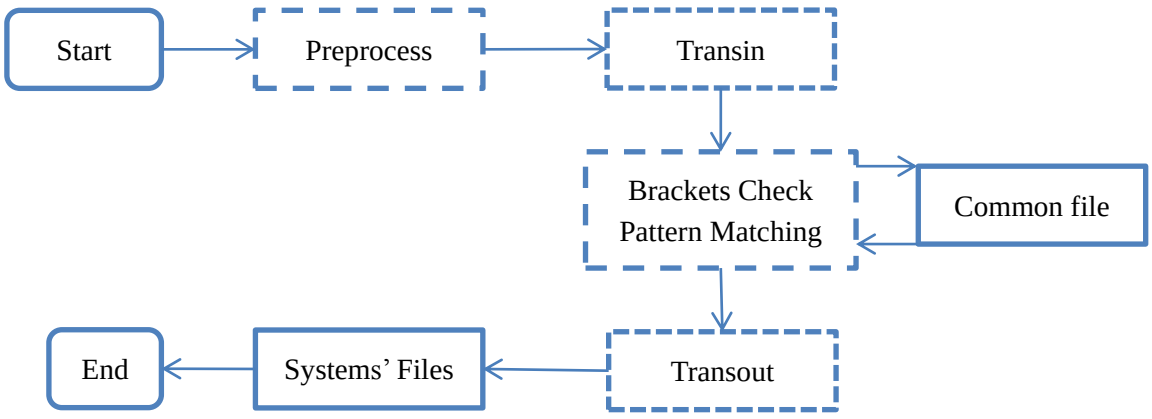


Figure 11 Converting Data Flow Chart

Figure 11 shows the process of converting data. After users upload files, "tranin" method convert the files into common file, and the data that are converted must satisfied the "Brackets Check" and "Pattern Matching". It is same when the common file transout into the five systems' files.

4.3.5 Summary

There some more flow charts involved, three of them have been illustrated in the section 4.2 for explaining Needs Analysis clearly. Some of the flow chart is not very important for the workbench,

which have not been listed here. These flow charts are used to developed the workbench, and make the developing process easier than without these charts.

4.4 Test Plans

If the process of development nearly finished, evaluation is a necessary part in the software development. If the workbench is successful and powerful, a huge number of data need to be tested through this workbench without any exception. Otherwise, the workbench cannot afford strange data which it never been considered. Therefore, the workbench needs to test as much data as possible. This data could from each system examples, and also can use data that created by people. In addition, the data that not belongs to the ILP and AV are also used to test the workbench to make the workbench stronger.

4.5 Conclusion

In this chapter, the developing languages have been discussed. Html, Css, JavaScript, Java and Shell are used together to developed the workbench. Needs analysis have been analyzed in common file, input, middle result, output and files management. Then the main workbench flow chart is created for developing the workbench. Finally, the test plans have been made for testing the workbench. All of these works are necessary in the process of developing software. In the next Chapter, the result is listed and explained in details.

5 RESULTS

This Chapter details the results of running the workbench. At the beginning, the environment of the workbench is described first. Following that, the final common file is explained by details. In the rest of this chapter, both interface and background of the workbench is illustrated.

5.1 Environment

CentOS5.6 has been used as Linux Operating System. Root user is chose to login in the project. Myeclipse9 is the development software to develop the workbench. Java environment is the last version of jdk which is JDK1.7.0 and JAVA_HOME, PATH and CLASSPATH are set in /etc/profile file. Apache-Tomcat7.0.19 is used as web server that can resolve Java codes. The whole path of the Apache-Tomcat7.0.19 is /usr directory. The systems Progol4.4, 1BCs, ACE-ilProlog-1.2.20, Prism1.12.1 and WEKA3.6.4 are installed into /root directory. WEKA do can install in any directory, because weka.jar is the only one used in the project for calling the weka algorithms which is needed to add in Referenced Libraries. Weka.jar allow the workbench to employ weka. Shell stores shells program. They are used to call linux terminal to run the particular system. They also need to be stored in /root directory with the execute permission. The file paths above must be satisfied. Otherwise, the workbench cannot run successfully. When the environment has already been set, the workbench could be run. In the practice, the workbench has been tested through IE9, Firefox 6.0.2 in Windows and Firefox 3.6.18 in Linux.

5.2 Common File

In section 4.2.1, the main structure of any file format can be divided into five parts, which includes settings, hypothesis, background knowledge, positive examples and negative examples. Therefore, the common file has been considered to store the data based on these parts. This section explains the style of the common file that is created by each file format.

5.2.1 Progol to CF

```
:- set(r, 1000)?
:- set(r,)?
:- unset(cover)?
:- fixedseed?
:- constraint(invent/2)?
:- modeh(1, class(+train,#direction)?
:- modeb(5, has_car(+train,-car)?
false:-class(T,eastbound),class(T,westbound).
number_of_cars(T,N):-
            train(T),findall(M,has_car(T,M),Ms),length(Ms,N),integer(N).
class(east1,eastbound).
:- class(west5,eastbound).
```

Figure 12 Progol Clauses

Figure 12 shows a progol file which is the example of east west train. Even though just small part of the whole data, it still contain all elements that progol file could have. Five colours are used here as signs.

Orange indicates settings, yellow denotes predicates, blue expresses rules (include integrity constraints), purple shows the facts, and negative examples are marked by green, finally, the gray lists error sentences. In the rest of the section, all examples use the same colour to indicate the data.

The following Figure 13 shows the common file that is converted by Progol file format.

```
from(Progol):-set(r),value(1000).
from(Progol):-unset(cover),value($).
from(Progol):-fixedseed(yes).
from(Progol):-parameter(constraint),value(invent/2).
from(Progol):-head_name(modeh),head_int(1),not(no),arity($),head_predict
(class),head_predict_name(+train,#direction).
from(Progol):-body_name(modeb),body_int(5),not(no),arity($),body_predict
(has_car),body_predict_name(+train,-car),repetition($),cwa($).
from(Progol):-integrity{false:-class(T,eastbound),class(T,westbound).}.
from(Progol):-rule{number_of_cars(T,N):-
train(T),findall(M,has_car(T,M),Ms),length(Ms,N),integer(N).}.
from(Progol):-fact{class(east1,eastbound).}.
from(Progol):-negative{:-class(west5,eastbound).}.
from(Progol):-other{:-set(r,)?}.
```

Figure 13 Common Format for Progol

From the two figures above, we could see the conversion from Progol to CF. The common file use the form *"from (Progol): -"* as the start of each sentence. Then the remaining sentence stores the real data from the Progol. The first four lines are settings in Progol. Keeping these data here is for make 100 present accuracy when it is converted back to Progol. The fifth and sixth lines show the mode declarations. This part can be shared with ACE and 1BC. Hence, the information is divided into several parts. The detail explanation is in 5.2.6. The following lines show the integrity, rule, fact, negative and other. All of them can be shared by other systems except WEKA. Braces are used to store all sentences from Progol.

5.2.2 ACE to CF

Figure 14 shows an ACE file, which is integrated from .s, .kb and .bg file. If the data do not belong to yellow, blue and purple parts, the workbench would store them into orange part. In fact, it works well. No setting can be shared between the different systems, because they employ the different algorithms based on different structures.

```
warmr_maxdepth(3).
predict(class(+Train,-Direction)).
rmode(5, has_car(+train,-car)).
label(X, pos) :- pos(X).
class(east1,eastbound).
pos?
```

Figure 14 ACE Clauses

The following Figure 15 shows the common file that is converted by ACE file format.

```
from(ACE):-rule{label(X, pos) :- pos(X).}.
from(ACE):-parameter(warmr_maxdepth),value(3).
from(ACE):-head_name(predict),head_int($),not($),arity($),head_predict(
class),head_predict_name(+Train,-Direction).
from(ACE):-body_name(rmode),body_int(5),not($),arity($),body_predict(
has_car),body_predict_name(+train,+-car),repetition($),cwa($).
from(ACE):-fact{class(east1,eastbound).}.
from(ACE):-other{pos?}.
```

Figure 15 Common Format for ACE

This common file is much similar with the last section. The only different part is in the start of the sentence -- *"from (ACE): -"*. The rest have same meanings as Progol.

5.2.3 1BC to CF

Figure 16 shows a 1BC file that include both .prd and .fct file. The orange part is the definition of the Properties as settings in 1BC. In Chapter 2, more fundamental knowledge has already been listed.

```
--INDIVIDUAL
train 1 train yes
--STRUCTURAL
car 2 1:train *:car cwa li
--PROPERTIES
class 2 train #class cwa
!
class(east1,eastbound).
class()
```

Figure 16 1BC Clauses

The following Figure 17 shows the common file that is converted by 1BC file format.

```
from(INDIVIDUAL):-individual(yes).
from(INDIVIDUAL):-name(train),arity(1),type_arguments(train),cwa(yes).
from(STRUCTURAL):-structural(yes).
from(STRUCTURAL):-name(car),arity(2),type_arguments(1:train,*:car),time(
1),cwa(yes),distribution(li).
from(PROPERTIES):-properties(yes).
from(PROPERTIES):-body_name($),body_int($),not(no),arity(2),body_predict
(class),body_predict_name(train,#class),repetition(1),cwa(yes).
from(BC):-block(!).
from(BC):-fact{class(t2,eastbound).}.
from(BC):-other{class()}.
```

Figure 17 Common Format for 1BC

There are three forms start of the sentences used to indicate different parts in 1BC. However, only the PROPERTIES with the body part and the facts except "!" can be shared by other systems.

5.2.4 Prism to CF

Figure 18 shows a Prism file. This could be from the .psm file or might be from .psm and .dat files. The

structure of Prism is very simple. Just rules and facts are involved.

```
prism_main([]):-hmm_learn(100),show_sw,viterbif(hmm([a,a,a,a,a,b,b,b,b,
b])).
values(out(_),[a,b]).
phenotype(ab).
aaa,df
```

Figure 18 Prism Clauses

The methods used here are also very simple. Storing any data from .psm file as rule, and the data from .dat file as facts as Figuer 19.

```
from(Prism):-rule{prism_main([]):-hmm_learn(100),show_sw,viterbif(hmm([a,a,
a,a,a,b,b,b,b,b])).}.
from(Prism):-rule{values(out(_),[a,b]).}.
from(Dat):-fact{phenotype(ab).}.
from(Dat):-other{aaa,df}.
```

Figure 19 Common Format for Prism

5.2.5 WEKA to CF

Figure 20 shows the weka file, which include relation, attributes and data. This is discussed in Chapter 2.

```
@relation eastwest
@attribute short {yes, no}
@attribute closed {yes, no}
@attribute class{eastbound, westbound}
@data
yes,eastbound
yes,
```

Figure 20 WEKA Data

Compare the Figure 20above, the information from Figure 21 would be easy to understand based on the last sections. Weka is an AV Learning system as discussed in Chapter 3. To be considered in the workbench is to show the common file is powerful (explanation in 5.2.7).

```
from(Weka):-weka_name(eastwest).
from(Weka):-weka_attribute(short),attribute_value({yes, no}).
from(Weka):-weka_attribute(class),attribute_value({eastbound, westbound}).
from(Weka):-weka_data_start(@data).
from(Weka):-weka_data(yes,eastbound).
from(Weka):-other(yes).
```

Figure 21 Common Format for WEKA

5.2.6 CF to others

First of all, the common file could convert back to its original file correctly based on the start words of

each sentence. Secondly, sharing data between different systems is also available. The following figure indicates the method have been used in this project to convert the ACE predicts and 1BC properties to Progol mode declarations.

