Inductive Logic Programming in Clementine

Sam Brewer[1] and Tom Khabaza[2]

Advanced Data Mining Group, SPSS (UK) Ltd
1st Floor, St. Andrew's House, West Street
Woking, Surrey GU21 1EB, UK
[1] sbrewer@spss.com / [2] tomk@spss.com

Abstract. This paper describes the integration of ILP with Clementine. Background on ILP and Clementine is provided, with a description of Clementine's target users. The benefits of ILP to data mining are outlined, and ILP is compared with pre-existing data mining algorithms. Issues of integration between ILP and Clementine are discussed. The implementation is then described, showing how the key issues are addressed, and describing in brief the Clementine mechanisms used to integrate ILP.

1. Introduction

The Clementine data mining system offers a range of components for the requirements of a typical data mining project. These components can be combined in various ways, ranging from the integration of models with visualisation tools, through various combinations of modelling methods, to meta-modelling, the technique of generating models to describe and analyse other models.

Clementine also has a facility called the CEMI (Clementine External Module Interface) which allows executable programs, external to Clementine, to be integrated into the software. This allows the Clementine user to create a customised the system and incorporate techniques not otherwise available.

ILP (Inductive Logic Programming) is a modelling technique which has many valuable features for data mining. ILP generates rules from multiple tables and from tables with multi-row examples, and can directly incorporate background knowledge into the modelling process; this makes it ideal for applications involving complex data, such as that used to describe molecules or electronic circuits.

The incorporation of ILP into Clementine increases the range of tools available and the range of data mining projects accessible to Clementine users.

2. Background

2.1 ILP

ILP is a form of modelling algorithm based on logic programming. Logic programs are rule-based programs written in first-order logic; inductive logic programming (ILP) integrates rule induction with logic programming.

A typical rule from a logic program might be:

```
will_buy(CustID, insurance) :-
    has_bought(CustID, Prod2), price(Prod2, Price),
    Price > 5000.
```

ILP would allow the induction of such a rule from data were terms such `has_bought(CustID,Prod)`, `will_buy(CustID,Prod)`, and `price(Prod,Price)` represent tables of data.

There are many varieties of ILP engine, for example:

D.A. Zighed, J. Komorowski, and J. Zytkow (Eds.): PKDD 2000, LNAI 1910, pp. 337−344, 2000.

- Tilde builds decision trees to predict the value of a target attribute;
- Warmr produces association rules;
- Progol produces a form of predictive rule not commonly encountered in data mining, but sometimes referred to as “compression rules”.

2.2 Clementine

Clementine is a powerful and popular data mining system. Its range of techniques, from data preparation and visualisation tools through to modelling and reporting, lends itself to a wide variety of data mining applications. Many industrial sectors benefit from the use of Clementine, including finance, retail, telecoms, manufacturing, engineering and science. Clementine appeals to users from across the spectrum of data mining expertise, ranging from non-technical business analysts to data mining and machine learning experts.

Clementine's “click-and-go” user interface provides a framework within which these two extremes of the user community can be accommodated. Each tool is configured with the most widely applicable default values in order to minimise knowledge required to operate them, whilst also offering options to alter these settings according to knowledge and expertise.

For example, consider a simple modelling stream (see figure 1) containing a source node, a type node and a neural net training node. The most basic set of operations involves setting a file to be read in through the source node, using the type node to select which field is to be modeled and executing the stream. (Note that the neural net training node is used in its default state; no modelling knowledge is required.) A more expert user has the option of editing the neural net training node and changing some aspects of its configuration.

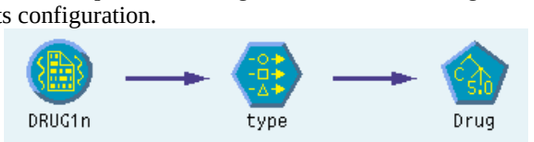

Figure 1: A Simple Clementine Modelling Stream Diagram

3. ILP Benefits to Data Mining

The strengths of Inductive Logic Programming make it suitable for applications outside the scope of current algorithms. Previous modelling techniques handle data from a single table, where one row represents one example. ILP can build models from tables containing multi-row examples and from multiple tables. Current algorithms may implicitly use extra information derived through exploration and transformation of the data; ILP can enrich the modelling process by explicitly incorporating background knowledge. This section shows that ILP has a broad potential within data mining, facilitating the data mining process for many applications not easily covered by current modelling methods.

3.1 ILP Can Extract Information From Multiple Tables

Prior to ILP, most modelling techniques handle data from a single table. If the data is stored over multiple tables, or a table consists of multi-row examples, there is a pre-processing overhead in converting the data into a single-table, single-row-example

format. Apart from the processing overhead and user time involved, re-shaping the table structure may lose potentially valuable information. ILP is able to handle both multi-row examples and multiple tables; re-structuring for these purposes is not required so the potential loss of information through pre-processing is reduced.

