

Un confronto di sistemi di ILP sul dataset Sisyphus
A Comparison of ILP Systems on the Sisyphus Dataset

Fabrizio Riguzzi
Dipartimento di Ingegneria, Università di Ferrara,
Via Saragat 1, 44100 Ferrara, Italy
friguzzi@ing.unife.it

SOMMARIO/ABSTRACT

In questo articolo presentiamo un confronto tra due sistemi di Inductive Logic Programming (ILP) sul dataset Sisyphus. L'obiettivo del confronto è di indagare sul comportamento di due sistemi allo stato dell'arte dell'ILP su un dataset di "grandi" dimensioni. Il confronto mostra che le limitazioni dei sistemi di ILP riguardano principalmente il tempo di calcolo piuttosto che l'occupazione di memoria.

In this paper we present a comparison between two Inductive Logic Programming (ILP) systems on the Sisyphus dataset. The aim of this comparison is to investigate the behaviour of two state of the art ILP systems on a "large" dataset. The comparison shows that the limitations of ILP systems regards mainly the execution time rather than the memory requirements.

Keywords: Machine Learning, Inductive Logic Programming, Relational Databases.

1 Introduction

In this paper we present a comparison of two Inductive Logic Programming (ILP) systems on the Sisyphus dataset. The aim of the comparison is to show how the systems behave on a large dataset. The considered systems are Aleph and Tilde. Both systems have an unacceptable execution time on the whole dataset, so they are run over samples extracted from the dataset.

The comparison shows that, on average, Tilde finds more accurate theories in a smaller time.

The paper is organized as follows: section 2 presents the field of ILP, section 3 describes the Sisyphus dataset, in section 4 we illustrate the main feature of Aleph, in section 5 the Tilde system is presented, in section 6 we report on the performed experiments, in section 7 we discuss related works and finally in section 8 we conclude.

2 Inductive Logic Programming

ILP [5] is a research field at the intersection of Machine Learning and Logic Programming. Its objective is the solution of a learning problem where the language used for describing both the instances and the concept to be learned is logic programming. In particular, the main aim of ILP is to devise algorithms that solve the following problem:

Given:

- a set E^+ of positive examples (ground facts)
- a set E^- of negative examples (ground facts)
- a background knowledge B (logic program)
- a hypothesis space \mathcal{H} described by a language bias L

Find: a logic program $P \in \mathcal{H}$ such that

- $\forall e^+ \in E^+, B \cup P \models e^+$
- $\forall e^- \in E^-, B \cup P \not\models e^-$

Nowadays ILP is a mature field, many algorithms have been proposed and they have been successfully applied to many domains. Examples of ILP systems are FOIL [8], Progol [6], Tilde [2] and Aleph [10]. ILP techniques have been successfully applied to problems in engineering, natural language processing, environmental sciences and life sciences.

Recently, ILP has been the subject of great interest due to the fact that ILP techniques can be used for Data Mining from relational databases. In fact, a relational database can be seen as a Prolog program: each relation can be represented by a Prolog predicate and each tuple is represented by a ground Prolog fact. In this way, ILP can be used to mine knowledge from databases containing more than one relation, differently from traditional Machine Learning techniques that require the data to be stored in a single table.

3 The Sisyphus Dataset

The Sisyphus dataset was made available through the web by the insurance company Swiss Life in 1998. It was supposed to be the subject of a workshop in ECML98. The workshop was later cancelled due to too few submissions. The dataset was removed from the web a few years later and now it is not publicly available.

The Sisyphus dataset is an extract of the data warehouse of Swiss Life, and contains information regarding life insurances and pension schemes of its clients. The dataset is multi-relational and is composed of 8 tables, for a total of 336.266 tuples. The number of tuples and attributes of each relation is indicated in table 1.

Table	Tuples	Attributes
vvert	34.986	18
parrol	111.077	5
part	17.267	8
eadr	505	3
padr	17.970	4
tfkomp	73.502	26
tfrol	73.332	8
taska	17.627	2
Total	336.266	74

Table 1: Tables of the Sisyphus Dataset.