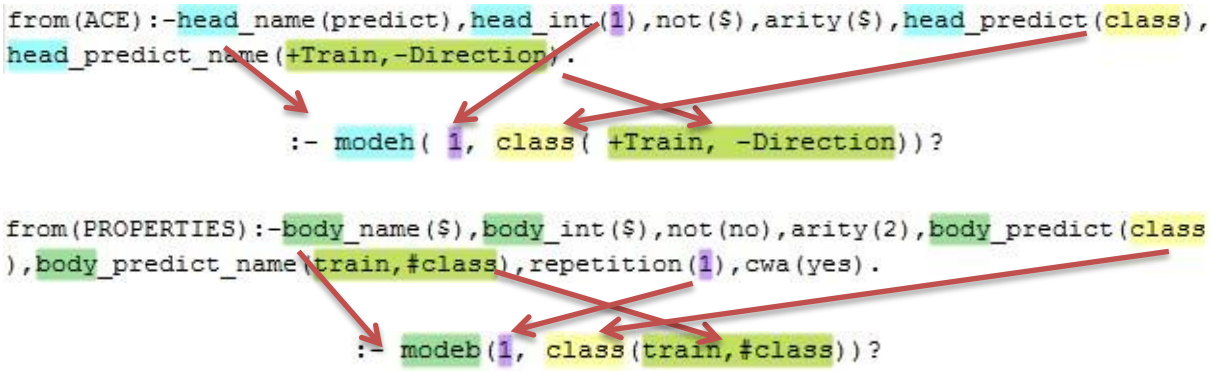


Figure 22 CF to Progol

From the Figure 22, it is clearly shown how common file share the data. In addition, the conversions are done by the workbench, there are always have some illegal programming for the particular systems. Therefore, some manual works are necessary.

5.2.7 Summary

The common file currently is satisfied to the workbench for the five systems. It is also a powerful common file. More systems in the workbench, more powerful the common file has. That is the reason why weka is integrated in this workbench. The arff file cannot share its data with other ILP Learning systems without using propositionalization (or some similar conversion algorithms), but it has not been affected to join into the common file. New style of the start of the sentence and some suitable predication could help any data format to be converted into common file and be converted from common file.

5.3 Interface of Workbench

The interface of the workbench involves nine pages, which are Home Page, Amending Page, Result Page, Error Page, Introduce Page and Help Page, Contact Page. Only the first three pages are used for analyzing data. In this section, the first three pages and some functions will be explained in detail.

5.3.1 Home Page

The home page is the first interface of the workbench as Figure 23 shown. The first page has shown data preparing in control penal and some introduction about the workbench in the side penal. As Figure 23 shown, users could upload their files in the control penal, and then the file list will be listed in the following subpanel. After finishing uploading files, users should choose one of the systems Progol, WEKA, 1BC, ACE and Prism. Progol is the default choice. The common file also is listed in the panel that helps users to go to common file page directly. It is also allowed users use this workbench without upload any files. The workbench will create empty files for users. Finally, when the first step has been finish, users can click *Next* to go to the amending page.

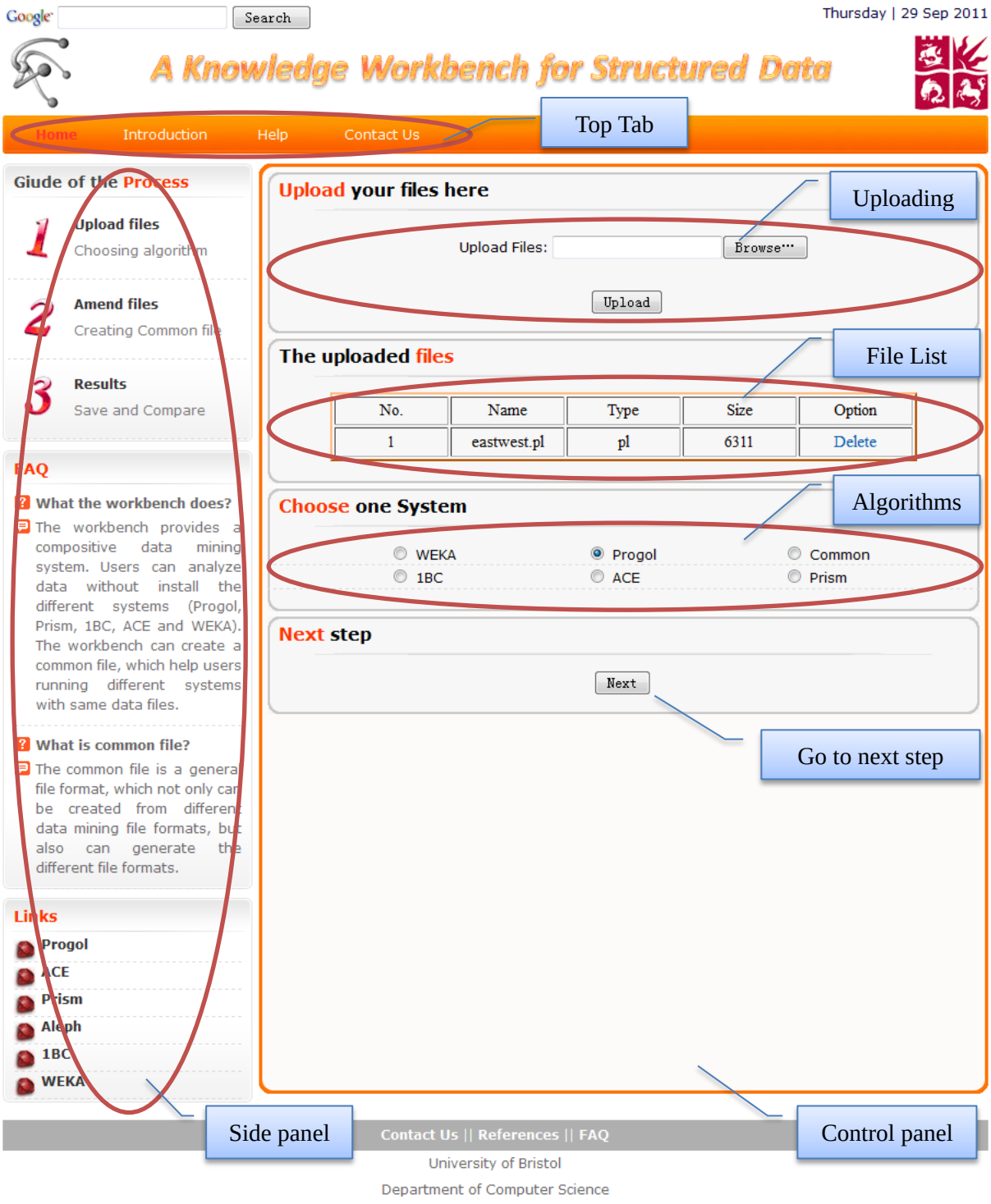


Figure 23 Home Page

5.3.2 Amending Page

Figure 24 shows the amending page in Progol part, the rest of the systems are demonstrated in section 5.5. As you can see in this figure, there are two main panels, which are updating panel and prompts panel. Users could use the information from prompts to amend their uploaded files (Progol just has one data file). From prompts panel, users could see the warnings, errors, integrity constraint, settings, mode declarations, rules, positive examples, and negative examples. All of them are counted in the subtab of each prompt.

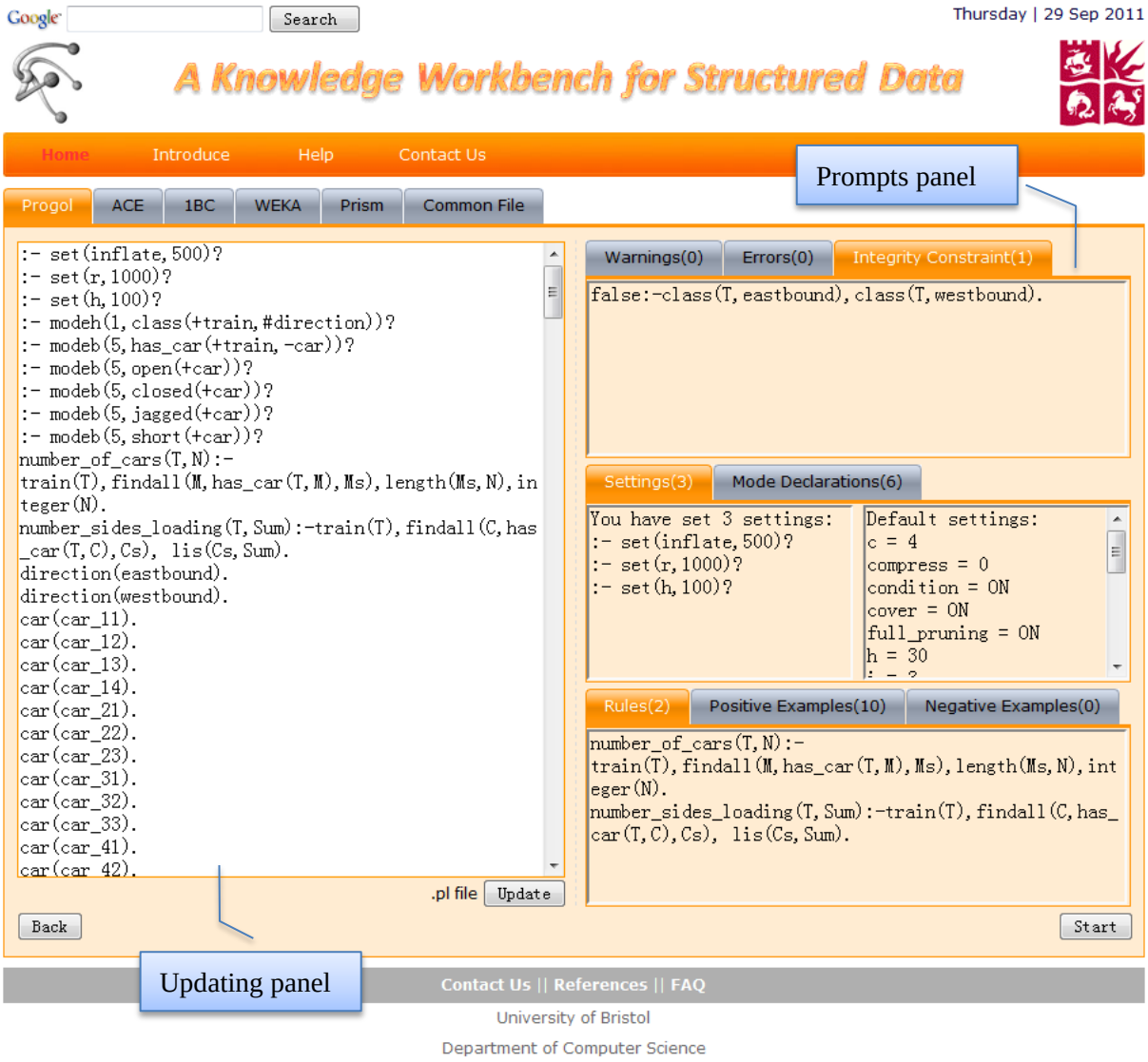


Figure 24 Amending Page

5.3.3 Result Page

Figure 25 shows the result page which contains three parts, the results, the amended original files and the common file. Users can read them through this page, and also can download them.

5.3.4 Summary

There are totally three steps in this workbench. Some useful prompts are provided when users amend their data. These characters of the workbench are definitely helpful to make a successful analysis of data for users who do not have depth background knowledge in such fields. These three pages have covered all of the process of analyzing data. However, the background of the workbench takes more responsibility for the workbench, which is described in the next section.