3.2 ILP Makes Explicit Use of Background Knowledge

It is commonplace in data mining for the data miner to derive new attributes describing significant relationships within the data; this takes place during the exploration and pre-processing phases of the data mining project. The new attributes are included as inputs to modelling, and their inclusion provides a form of "background knowledge" to the modelling process. However, these derived attributes are treated in the same way as the original attributes in the dataset; the data has been manipulated to have an influence on the modelling process.

By comparison, ILP makes use of "explicit" background knowledge; this information is incorporated directly into the modelling process. It takes the form of tables of data, and of rule-based knowledge expressed as logic programs, describing links and relationships amongst the tables,.

This means that "background" information, which might previously have been discarded because of the difficulty of incorporating it into the modelling process, can be included directly under ILP.

The capability of using explicit rule-based background knowledge also means that rules discovered by ILP can be used as background knowledge later in the data mining process.

3.3 ILP Allows Rapid Prototyping of Custom Algorithms

The use of logic programs for background knowledge also enables "rapid prototyping" of novel modelling algorithms, because the type of modelling provided by ILP is partly determined by the background knowledge. This will be a very valuable feature in situations where the patterns in the data are beyond the scope of available algorithms and modelling paradigms, because it allows the data miner to try "custom algorithms" on an ad-hoc basis for quick assessment of their suitability for a problem. The bespoke algorithm would be a combination of the ILP induction algorithm and the logic program expressed in the background knowledge; this method is made possible because the ILP engines are programmable learning systems.

4. ILP Integration with Clementine

The integration of Inductive Logic Programming with Clementine takes place through the Clementine mechanism "CEMI" mechanism (Clementine External Module Interface). This tool provides an protocol to enable external programs to be "plugged in" to Clementine. A text-based specification file is used to update Clementine with the information necessary for communicating with the external executable. The ILP engines are integrated into Clementine using this device.

4.1 Guidelines for Integration

It is important that the integration of any new technique fit smoothly into the Clementine framework.

4.1.1 Integration with the Clementine stream

Clementine is a "visual" data mining tool and any integration must be consistent with the visual programming ethos. An ILP engine will be integrated as a modelling node, and used as other modelling nodes within Clementine streams. The overall design of any modelling node in a stream is to be used as follows:

1. Connect data to a modelling node;
2. Select the target field to be modelled;
3. Insert any other necessary information;
4. Press execute;
5. A model is generated.

This is the initial framework for an ILP node. It will reside in the Models palette with other algorithms, and will be placed in a stream with a simple "click-and-drop".

There are some subtle problems for ILP. Algorithms in a stream read data from one node further upstream; the ILP node must also follow this pattern, which is in conflict with its ability to handle data from multiple sources. This problem is resolved by allowing only one explicit connection to the ILP node; further data sources are added in the settings of the node.

4.1.2 The ILP Node

The second consideration is how the user interacts within the node itself. Part of the Clementine philosophy is that any tool should be usable by a sizeable subset of the Clementine user base; at one end of the spectrum, this involves users with neither machine learning nor database experience. This means that the default use of a node must involve as few settings as possible, so that a minimal amount of technical knowledge is needed. However, below this simple level must be options for customised settings.

As with other modelling nodes in Clementine, it is necessary for the target field to be specified. This is a familiar thing for the Clementine user to have to do, so including this as a compulsory setting within the node is within our guidelines.

As only one table will be explicitly sent to the ILP node, all other tables must be listed in the settings of the node. Since there is a facility for specifying data files, logic program files may also be included. Background knowledge is included in the node as user-defined files; there is also a library containing both general-purpose and domain specific files of background knowledge.

Other settings are available within the node, but it is not compulsory for the user to alter these. There may, for example, be expert settings to tune the behaviour of the ILP engine. Any settings required by the expert user but not available in the interface can be included in a file as background knowledge.

4.2 Implementation of Integration

The physical integration of the ILP node is achieved through the Clementine External Module Interface. Information such as node type (process, terminal, model) is defined here, as is the edit dialogue. All of the information contained in the ILP node is passed through the CEMI as command line arguments to the executable,

which is run externally to Clementine. Output resulting from the execution of the ILP engine will be sent to a file; the location of this file is specified within the CEMI, allowing Clementine to locate and display the results via a model browser. For example, included in the command line arguments are the name of the target predicate, the pathname of the main Clementine table, all other file pathnames and any other settings.

4.2.1 Integration of Clementine Data with the ILP Engines

In order for the ILP engine to process the data tables, they must be converted into Prolog. In logic terms, a table is simply a list of facts describing a predicate's extension. Files containing Prolog background knowledge are left unchanged.

