IndLog — Induction in Logic

Rui Camacho

LIACC, Rua do Campo Alegre, 823, 4150 Porto, Portugal
FEUP, Rua Dr Roberto Frias, 4200-465 Porto, Portugal
rcamacho@fe.up.pt
http://www.fe.up.pt/∼rcamacho

Abstract. IndLog is a general purpose Prolog-based Inductive Logic Programming (ILP) system. It is theoretically based on the Mode Directed Inverse Entailment and has several distinguishing features that makes it adequate for a wide range of applications. To search efficiently through large hypothesis spaces, IndLog uses original features like *lazy evaluation* of examples and Language Level Search. IndLog is applicable in numerical domains using the *lazy evaluation* of literals technique and Model Validation and Model Selection statistical-based techniques.

IndLog has a MPI/LAM interface that enables its use in parallel or distributed environments, essential for Multi-relational Data Mining applications. Parallelism may be used in three flavours: splitting of the data among the computation nodes; parallelising the search through the hypothesis space and; using the different computation nodes to do theory-level search.

IndLog has been applied successfully to major ILP literature datasets from the Life Sciences, Engineering, Reverse Engineering, Economics, Time-Series modelling to name a few.

keywords: Inductive Logic Programming

1 Introduction

The objective of an ILP system is the induction of logic programs. As input an ILP system receives a set of examples ($E = E^+ \cup E^-$) of the concept to learn (divided in positive, E^+, and negative examples, E^-), and sometimes some prior knowledge (or *background knowledge*, B). Both examples and background knowledge are usually represented as arbitrary definite logic programs. An ILP system attempts to produce a logic program (H - set of hypotheses) where positive examples succeed and the negative examples fail.

The problem of ILP is to find a consistent and complete theory, ie a set of hypotheses that “explain” all given positive examples and is consistent with the given negative examples. An ILP system performs a search through the permitted *hypotheses space* to find a set with the desired properties.

The hypotheses generated during the search are evaluated to determine their quality. Coverage is quite often used to estimate the quality of an hypothesis.

The *coverage* of an hypothesis h is the number of positive (*positive cover*) and negative examples (*negative cover*) derivable from $B \wedge h$. The time needed to compute the coverage of an hypothesis depends primarily on the cardinality of E and on the theorem proving effort required to evaluate each example using the background knowledge.

2 The *IndLog* system

IndLog [4] is an empirical ILP system written in Yap Prolog [8]. In the line of MIS, *IndLog* traverses the generalisation lattice in a top-down fashion. However, *IndLog* improves on both MIS and FOIL by using available or user supplied knowledge to traverse the generalisation lattice efficiently. *IndLog* differs from MIS and FOIL by explicitly generating the bottom of the generalisation lattice. This technique of building an initial clause to reduce the search space is characteristic of the technique of Mode Directed Inverse Entailment. The use of the bottom clause of the lattice together with further uses of knowledge either provided by the user or deduced from the available data leads to major efficiency improvements.

IndLog can handle non-ground background knowledge, can use nondeterminate predicates, uses a strong typed language and makes use of explicit bias declarations such as mode, type and determination declarations.

IndLog differs from other ILP systems, like Progol or Aleph, in the use of the Incremental Language Level Search strategy [2] and in a special feature to handle large datasets called *lazy evaluation* of examples. Lazy evaluation of literals together with Model Validation and Model Selection techniques enable IndLog to handle properly numerical domains. An interface to MPI/LAM enabled the development of a distributed/parallel module of IndLog adequate for Multi-Relational Data Mining applications.

3 IndLog Specific Features

3.1 Lazy Evaluation of examples

Language bias may be used to avoid the generation, and therefore, the evaluation of a significant number of hypotheses. However, once an hypothesis has been generated the problem then is how to evaluate it efficiently using the available data (examples and background knowledge). IndLog uses *lazy evaluation of examples* [1] as a way to avoid unnecessary use of examples and therefore speed up the evaluation of each hypothesis. We distinguish between lazy evaluation of positive examples, lazy evaluation of negative examples and total laziness. Total laziness is based on the fact that generating hypotheses is very efficient and although we may generate more hypotheses we may still gain by the increase in speed of their evaluation process. This technique may be very useful in domains where the evaluation of each hypothesis is very time-consuming. It consists in making a lazy evaluation of negatives, and then only evaluate the positives is the hypothesis is consistent with the negatives.