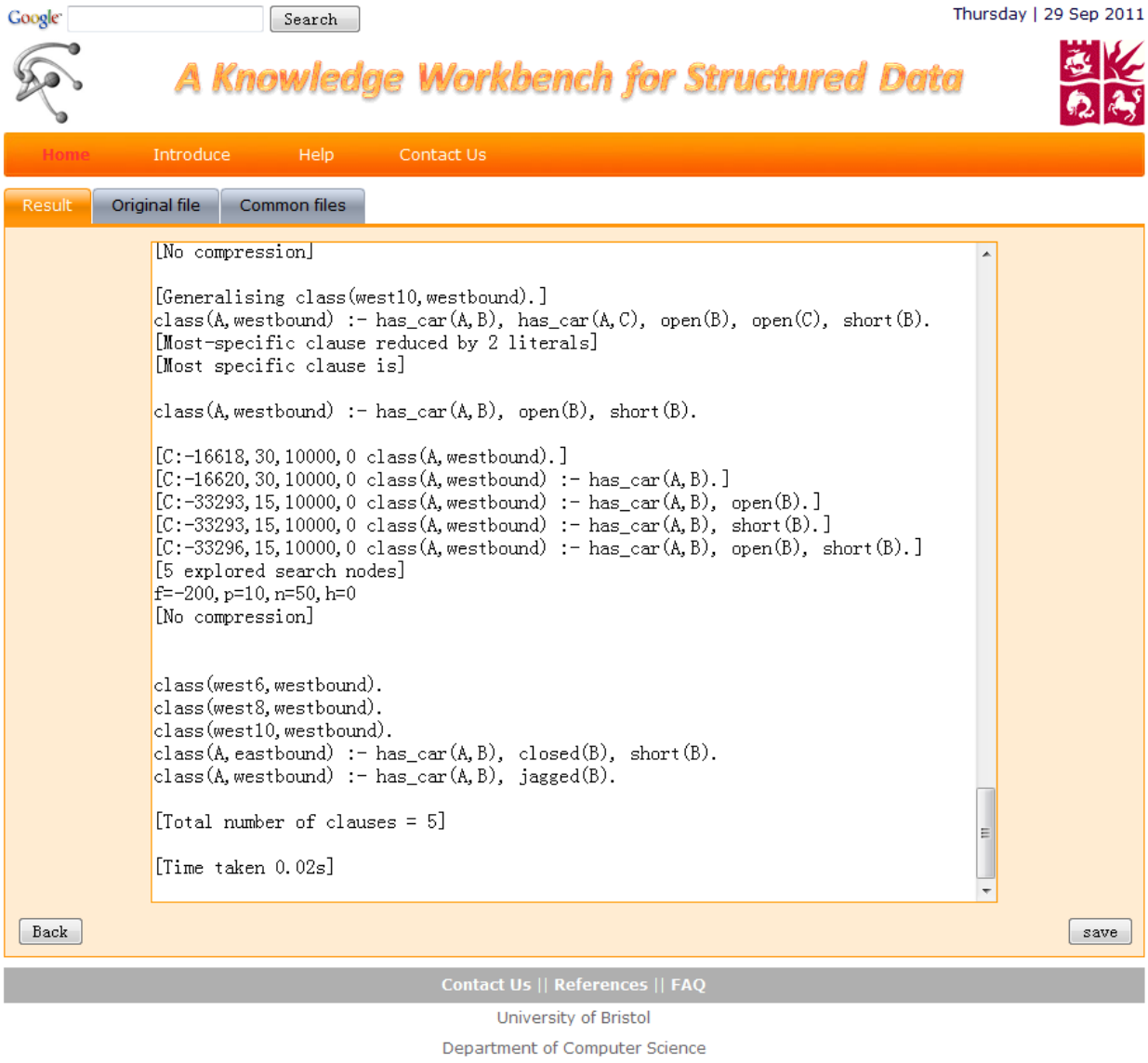


Figure 25 Result Page

5.4 Background of Workbench

In the background of the workbench, all Java methods are utilized by the workbench for supporting conversion data and file management. It also involve a number of pages, all of them are used to handle the conversion of the data, running the algorithms and the management of the files.

5.4.1 Converting Data

As mentioned in the Chapter 4 and described in section 5.2, converting data is the core of the workbench, because all of the uploaded file will be convert into common file firstly. The data can be seen in the amending page is converted from common file. When users update their amended data, the common file would be generated automatically, then converted into another system file format except the one that users amended. There are two java methods, called transin and transout, to take charge the conversion of the workbench. Transin methods control the input data. Because of the diversity of the data format, particular methods have been made for each system file format as shown in Figure 26.

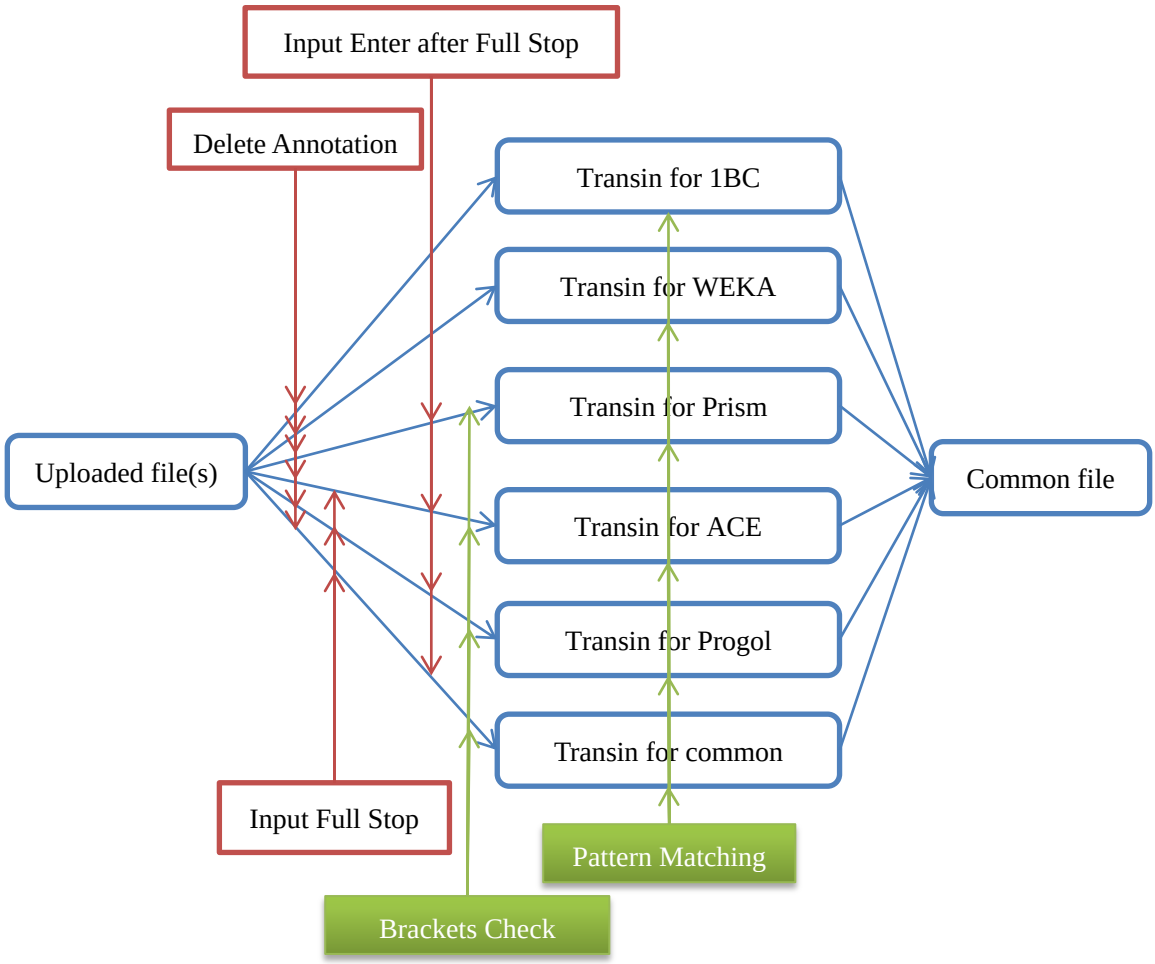


Figure 26 Converting Data into Common File

Figure 26 clearly shows how the uploaded files convert into common file. Methods in the red box means the preliminary preprocessing, which include delete all annotation, input full stop at the end of the sentence (there is no full stop symbol like ".", "?") and input enter after full stop (this methods could make each whole sentence only be store in one line). Some of the system file formats do not need Input Full Stop or Input Enter after Full Stop. After the preprocessing, two more methods are used to choose correct data to convert into common file as a filter. Full Pattern Matching is used here to choose suitable data that can avoid Array Index out Of Bounds Exception when the workbench converts data. Any unsatisfied data will be store as *"from (InputFileType):-other {content}."* For example, the following pattern matching could match all correct head mode declaration for Progol:

"^\\s*:-\\s*modeh\\b\\s*\\(\\s*([\\-0-9*]*\\s*\\,)+\\s*(not)*\\s*[_a-z]+\\s*\\
(\\s*[\\s\\+\\#\\-\\(\\)\\[\\]\\|\\=_a-zA-Z0-9]+\\s*(\\,\\s*[\\s\\#\\+\\-\\(\\)\\[\\]\\|\\
=_a-zA-Z0-9]*)*\\s*\\)\\s*\\)\\s*\\?\\s*$"

All the other pattern functions are used same approach.

On the other hand, when the common file converts into the systems' files, the exactly same approaches are used for each particular system's files. However, no preliminary preprocessing (red boxes in Figure 26) is used here. Partial of the codes of transin and transout methods are listed in the appendix.

5.4.2 Calling Systems

The workbench has integrated Progol, ACE, 1BC, Prism and WEKA all together instead of creating such data mining systems. Therefore, the workbench has the responsibility for calling these systems to calculate the results. For Progol, Linux command is required for calling the system, and then the result would be displayed in the Terminal. Thereby, *"Runtime.getRuntime().exec(command);"* (java codes) has been used here for connecting Linux Terminal. Finally, using input and output stream store the results into the file. In addition, the command must include the path of the Progol directory. For ACE, Linux command is also required. However, when the users' data include some errors, the terminal would be stuck for waiting the standard input. There is no result if the same methods used as Progol. Therefore, a new Java method has been used here, called *ProcessBuilder*, to accomplish our requirement without any stuck. For 1BC, the requirements are similar with the above systems, but it is always needed input arguments to run the system. Using the same methods as Progol can handle it. For Prism, as mentioned in Chapter 2, it is required input command to run Prism. Nevertheless, Prism has not provided an interface for Java. Fortunately, Prism has an *upprism* command that can run the prism without any interactive execution [19]. We just need to add a clause body of prism_main0/1 into the first of the .psm file for a batch execution and then use the same methods as Progol. For WEKA, because it is a free open system that is developed by Java, the workbench just put the weka.jar in the References Libraries for call the algorithms in WEKA.

All of the system can be run through the workbench. However, one more difficulty has been encountered. When the users update their data through website, the <textarea> tabs (HTML) will store an invisible enter symbol in the file. The systems cannot deal with such files. Therefore, *dos2unix* command is required to delete such symbol. For convinent, all the ILP Learning systems need a couple of commands. Shell is used here for combine these command, then just calling the shell to deal with the data is the best way to calculate the results. The partial shell codes have been listed in the Appendix.