The table `part` contains the data of all the clients (partners). Tables `eadr` and `padr` describe respectively their electronic and postal addresses. Each partner has a role in one or more insurance policies (table `vvert`) that is described in the table `parrol`. An insurance contract can have many components (e.g., a component in the case in which the insured person becomes disabled). Each component (table `tfkomp`) is correlated with a record in the table `tfrol` that specifies further properties of the tariffs applied to the partner. For this dataset, a class is assigned to every partner, described in the `taska` table.

A diagram describing the schema of the database is shown in Figure 1.

The dataset is distributed in the form of Prolog facts, each record being of type: `table_name(attribute1, ..., attributen)`. Every table is contained in a different file. In order to reduce the dimension of the files, the table names have been abbreviated by using a single letter. The dimensions in bytes of the obtained files is indicated in table 2.

The dataset is interesting due to its dimension: in fact its dimension is large if compared to usual ILP datasets, that are of the order of a few hundred Kilobytes. However, it is not so large for it not to be contained in main memory of even an entry level personal computer. The time required to load the whole database (excluding table `taska`) into memory is 56.13 seconds with Sicstus Prolog 3.11.0 and 13.17 seconds with Yap Prolog 4.4.4, on a personal com-

Table	File dimension (bytes)
tfkomp	7,163,578
parrol	3,470,099
tfrol	2,852,867
vvert	2,504,030
part	512,154
padr	395,505
eadr	8,014
taska	239,004
Total	17,145,251

Table 2: Dimension of the Sisyphus dataset tables.

puter with a 1133 MHz Pentium III Mobile, 512 MB of Ram and the Windows 2000 operating system.

Apart from the dimension, the dataset is not particularly problematic for ILP systems: it is a typical example of a dataset from a financial domain, where each client is connected to the set of his/her transactions, each described by a number of attributes. The ILP system will find a theory that contains tests on the attributes comparing them with constants. The possible tests differ depending on the type of attributes: for nominal attributes only equality tests will be used, while for numerical attributes, besides equality tests, also smaller than and greater than tests will be used.

4 Aleph

Aleph [10] implements a learning algorithm similar to Progol. It is a sequential covering algorithm (also called a separate and conquer algorithm) because it learns clauses one by one and, at each step, it removes the positive examples covered by the clause. The main cycle is the following

```

function Aleph( $E^+$ : pos. ex.,  $E^-$ : neg. ex.,  $B$ : back. kn.)
 $P := \emptyset$ 
repeat /* covering cycle */
    select one positive example  $e_i^+$ 
    build most specific clause  $\perp_i$ 
     $C := \text{GenerateClause}(\perp_i, E^+, E^-, B)$ 
     $P := P \cup \{C\}$ 
    remove from  $E^+$  the positive examples covered by  $C$ 
until  $E^+ = \emptyset$ 
return  $P$ 

```

The function `GenerateClause` returns a clause that is more general than \perp_i and that covers a number of positive examples and no (or a few) negative examples.

The function `GenerateClause` searches the space of clauses top-down, i.e., it starts from the most general clause ($p(X) \leftarrow$ if p is the predicate to be learned) and gradually refines it by adding literals taken from \perp_i . The search is performed by means of a branch and bound algorithm.