3.2 Incremental Language Level Search

We define a partition of the definite clauses $\mathcal{D} = \bigcup_{i=0}^{\infty} \mathcal{L}_i$. Each subset $\mathcal{L}_i$ is called a *language level* and is defined as:
$\mathcal{L}_i = \{$clause | maximum number of occurrences of a predicate symbol in the body of clause is $i\}$
where the *level* i of a language $\mathcal{L}$ is the maximum number of occurrences of a predicate symbol in the body of the clauses belonging to the language $\mathcal{L}$.

The maximum number of occurrences of predicate symbols in the body of the clauses determines to which subset the clause belongs. The language $\mathcal{L}_0$ is composed of definite clauses with just the head literal. The language $\mathcal{L}_1$ is composed by definite clauses whose literals in the body have no repeated predicate symbols. The language $\mathcal{L}_2$ will contain clauses whose literals in the body have a maximum number of occurrences of the same predicate symbol of two.

IndLog searches one language at a time starting at language level 0 and progressing incrementally one level at a time. One very important property of the partitioning by language level is that all clauses in language $\mathcal{L}_{i+1}$ are subsumed by at least one clause in language $\mathcal{L}_i$. An advantage of the search by language levels is that the most probable sub-lattices are searched first.

3.3 Cost Search

For some applications the target predicate may be modelling a functional relation whose output value is a numerical value. Constructing the model for such function involves the minimisation of a cost function other than coverage. *IndLog* uses *lazy evaluation* of literals [5] as a basic technique to handle numerical domains. It also improves over other ILP system by means of statistical-based Model Validation and Model selection tests. These latter techniques revealed very important in noisy datasets. IndLog uses and interface to the R-project library providing to the user a large number of numerical and statistical methods to be used as ILP background knowledge.

3.4 Parallel/distributed execution

A parallel implementation of an ILP algorithm may: i) improve the quality of the solutions found by searching more space in the same time of the sequential execution and/or; process larger datasets distributing the examples among the computing nodes (loading all of then in a single node may be impossible in some cases) or; get the same solution of the sequential execution much faster.

Using a MPI/LAM interface IndLog [6] has a module for parallel or distributed execution, essential for Multi-relational Data Mining applications. In IndLog, parallelism may be used in three flavours: splitting of the data among the computation nodes; parallelising the search through the hypothesis space and; using the different computation nodes to do theory-level search, that is generating different theories in different computation nodes and choosing the best one.

4 The applications

IndLog was successfully applied to major datasets from the ILP literature. It is currently being applied to the problem of Protein Folding (predicting the secondary and tertiary structure of proteins). In the first stage of this study IndLog induces rules to predict the start and end points of an α-helix. It is being applied to two "Structure-Activity Relationship" problems: understanding of anti high blood pressure drugs and; anti malaria drugs. The parallel and distributed module is of capital importance to process very large datasets. IndLog is currently being used in the analysis of the firewall logs of a university campus. In this application approximately 50 MB of data is generated per day. It has been successfully applied to Time-Series prediction problems [7] and Reverse Engineering tasks [3]. IndLog automatically computed the thresholds of a TAR model, used in Time-Series applications.

References

1. Camacho, Rui, As lazy as it can be, 8th Proceedings of the Scandinavian Conference on AI, ed. Biornar Tessen et al. IOS press, 47-58, (2003).
2. Camacho, R, Improving the efficiency of ILP systems using an Incremental Language Level Search, Twelfth Belgian-Dutch Conference on Machine Learning (Benelearn 2002), The Netherlands, 2002.
3. Camacho, R. e Brazdil, P., Improving the robustness and encoding complexity of behavioural clones, Twelfth European Conference on Machine Learning (ECML-01), Freiburg, Germany, 2001
4. Camacho, R., Inducing Models of Human Control Skills using Machine Learning Algorithms, Dep. Electrical Engineering and Computation, Univ. Porto, (2000).
5. Srinivasan, A. and Camacho, R., Numerical Reasoning with an ILP System Capable of Lazy Evaluation and Customized Search, Journal of Logic Programming, 2-3, Vol. 40, pp 185-213, 1999.
6. Rui Camacho, "From sequential to Parallel Inductive Logic Programming" 6th International Meeting on high performance computing for computational science VECPAR 2004, Valencia, Espanha, Junho 2004
7. Alves, A., Camacho, R., Oliveira, E., "Learning Time Series Models with Inductive Logic Programming", European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems (EUNITE 2003), 10 - 12 July 2003 in Oulu, Finland
8. Costa, V. and Damas, L. and Reis, R. and Azevedo, R., YAP Prolog User's Manual, Universidade do Porto, (1989).