5.4.3 Summary

The background of the workbench can be seen by the users, but they provide an irreplaceable important role of the workbench. Whether the workbench is successful, the background holds more than 50 present. The interface of the workbench only can show the information of the data, but the background offer the most of the calculation for the workbench. After the conversion and calling systems work well, the main systems' results will be illustrated in the following section.

5.5 Each System Results

In this section, the results of each system will be explained. Because of the involved pictures are big, some interface will just explain once by one of the systems in the workbench. In addition, the result from each system is not explained in this project, because the project is focus on the workbench function. The following Figure 27 is the tab of the workbench. User could choose them to run the first five systems. The last tab indicates the common file data and the information that is included in common file.

Figure 27 Top Tags

5.5.1 Progol Results

The Results of Progol part has been displayed in section 5.3 as an example of interface of the workbench. Figure 28 indicates that the original file is displayed in the Result Page, and users can download the file.

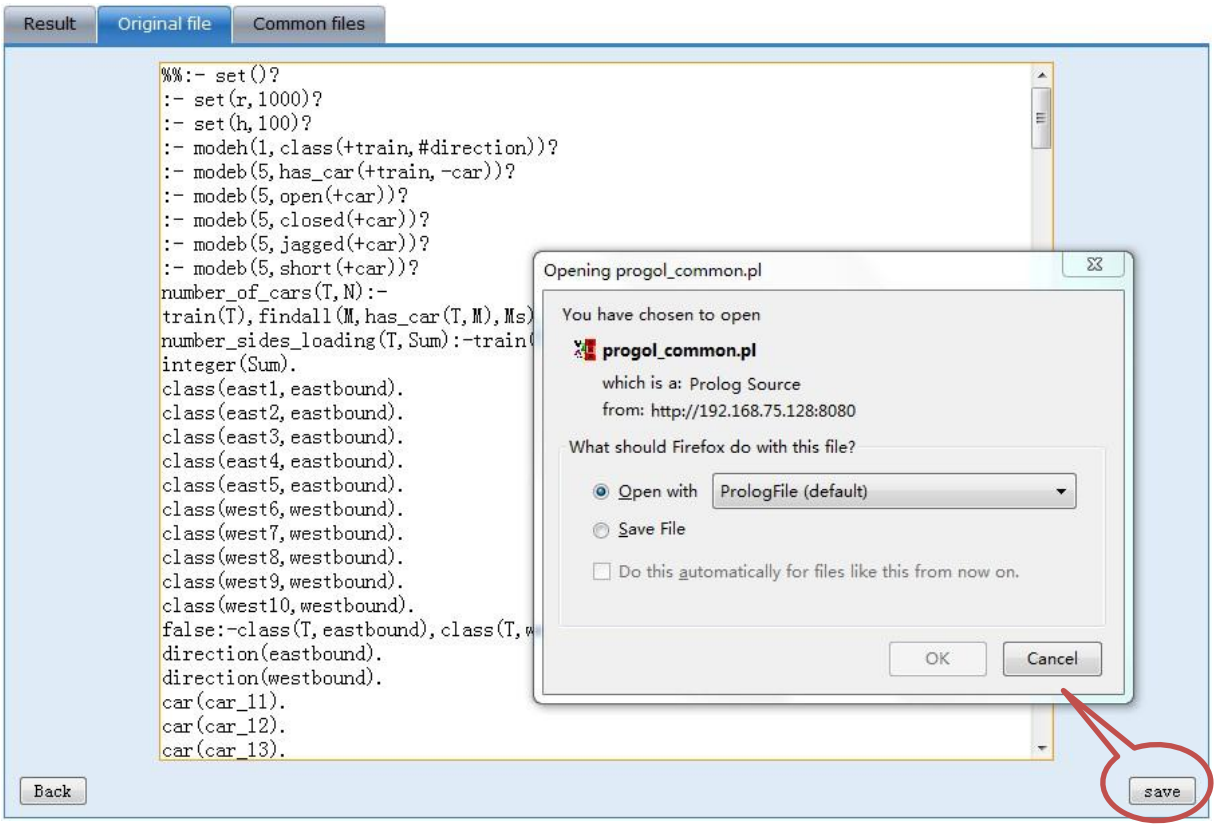


Figure 28 Progol Result

5.5.2 ACE Results

In ACE part, as we can see in Figure 29, the uploaded files are displayed separately in the left of the workbench (data from ACE examples Bongard). Users can update the data and click Update to save the amended files. In the right of the workbench, there are some similar prompts with Progol, which include the number of warnings, errors, parameters, mode declarations, rules and examples. In the parameters part, there are 26 parameters have been set, which are shown in the left of this part. The other side shows the default parameters in ACE. Users can find the parameters they want to used or check the default value use this part.

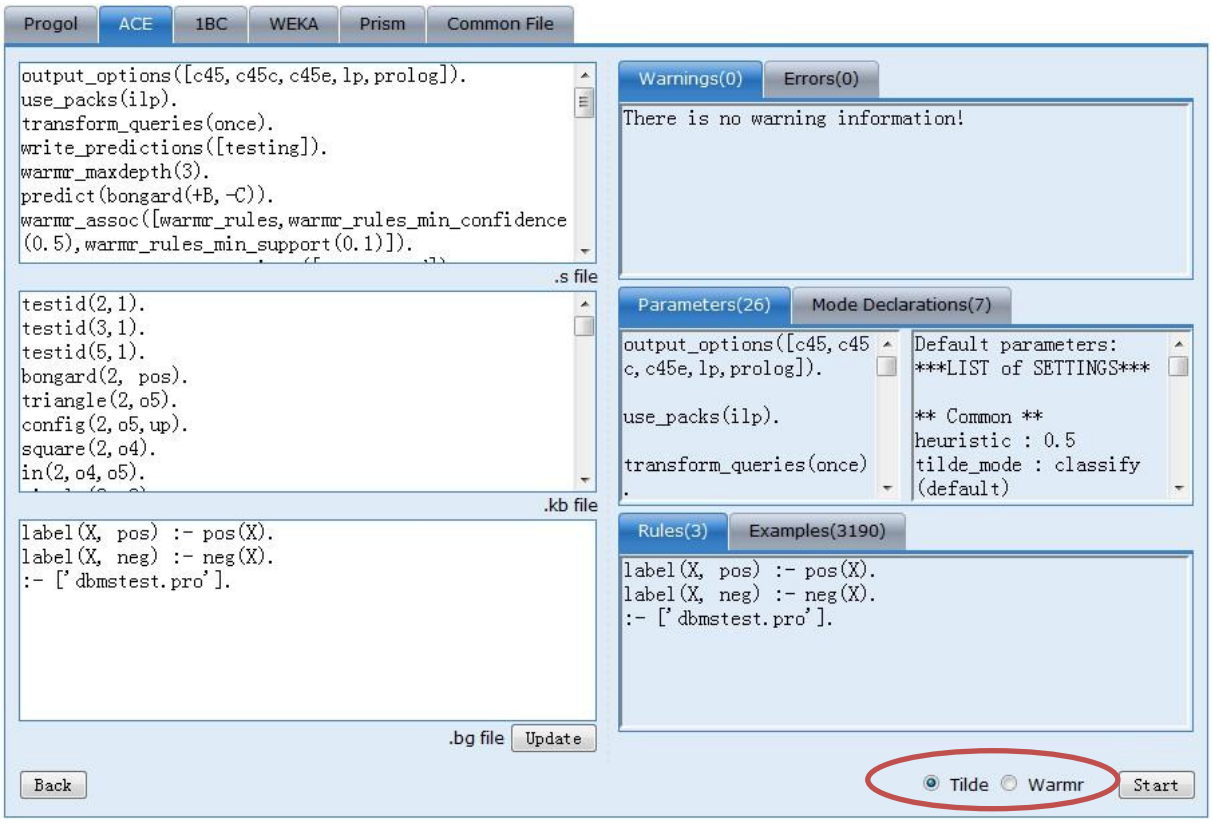


Figure 29 Amending Page for ACE

At the bottom of the amending page, there are two algorithms that can be chosen by user to run Tilde and Warmr in ACE system.

The following Figure 30 shows the rest of the prompts that are hidden in the Figure 29. The mode declarations part shows the declarations that users have already set in the left, and the other side shows the optional declarations in the data files. This prompts is same as Progol system.

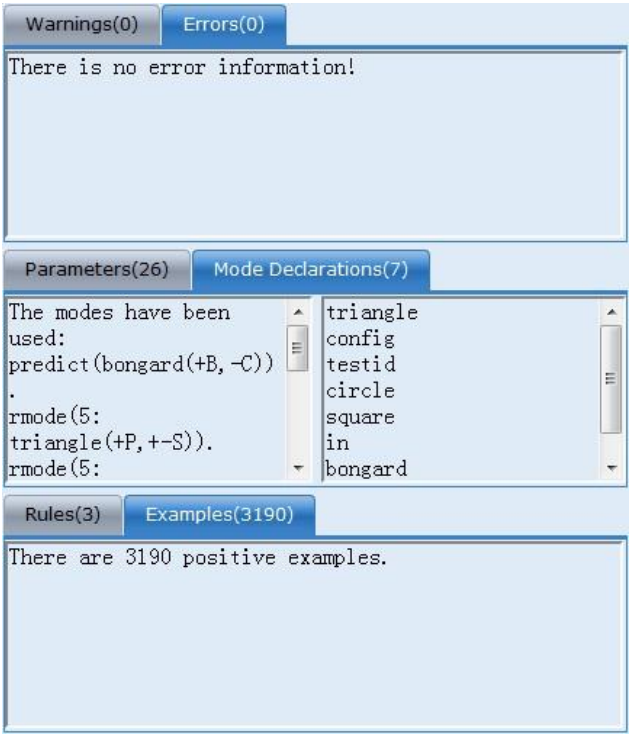


Figure 30 ACE Prompts

The following Figure 31 shows the result of Bongard by running Tilde.