4.2.2 Configuring ILP

An ILP engine requires configuration for each learning task: this specifies what data and background knowledge is available and how it should be used. This is information of a highly technical nature, so Clementine's ILP node generates this automatically.

The generated configuration information contains "mode declarations" and type descriptions (all expressed in Prolog). Mode declarations are used by ILP engines to guide the modelling (rule-induction) process. A "head" mode declaration states the target to be modeled, and "body" mode declarations describe other attributes and relationships in the data, and the way these will be used in the rules to be constructed.

Mode declarations can be used in a variety of ways which have a direct effect on the outcome of the modelling process, and only a subset of these are supported by Clementine's ILP node. It is therefore important for the expert user to be able to override this automatic generation and apply alternative mode declarations as part of the background knowledge. Clementine also provides an intermediate level of automation, where the user can choose whether to generate mode declarations which describe whole records or individual fields.

Type definitions are also generated automatically, using the heuristic that fields of the same name are assumed to be of the same type. This (theoretically baseless) assumption greatly simplifies the user interface to ILP.

The configuration contains type descriptions, mode declarations and attribute predicates (depending upon user options).

5. Overview of Clementine ILP Nodes

Three ILP engines will be incorporated into Clementine: Tilde, Warmr and Progol. These engines are very different and demonstrate that ILP is not an algorithm, but a family of algorithms.

Tilde (Top-down Induction of Logical DEcision Trees) is based on the C4.5 algorithm; it uses the decision tree method to generate clauses which describe the set of examples given. The rules produced are similar in structure and are therefore quite familiar to the Clementine user.

Warmr is based on the Apriori association algorithm, and builds such rules from multiple tables. In comparison to Tilde which performs "predictive" learning, the output from Warmr is more "descriptive".

Progol is not based on any immediately recognisable algorithms; it produces "compression rules" by generalising from Prolog examples, which are effectively

records from a table of data, which have been translated into Prolog facts. Even simple runs of the Progol node have shown that it finds rules which C5.0 needs additional (exploratory-derived) attributes to find.

5.1 The Progol Node

The Progol node edit dialogue is typical of the nodes for integrated ILP engines; this is shown in figure 2.

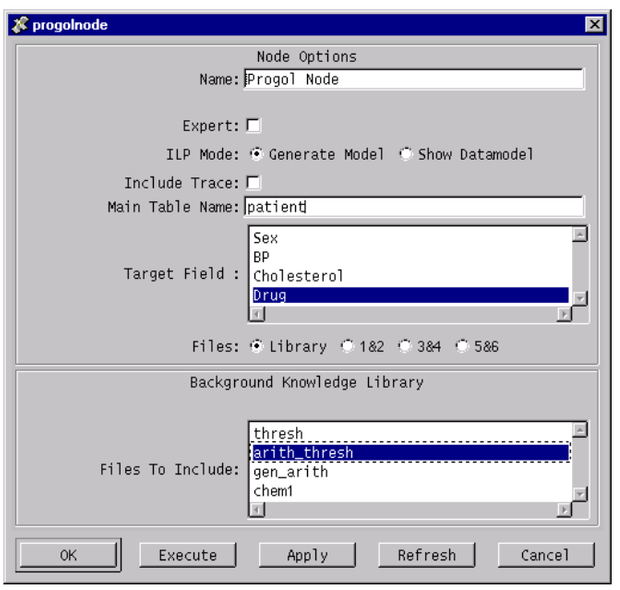

Figure 2: Progol Node Edit Dialogue

5.2 Options

Name This is the name given to the generated model.

Expert This is switched off by default; it reveals an extra dialogue containing some expert settings.

ILP Mode This gives the option of generating a Progol model (the default) or generating a datamodel; this is the automatically generated file containing mode declarations and types. It may be useful for the user to view this before executing an ILP model.

Main Table Name
This is procedure name given to the prolog facts within the table containing the target attribute.

Target Field This is where the user specifies the target attribute. A list of fields in the main table will appear when the node is connected; the user only has to highlight one of them.

The last pane defaults to the background library list. This can be changed to show another pane which provides a "set file" button; here the other data files and background knowledge files can be listed.