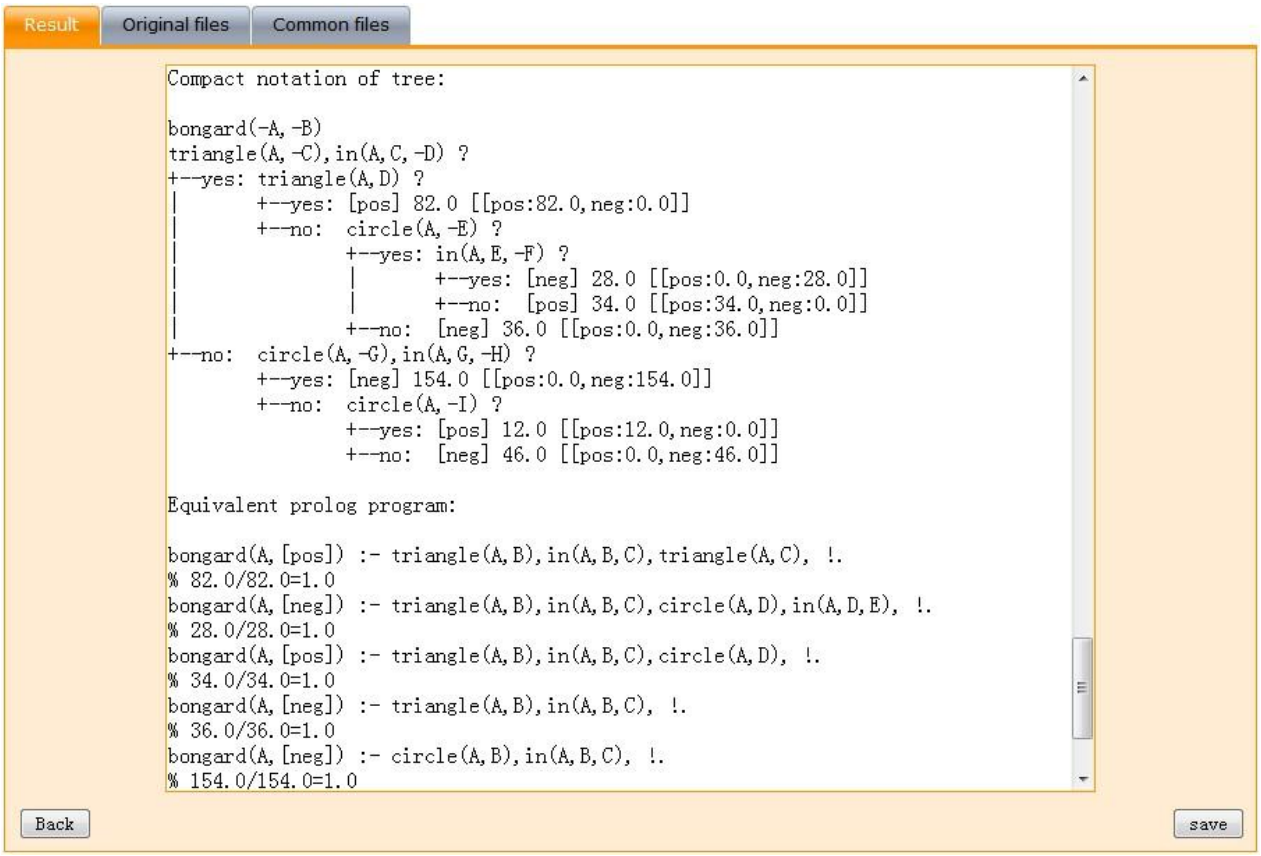


Figure 31 Result of ACE (Tilde)

5.5.3 1BC Results

Figure 32 shows the 1BC Amending Page, the uploaded data from 1BC example -- eastwest. The edit part is similar with Progol and ACE. However, the prompts part have some differents. As we can see the bottom of the figure, 1BC need users set the parameters to run the system. Setting parameters part give the two optional algorithms that 1BC and 1BC2, and then users need to choose one of the stems that is listed predicts in the .prd file. Finally, users choose the number of literals and the number variables.

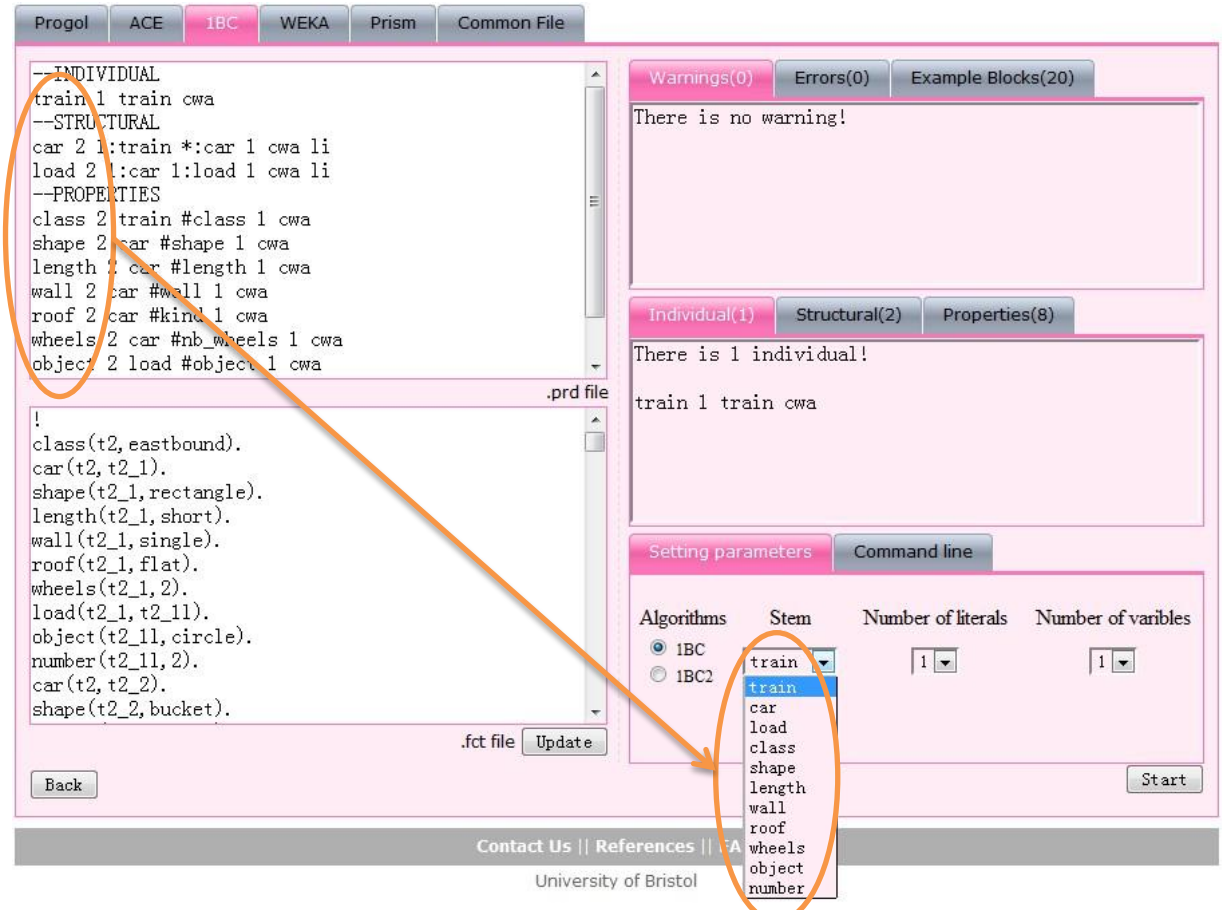


Figure 32 Amending Page for 1BC

On the other tabs in the bottom prompts which is shown in Figure 33, the workbench provides command line for users who can use commands. The files name has been changed during data conversion. Therefore, the final part of the command which is the name of the file can be ignore in the command line, even the users use their original file name it also can go through the workbench, because the name of the file will be ignore.

Figure 33 1BC Prompts

After given the command line above or set parameters in the first tab, the result can be given as following Figure 34.

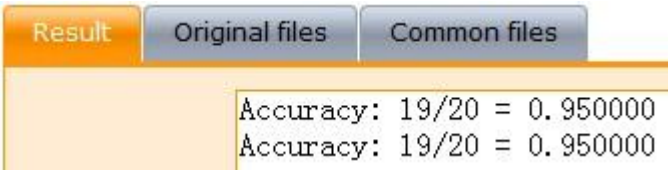


Figure 34 Result of 1BC

5.5.4 WEKA Results

Weka includes a lot of algorithms for AV data mining. The workbench now just import J48 into the classify parts. The example used here is the weather.arff in weka data file. There is no preprocessing for the change of data. Nevertheless, we also can update the data by hand as shown in Figure 35:

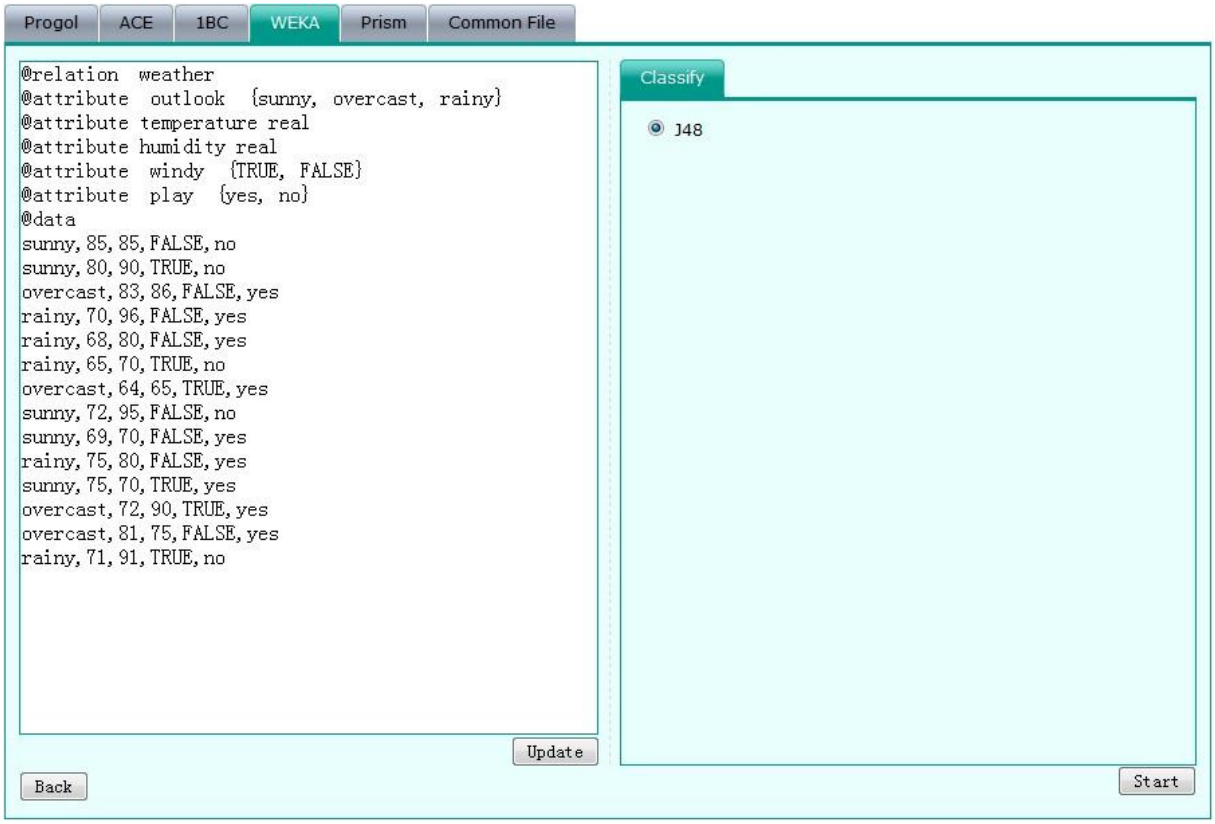


Figure 35 Amending Page for WEKA

After clicking Start, the following result Figure 36 is given.

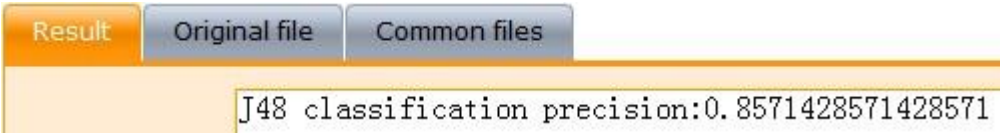


Figure 36 Result of WEKA J48

5.5.5 Prism Results

Figure 37 shows the Prism amending page. The left of the workbench and the top two tags are similar with the ACE. Here only the bottom tab on the right is needed to be explained. In this workbench, Prism can be run by “upprism” command, because it does not need interactive execution. Users need add the

prism_main0/1 in the upprism tab. It is all the same when people use prism in the terminal. The figure 38 shows the result of the prism after we put:

prism_main([]):-hmm_learn(100),show_sw.

in the upprism part.

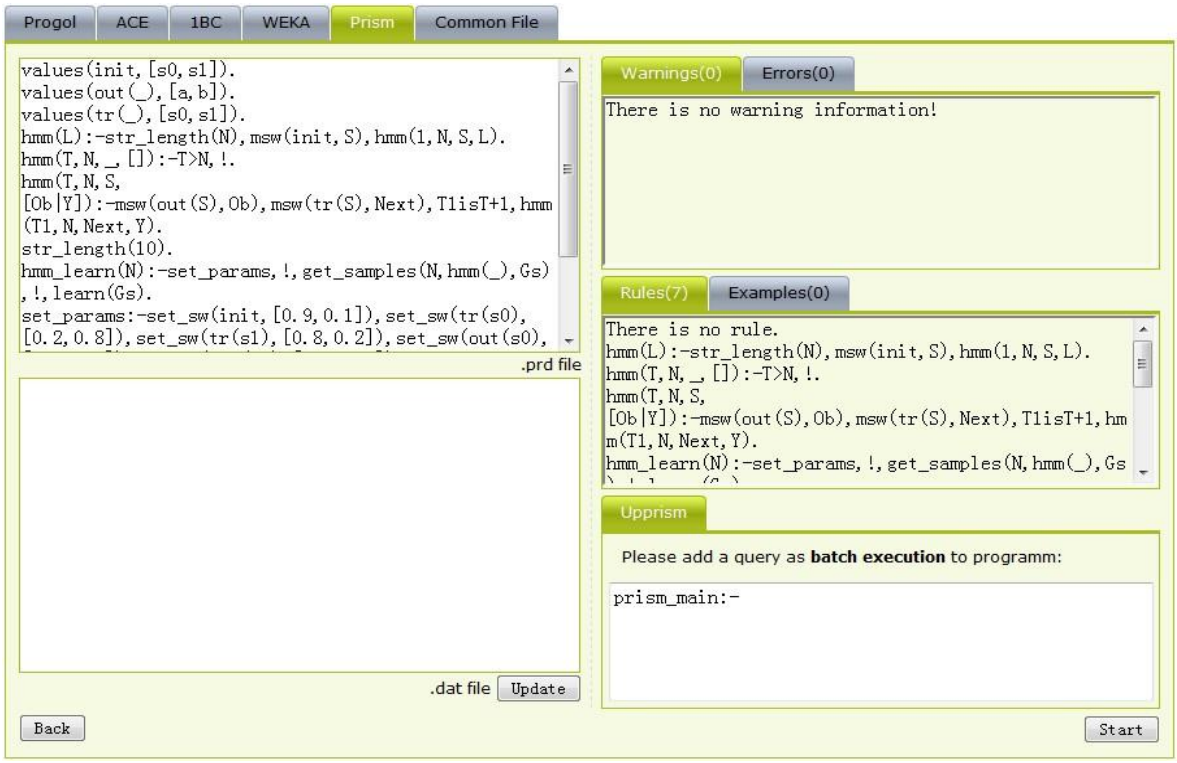


Figure 37 Amending Page for Prism

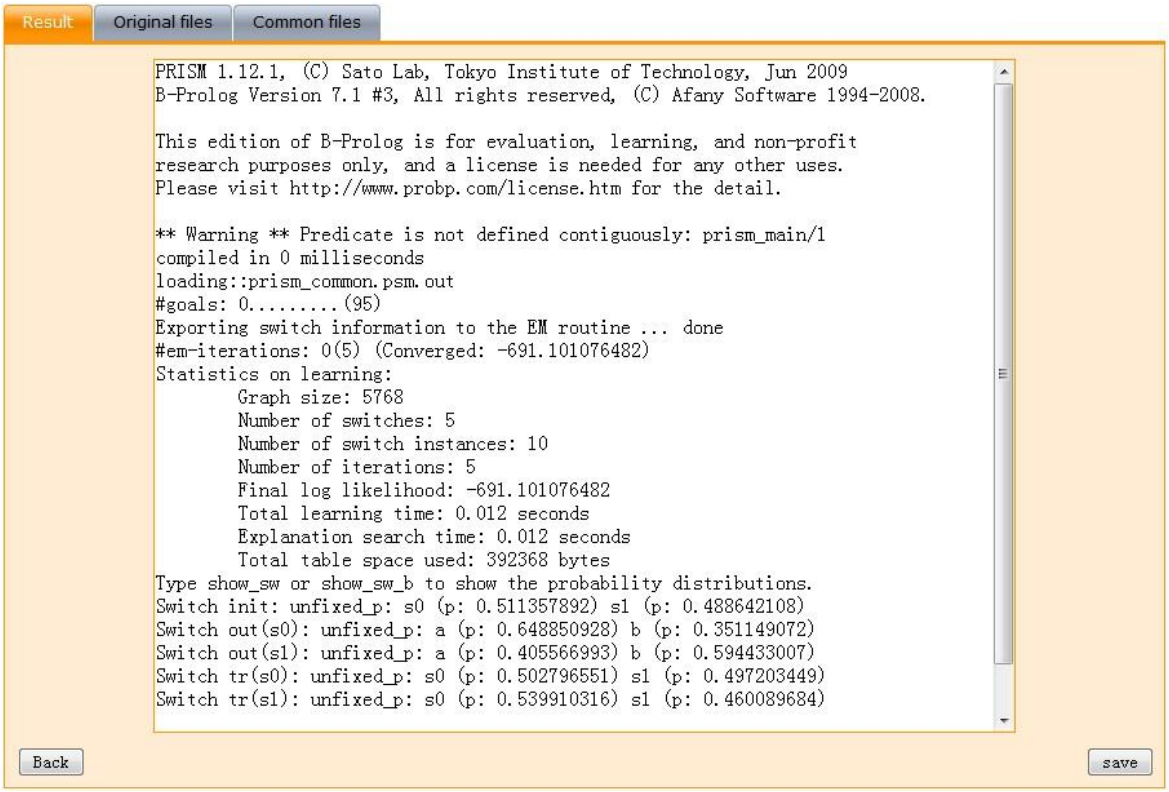


Figure 38 Result of Prism

5.5.6 Common Results

Figure 39 shows the common amending page. The example used here is same as Progol. The common file also has two parts, updating and prompts. In the prompts part, the information about Progol, 1BC and ACE are listed in this part, and also the rules and examples (as facts), which can be consult by Prism users. At the right bottom of page, there are four acceptable systems which can be run by the data in the common file is listed. Users can choose one of the following (highlighted below) to run the system directly. The result page would be one of the result pages above.

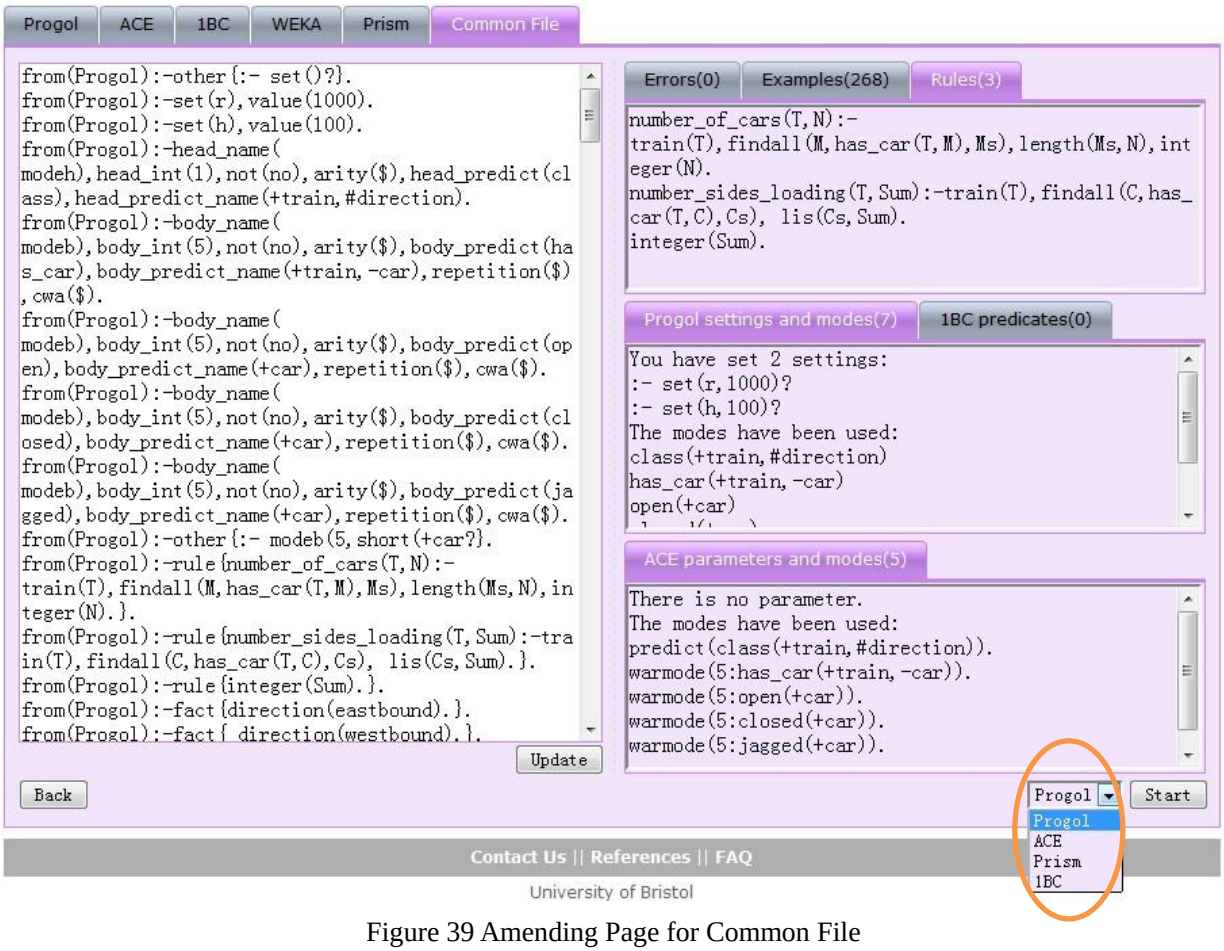


Figure 39 Amending Page for Common File

5.5.7 Summary

As we can see in the section 5.5, the workbench can successfully run all of the five systems and the prompts can provide the correct information to users. In addition, all of the functions can be run successfully like Update, Back, Next, Start and Save. In the next Chapter, the main conclusions are given.

6 CONCLUSIONS

This chapter discusses the result. Whether the workbench satisfied the initial design requirement or not? Comparing with other workbenches, which advantaged and disadvantages it has? Finally, reviewing the aims and objectives of the project to judge whether the workbench has been accomplished and if so how successfully it is.

6.1 Common File Format

The current common file format allows interoperability of the different systems and their proprietary file formats as the example used in section 5.2. The common file is the center of the workbench. Most of the methods are used to deal with the different data formats as described in section 5.4.1. From the results part, all systems can work successfully that indicate the common file have done well.

6.2 Structure of the Workbench

The workbench used JSP to develop as a website. Any people can use this workbench through the Internet. It is absolutely one of the most convenient data mining workbenches. Users can use five systems to analyze data at one time without install any one systems. The workbench could automatically help users converting their data into other data formats and show the prompt when the users amend data. Users do not need strong background knowledge of this area to use the workbench.

6.3 Evaluation

Based on the aims and objectives of the projects, the following part is considered to evaluate, which includes the whole workbench, the common file and the exceptions.

6.3.1 Web Standard Workbench

The first aim of the project is to develop a KD workbench for structured data which integrate Progol, ACE, 1BC, Prism and Weka systems. This was completed with the JSP, Oracle and those systems. Therefore, this aim can be considered a success as it met the requirement of using the workbench and returning the specific results.