5.3 A Simple Modelling Stream Using the Integrated Progol Node

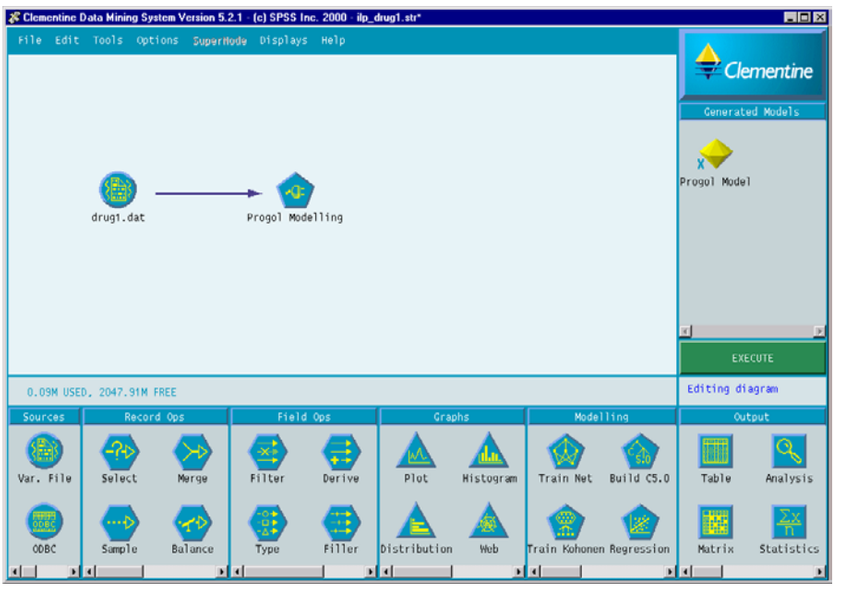

Figure 3: Clementine User Interface and ILP Node

Model Execution

After the stream has been executed, the rules generated from the ILP node form a model, which appears as a new "gem" icon on the "Generated Models" palette. When this new node is included in stream execution, data passing through the node is analysed according to the rules and a new field is created, containing predicted values. If the generated model contains rules with different predicted values which match the same record, current versions use a "first hit" approach to conflict resolution.

ILP Made Easy!

The following is a list of "click-steps" which the user has undergone in figure 3 to generate an ILP model; this demonstrates that we have achieved the design goals of incorporating ILP as a modelling algorithm in a Clementine stream (section 4.1.1.)

1. Connect data to a Progol node;
2. Select the target field to be modelled;
3. List any other files to be included;
4. Press execute;
5. A model is generated.

6. Summary

We have seen that ILP has capabilities beyond pre-exising algorithms, and that it is beneficial to have it integrated into a data mining system. ILP will enable Clementine to deal with complex data without incurring large pre-processing overheads.

ILP's ability to model data from multiple tables and with multi-record examples means that during the modelling phase, complex data can retain its structure, which may provide information. Background knowledge can also be incorporated into the modelling process, without the need for information to be added to the data itself through transformation.

This facility could be used in a more "foreground" context; the purpose of background knowledge could to alter the algorithm. This would lend ILP to rapid prototyping of bespoke algorithms.

The integration of ILP into Clementine presents some complex implementation issues, its representation in the user interface being the most obvious, as the stream construction used for Clementine nodes does not provide a perfect match with ILP. There are also complex low-level issues involved in integration. To cater for the non-technical user, the ILP node must be simplified so that various ILP components such as the mode declarations and type definitions are generated automatically. This approach provides a subset of ILP for the non-technical user. However, Clementine's integration of ILP also caters for the knowledgeable user, by providing advanced options. The range of possible applications is increased with these facilities.

The benefits of combining Clementine with ILP are twofold; firstly it will provide a set of techniques which considerably extend the capability of the software. The pre-processing of complex data and the risk of losing potentially valuable data involved, will be removed. This will facilitate data mining for such applications. Secondly, it will bring ILP to a wider and more commercial audience.

7 Acknowledgements

This work was supported by ESPRIT project number 28623 "ALADIN", which is a collaboration between Perot Systems Netherlands, British Telecom, the University of York, the Katholieke Universiteit Leuven and SPSS / Integral Solutions Ltd.

8 References

[1] H. Blockeel & L. De Raedt, Tilde and Warmr User Manual, Version 2.0, Katholieke Universiteit Leuven, April 1999.
[2] S. Brewer & T. Khabaza. Guidelines for ILP in Data Mining, Version 1.5, ALADIN Project Internal Report, November 1999.
[3] L. Dehaspe, WARMR The Frequent Query Discovery Engine User's Guide 2.1, October 1998.
[4] T. Khabaza, Note on the Integration of Inductive Logic Programming with the Clementine Data Mining System, ALADIN Project Internal Report, June 1998.
[5] S. Muggleton & J. Firth, CProgol4.4: Theory and Use, University of York, June 1998.
[6] Integral Solutions Ltd, "Introduction to the External Module Interface", Clementine Reference Manual, version 5, September 1998.
[7] S. Muggleton, "Inverse Entailment and Progol", New Generation Computing, 13:245-286, 1995.
[8] H. Blockeel and L. DeRaedt, "Top-down Induction of first order Logical Decision Trees", Artificial Intelligence 101 (1-2), 1998.
[9] L. Dehaspe and H. Toivonen, Frequent query discovery: a unifying ILP approach to association rule mining. Technical Report CW-258, Department of Computer Science, Katholieke Universiteit Leuven, March 1998. http://www.cs.kuleuven.ac.be/publicaties/-rapporten/CW1998.html