6.3.2 Converting Data File

The second aim of the project is to create a common file that can store all the information from the different file formats and also can convert data from the common file to each other file formats. The common file also can assort the data. If the meanings of the data are similar or same, it will be stored in one single sentence and also can convert to these file formats that need the sentence. This is also accomplished with Java. If the upload data is intact, the common file could store all the information without any exception. In addition, when the common file is generated, the common file also can confirm some type of illegal data. However, not all of the data can be converted correctly. It will trigger the exception when the workbench receives the illegal data that have not been considered.

6.3.3 Exception

The number of the exception happened during the test of the workbench is one of the most important criterion of evaluation. When the error encountered, the web.XML of the workbench could rout up the following error page as Figure 40 instead of the 404 or 500 error pages.

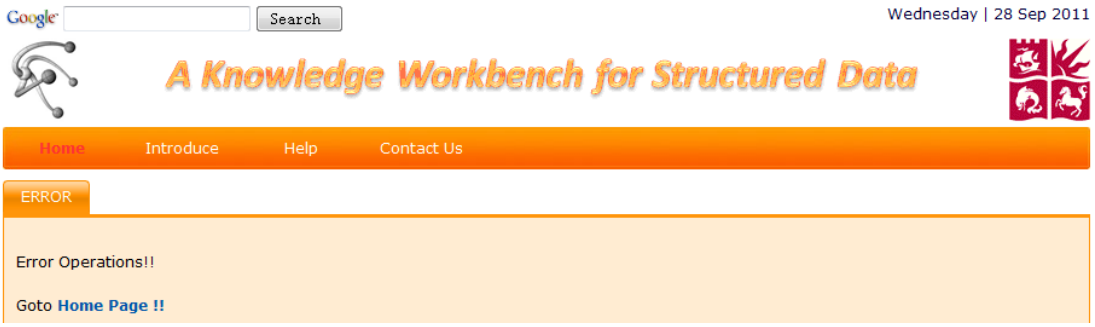


Figure 40 Exception Page

Hitherto, no exception encountered based on the powerful test file. However, it cannot be guaranteed to all the cases.

7 FUTURE WORKS

This project is actually a huge project and each part of the workbench has lots of knowledge that are needed to be studied. This chapter lists the future works which are not achieved at recently.

7.1 Extension of the Workbench

The extension of workbench has two parts. One is based on the self of workbench, another is the common file.

7.1.1 Workbench

In recent years, lots of data mining systems have been developed for meeting the increasing demands of people, like Oracle data mining system, Orange data mining system. They are all very good KD workbench. If a KD workbench is most powerful, the workbench must have the most algorithms can be run, the most convenient operations need to be used, the fastest can find out the results and the most accurate on the results. From this point, the project has really lots works, like increase the speed and the accuracy of transfer data, integrate more data mining systems and improve the common file.

7.1.2 Common File

Although the common file can satisfied the workbench for the five systems, Progol, ACE, 1BC Prism and Weka, it still cannot be used as a comprehensive common file. There are several popular data mining systems, such as FOLP and Aleph, have not been considered in the project. The common file is not a really common format if some data mining systems have not been considered. Therefore, integrating more data mining systems would make an improvement for generating much better common file.

7.2 Database Services

When the databases are discussed, few people may be strange about them, particularly in data mining field. In this project, database technique has not been used. This is a big limitation of the workbench when users want to use their own database to transfer data. If they want to use the workbench, making a data file is required at the beginning. Therefore, the workbench should develop a connection between users' database and itself, like Weka. That is absolutely essential part of a successfully compositive workbench.

7.3 Conclusion

The listed future works above are important. Finishing these functions, however, are not enough. For convenient, the workbench should provide more correct prompts during the users amending data. For accuracy, a perfect common file and transfer algorithms will be required. Nevertheless, there is not enough or mature theory to support this idea. To consummate the workbench, people need a mass of work. That would be long rough way.

BIBLIOGRAPGY

[1] Agrawal R., H. Mannila, R. Srikant, H. Toivonen, A. I. Verkamo. *Fast discovery of association rules.* In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, pages 307-328. The MIT Press, 1996.

[2] Blockeel H. , L. Dehaspe, J. Ramon, J. Struyf, A. Van Assche, C. Vens, D. Fierens. *The ACE Data Mining System User's Manual*. Katholieke Universiteit Leuven, 2009.

[3] Blockeel H., Luc De Raedt. *Top-down induction of first-order logical decision trees*. Katholieke Universiteit Leuven, Department of Computer Science Celestijnenlaan 200A, 3001 Heverlee, Belgium. Artificial Intelligence 101 285-297, 1998.

[4] Bongard M.. *Pattern Recognition*. Spartan Books, 1970.

[5] Flach P.. *Simply Logical Intelligent Reasoning by Example*, the Netherlands John Wiley, 1994.

[6] Flach P. and Lachiche N.. *1BC: A First-Order Bayesian Classifier*. In: Ninth International Workshop on Inductive Logic Programming (ILP'99), S. Dzeroski and P. Flach, editors, pages 92--103. Springer-Verlag, June 1999.

[7] Krzysztof Cios, Witold Pedrycz, Roman Swiniarski. *DATA MINING Methods for KNOWLEDGE DISCOVERY.* Kluwer Academic Publishers, 1998.

[8] Lachiche N. and Flach P.. *1BC2: a true first-order Bayesian classifier.* In: Proceedings of the 12th International Conference on Inductive Logic Programming, pages 133--148. Springer-Verlag, July 2002.

[9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten. *The WEKA Data Mining Software: An Update. SIGKDD Explorations*, Volume 11, Issue 1, 2009.

[10] Muggleton. *Inverse entailment and Progol.* New Generation Computing, 13:245-286, 1995.

[11] Muggleton, John Firth. *CProgol4.4: a tutorial introduction*. Department of Computer Science, University of York, United Kingdom.

[12] Nilsson U., Jan Maluszynski. *Logic programming and progol (2ED)*. John Wiley & Sons Ltd. 2000. Page 19-31.

[13] Pfenning F.. *Logic Programming*. Carnegie Mellon University. 15-819K. 2006. Page 1-14.

[14] Quinlan J.R.. *Induction of decision trees*. Machine learning, 1:81-106, 1986.

[15] Quinlan J. R.. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in Machine learning. Morgan Kaufmann, 1993.

[16] Raedt. *Logical and Relational Learning*. Springer-Verlag Berlin Heidelberg, 2008. Page 1-287.

[17] Ray O.. *Hybrid Abductive Inductive Learning*, PhD thesis, University of London, Imperial College of Science, Technology and Medicine, Department of Computing, UK, 2005.

[18] Srihari K. S.. *BET: An ILP Workbench with a Generic ILP Engine*. Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, Mumbai 400 076.

[19] Taisuke Sato, Neng-fa Zhou, Yoshitaka Kameya and Yusuke Izumi. *PRISM User's Manual (Version 2.0).* Tokyo Institute of Technology, CUNY Brooklyn College, 2010.

[20] Zupan Blaz, Gregor Leban, Janez Demsar, Tomaz Curk. *Widgets and Visual Programming. Orange Data Mining Fruitful & Fun*, 2010.

APPENDIX A: Source Code (Partial)

Download page for Progol Result

```
<%@ page language="java" import="java.util.*,java.io.*,com.jiao.file.*"
    pageEncoding="GBK"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>A Knowledge Workbench for Structured Data -- Download
            Progol Result Page</title>
    </head>
    <body>
        <%
            String path11 = this.getServletContext().getRealPath(
                    "./files/resultprogol/");
            File file11 = new File(path11);
            File[] files11 = file11.listFiles();
            String file = "" + files11[0];

            // clear outputstream
            out.clear();
            out = pageContext.pushBody();

            //import smartupload
            SmartUpload upload = new SmartUpload();
            upload.initialize(pageContext);
            upload.setContentDisposition(null);
            upload.downloadFile(file);
        %>
    </body>
</html>
```

The following code is from layout.css that decide the global style settings

```
@charset "utf-8";
/* CSS Document */

/* Global style*/
body { margin:0 auto; font-size:12px; font-family:Verdana; line-height:1.5;}
ul,dl,dd,h1,h2,h3,h4,h5,h6,form,p { padding:0; margin:0; font-family: Verdana; }
ul { list-style:none;}
img { border:0px;}
input,button,select,textarea{ outline:none;}
textarea{resize:none;}
```

```
a { color:#05a; text-decoration:none;}
a:hover { color:#f00;}
.clearfloat {clear:both;height:0;font-size: 1px;line-height: 0px;}
/* Global style-end*/
```

The following codes are the partial of the tranin methods for Progol

```
    private DeleteAnnotation deleteannotation = new DeleteAnnotation();
    private Check check = new Check();
    private AppendFiles appendFile = new AppendFiles();
    private Enterafterstop eas = new Enterafterstop();
    private Inputfullstop ifs = new Inputfullstop();

    // Transform the progol file into common file
    public void transinForProgol(File filePathUpload, File filePathComm,
            File fileNoAnn, File tempProgol, File temp) {
        String str = null;
        String commonFile = filePathComm + "/" + "commonfile.cmff";

        // Delete annotations in the files
        deleteannotation.deleteAnnotation(filePathUpload, fileNoAnn);

        /*
         * Input full stop for the sentences which do not have ",", ":-", "?"
         * and "."
         */
        ifs.inputFullStop(fileNoAnn, temp);

        // Each sentence end by full stop
        eas.enterafterstop(fileNoAnn, temp, tempProgol);

        String[] file = tempProgol.list();
        String fileProgol = tempProgol + "/" + file[0];
        try {
            BufferedReader inProgol = new BufferedReader(new FileReader(
                    fileProgol));
            while ((str = inProgol.readLine()) != null) {
                String content = "";
                if (StackCheck.stackCheck(str)) {
                    if (check.check(str, ".*[^\\?\\.]$")) {
                        content = "from(Progol):-other{" + str + "}.";
                        appendFile.appendFile(commonFile, content + "\n");
                    }
                    // match set and unset
                    else if (check
```

```
                .check(str,

"^\\s*\\:\\-\\s*((un)*set\\s*\\(\\s*[a-zA-Z]+\\s*(\\s*\\,\\s*[0-9a-zA-Z]+\\s*)?\\s*\\)\\s*\\,?\\s*)+\\s*\\?\\s*$")) {
            String[] dataArr = str
                    .split("\\:\\-|\\s*\\)\\s*\\,\\s*|\\s*\\)\\s*\\?\\s*");
            for (int i = 1; i < dataArr.length; i++) {
                if (check.check(str, ".*set\\b.*")) {
                    String[] dataArr1 = dataArr[i]
                            .split("\\s*\\(\\s*");
                    if (check.check(dataArr1[1], ".*\\,.*")) {
                        String[] dataArr2 = dataArr1[1]
                                .split("\\s*\\,\\s*");
                        content = "from(Progol):-set("
                                + dataArr2[0] + "),value("
                                + dataArr2[1] + ").\n";
                    } else {
                        content = "from(Progol):-set("
                                + dataArr1[1] + "),value($).\n";
                    }
                } else if (check.check(str, ".*unset\\b.*")) {
                    String[] dataArr1 = dataArr[i]
                            .split("\\s*\\(\\s*");
                    if (check.check(dataArr1[1], ".*\\,.*")) {
                        String[] dataArr2 = dataArr1[1]
                                .split("\\s*\\,\\s*");
                        content = "from(Progol):-unset("
                                + dataArr2[0] + "),value("
                                + dataArr2[1] + ").\n";
                    } else {
                        content = "from(Progol):-unset("
                                + dataArr1[1] + "),value($).\n";
                    }
                }
                appendFile.appendFile(commonFile, content);
            }
        }
        // match fixedseed
        else if (check.check(str,
                "^\\s*\\:\\-\\s*fixedseed\\b\\s*\\?$")) {
            content = "from(Progol):-fixedseed(yes).\n";
            appendFile.appendFile(commonFile, content);
        }
        // Match settings with :- constraint and commutative.
        else if (check
```

```
                .check(str,

"^\\s*\\:\\-\\s*[constraint|commutative]+\\s*\\(\\s*[_a-z0-9\\/\\s*]+\\)\\s*\\?\\s*$")) {
            String[] dataArr = str
                    .split("\\s*\\:\\-\\s*|\\s*\\(\\s*|\\s*\\)\\s*\\?\\s*");
            content = "from(Progol):-parameter(" + dataArr[1]
                    + "),value(" + dataArr[2] + ").\n";
            appendFile.appendFile(commonFile, content);
        }
        // match modeh
        else if (check
                .check(str,

"^\\s*:-\\s*modeh\\s*\\(\\s*([\\-0-9\\*]*\\s*\\,)+\\s*(not)*\\s*[_a-z]+\\s*\\(\\s*[\\s\\+\\#\\-\\(\\)\\[\\]\\|\\=_a-zA-Z0-
9]+\\s*(\\,\\s*[\\s\\#\\+\\-\\(\\)\\[\\]\\|\\=_a-zA-Z0-9]*)*\\s*\\)\\s*\\)\\s*\\?\\s*$")) {
            String[] dataArr4 = str.split(",", 2);
            String[] dataArr5 = dataArr4[0].split("\\(");
            String[] dataArr6 = dataArr5[0].split("-");
            content = "from(Progol):-head_name(" + dataArr6[1]
                    + "),";
            content = content + "head_int(" + dataArr5[1] + "),";
            if (check
                    .check(dataArr4[1],

"^\\s*not\\s*[_a-z]+\\s*\\(\\s*[\\s\\+\\#\\-\\(\\)\\[\\]\\|\\=_a-zA-Z0-9]+\\s*(\\,\\s*[\\s\\#\\+\\-\\(\\)\\[\\]\\|\\=_a-zA-Z0
-9]*)*\\s*\\)\\s*\\)\\s*\\?\\s*$")) {
                content = content + "not(yes),arity($),";
                String[] dataArr7 = dataArr4[1].split("[\\(\\)]");
                String[] dataArr8 = dataArr7[0].split(" ");
                content = content + "head_predict("
                        + dataArr8[dataArr8.length - 1] + ")";
                content = content + ",head_predict_name("
                        + dataArr7[1] + ")";
            } else {
                content = content + "not(no),arity($),";
                String[] dataArr7 = dataArr4[1].split("[\\(\\)]");
                content = content + "head_predict(" + dataArr7[0]
                        + ")";
                content = content + ",head_predict_name("
                        + dataArr7[1] + ")";
            }
            content = content + ".\n";
            appendFile.appendFile(commonFile, content);
        }
```

```
                    // Match modeb
                    else if (check
                            .check(str,

        "^\\s*:-\\s*modeb\\s*\\(\\s*([\\-0-9\\*]*\\s*\\,)+\\s*(not)*\\s*[_a-z]*\\s*\\(\\s*[\\s\\+\\#\\-\\(\\)\\[\\]\\|\\=_a-zA-Z0-
9]+\\s*(\\,\\s*[\\s\\#\\+\\-\\(\\)\\[\\]\\|\\=_a-zA-Z0-9]*)*\\s*\\)\\s*\\)\\s*\\?\\s*$")) {
                        String[] dataArr4 = str.split(",", 2);
                        String[] dataArr5 = dataArr4[0].split("\\(");
                        String[] dataArr6 = dataArr5[0].split("-");
                        content = "from(Progol):-body_name(" + dataArr6[1]
                                + "),";
                        content = content + "body_int(" + dataArr5[1] + "),";
                        if (check
                                .check(dataArr4[1],

        "^\\s*not\\s*[_a-z]*\\s*\\(\\s*[\\s\\+\\#\\-\\(\\)\\[\\]\\|\\=_a-zA-Z0-9]+\\s*(\\,\\s*[\\s\\#\\+\\-\\(\\)\\[\\]\\|\\=_a-zA-Z0
-9]*)*\\s*\\)\\s*\\)\\s*\\?\\s*$")) {
                            content = content + "not(yes),arity($),";
                            String[] dataArr7 = dataArr4[1].split("[\\(\\)]");
                            String[] dataArr8 = dataArr7[0].split(" ");
                            content = content + "body_predict("
                                    + dataArr8[dataArr8.length - 1] + ")";
                            content = content + ",body_predict_name("
                                    + dataArr7[1] + "),repetition($),cwa($)";
                        } else {
                            content = content + "not(no),arity($),";
                            String[] dataArr7 = dataArr4[1].split("[\\(\\)]");
                            content = content + "body_predict(" + dataArr7[0]
                                    + ")";
                            if (check.check(dataArr4[1], ".*[\\(\\)].*")) {
                                content = content + ",body_predict_name("
                                        + dataArr7[1]
                                        + "),repetition($),cwa($)";
                            } else {
                                content = content
                                        + ",body_predict_name($),repetition($),cwa($)";
                            }
                        }
                        content = content + ".\n";
                        appendFile.appendFile(commonFile, content);
                    }
                    // Match Integrity Constraint
                    else if (check.check(str,
                            "^\\s*(false\\b)*\\:\\-.*[A-Z]+.*\\.$")
```

```
                    && !check.check(str, ".*\\?.*")) {
                content = "from(Progol):-integrity{" + str + "}.\n";
                appendFile.appendFile(commonFile, content);
            }
            // Match rules
            else if (check
                    .check(str,

".*[_a-z0-9]+(\\([\\'\\,\\;\\[\\|\\]\\(\\)\\$_a-z0-9A-Z]*\\))?\\s*\\:\\-\\s*[\\'\\,\\;\\[\\|\\]\\(\\)\\$_a-z0-9A-Z]+.*$")) {
                content = "from(Progol):-rule{" + str + "}.\n";
                appendFile.appendFile(commonFile, content);
            }
            // Match rules
            else if (check
                    .check(str,
                            "\\s*[_a-z0-9]+\\s*\\(\\s*[\\[\\]\\(\\)_A-Za-z0-9\\,]+\\s*\\)\\s*\\.\\s*$")
                    && check.check(str, ".*[A-Z]+.*")) {
                content = "from(Progol):-rule{" + str + "}.\n";
                appendFile.appendFile(commonFile, content);
            }
            // Match fact
            else if (check
                    .check(str,
                            "^\\s*[_a-z0-9]+\\s*(\\(\\s*[\\-_a-z0-9]+.*\\))?\\.\\s*$")
                    && !check.check(str, ".*\\?.*")
                    && !check.check(str, ".*mode[h|b].*")) {
                if (check.check(str, "^\\s*set\\b.*")) {
                    content = "from(Progol):-other{" + str + "}.\n";
                } else if (check.check(str, ".*[\\:\\-]+.*")) {
                    content = "from(Progol):-other{" + str + "}.\n";
                } else if (check.check(str, ".*\\)[_a-z0-9]+.*")) {
                    content = "from(Progol):-other{" + str + "}.\n";
                } else if (check.check(str, ".*[A-Z]+.*")) {
                    content = "from(Progol):-other{" + str + "}.\n";
                } else if (check.check(str,
                        ".*[_0-9a-z]\\s+[_0-9a-z].*")) {
                    content = "from(Progol):-other{" + str + "}.\n";
                } else if (check.check(str,
                        "^\\s*[_a-z0-9]+(\\,[_a-z0-9]+)+\\)\\.$")) {
                    content = "from(Progol):-other{" + str + "}.\n";
                } else if (check.check(str, "^\\s*[_a-z0-9]+\\.$")) {
                    content = "from(Progol):-fact{" + str + "}.\n";
                } else {
                    content = "from(Progol):-fact{" + str + "}.\n";
```

```
                        }
                        appendFile.appendFile(commonFile, content);
                    }
                    // Match rules
                    else if (check
                            .check(str,
                                    "^\\s*[_a-z0-9]+\\s*\\(\\s*[\\,\\[\\]\\_a-z0-9A-Z]+\\s*\\)\\s*\\.\\s*$")) {
                        content = "from(Progol):-rule{" + str + "}.\n";
                        appendFile.appendFile(commonFile, content);
                    }
                    // Match Negative Example
                    else if (check.check(str, "^\\s*\\:\\-.*\\.$")
                            && !check.check(str, ".*\\?.*")
                            && !check.check(str, ".*mode[h|b].*")) {
                        content = "from(Progol):-negative{" + str + "}.\n";
                        appendFile.appendFile(commonFile, content);
                    } else {
                        content = "from(Progol):-other{" + str + "}.";
                        appendFile.appendFile(commonFile, content + "\n");
                    }
                } else {
                    content = "from(Progol):-other{" + str + "}.";
                    appendFile.appendFile(commonFile, content + "\n");
                }
            }
            inProgol.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

The Java Code here is for calling ACE systems without stuck

```
    public void runaceCommand(String command, File result) {
        String result1 = result + "";
        AppendFiles af = new AppendFiles();
        try {
            ProcessBuilder pb = new ProcessBuilder(command);
            pb.redirectErrorStream(true);
            Process p = pb.start();
            p.getOutputStream().close();
            BufferedReader br = new BufferedReader(new InputStreamReader(
                    p.getInputStream()));
            String tmp = null;
            while ((tmp = br.readLine()) != null) {
```

```
                af.appendFile(result1, tmp + "\n");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

The following Code is JavaScript that is used to control Top Tag

```
// JavaScript Document
// swith tag
function switchTag(tag,content)
{
    for(i=1; i <=6; i++)
    {
        if ("tag"+i==tag)
        {
            document.getElementById(tag).getElementsByTagName("a")[0].className="selectli"+i;

    document.getElementById(tag).getElementsByTagName("a")[0].getElementsByTagName("span")[0].className="selectspan"+i;
        }else{
            document.getElementById("tag"+i).getElementsByTagName("a")[0].className="";

    document.getElementById("tag"+i).getElementsByTagName("a")[0].getElementsByTagName("span")[0].className="";
        }
        if ("content"+i==content)
        {
            document.getElementById(content).className="";
        }else{
            document.getElementById("content"+i).className="hidecontent";
        }
        document.getElementById("content").className=content;
    }
}
```

The following code is shell that is used to run 1BC

```
#run_1bc.sh
#!/bin/bash
cd /usr/apache-tomcat-7.0.19/webapps/KnowledgeWorkbench/files/temp1bc/
while getopts "a:b:c:d:" arg
do
        case $arg in
             a)
```

```
                ;;
            b)
                ;;
            c)
                ;;
            d)
                ;;
            ?)
            echo "unkonw argument"
        exit 1
        ;;
        esac
done
/root/1BCs/1BC -r $2 $4 $6 $8
```

The following code is in the XML, which show the index page, the error page when 404, 500 and java.lang.NullException encountered.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <display-name></display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <error-page>
        <error-code>404</error-code>
        <location>/erroroperation.jsp</location>
    </error-page>
    <error-page>
        <error-code>500</error-code>
        <location>/erroroperation.jsp</location>
    </error-page>
    <error-page>
        <exception-type>java.lang.NullException</exception-type>
        <location>/erroroperation.jsp</location>
    </error-page>
</web-app>
```