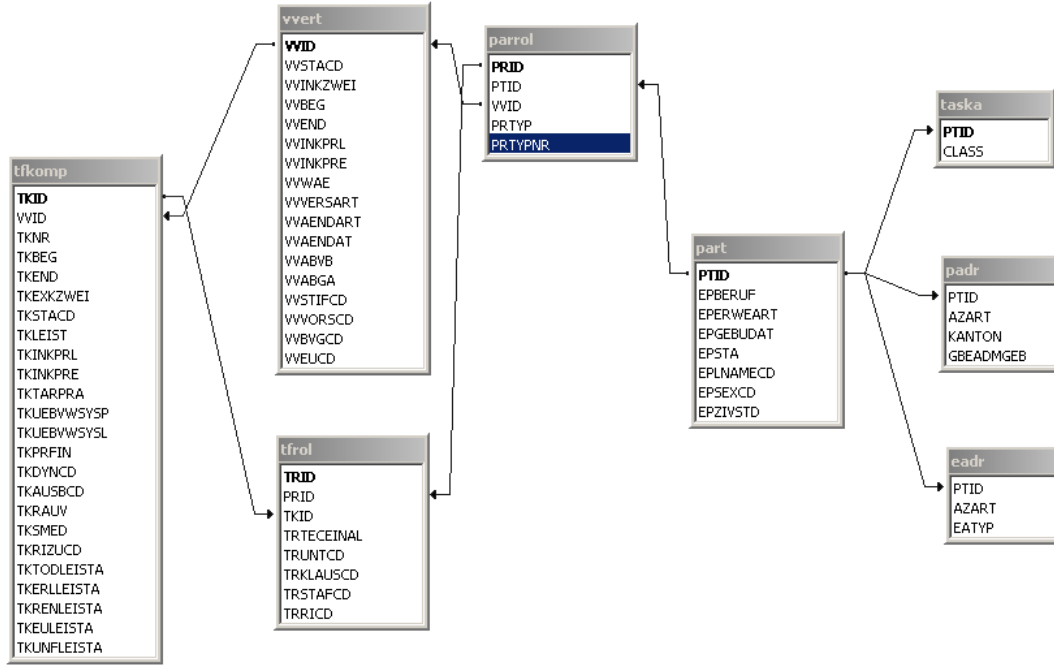


Figure 1: Database schema of the Sisyphus Dataset.

```

function GenerateClause( $\perp, E^+, E^-, B$ )
bestclause := anything; bestscore := -inf; i := 0
active := { $p(X) \leftarrow$ }
while active is not empty and i  $\leq n$ 
  /* specialization cycle */
  remove the first clause D from active
  let  $S_D = \{D_1, \dots, D_k\}$  be the set of
    specializations of D
  compute the evaluation function  $h_j$  for each
    clause in  $S_D$ 
  compute an upper bound  $u_j$  of the evaluation
    function for each clause in  $S_D$ 
  for  $j := 1$  to  $k$ 
    if  $u_j \leq \text{bestscore}$  then
      prune  $D_j$ 
    else
      if  $D_j$  is a complete solution and
         $h_j \geq \text{bestscore}$  then
          bestclause :=  $D_j$ ; bestscore :=  $h_j$ 
          prune nodes in active with upper bound
            lower than  $h_j$ 
          add  $D_j$  to active
  i := i + 1
return bestclause

```

The nodes in *active* are ordered according to a dual search key. The value of this search key depends on the settings imposed by the user for the search strategy and evaluation function. For example, with breadth first as search strategy

and coverage for the evaluation function (the default settings), the primary and secondary keys are respectively L and $P - N$, where L is the number of literals in the clause, P is the number of positive examples covered by the clause and N is the number of negative examples covered by the clause. This means that shorter clauses will be higher up in *active* and, among clauses with equal length, those with a higher difference $P - N$ will be higher up.

The specialization of a clause D are obtained by using a refinement operator. Aleph adopts as a default the refinement operator of Progol that uses the most specific clause \perp_i for selecting literals to be added to D . In this way it ensures that all the refinements will be more general than \perp_i and therefore will cover at least the positive example used to generate \perp_i .

The computation of the upper bound depends on the search strategy and evaluation function. In cases where no upper bound can easily be obtained it is taken to be $+\text{inf}$, resulting in minimal pruning.

A solution is complete when it satisfies two constraints: it has a minimum accuracy (number of positive examples covered over the total number of examples covered, 0 by default) and it covers a maximum number of negative examples (0 by default).

5 Tilde

Tilde solves a learning problem slightly different from the typical ILP problem:

Given:

- a set E of examples (ground facts for a target predicate p/n)
- an argument of p/n to be predicted (e.g. C in $p(\mathbf{X}, C)$)
- a background knowledge B (logic program)
- a hypothesis space \mathcal{H} described by a language bias L

Find: a first order logical decision tree $T \in \mathcal{H}$ such that

- T assigns to each example $p(\mathbf{x}, c)$ the class c .

A first order logical decision tree (FOLDT) is a binary decision tree in which: (1) the internal nodes of the tree contain a conjunction of literals (2) different internal nodes may share variables under the following restriction: a variable that is introduced in a node (which means it does not occur in higher nodes) must not occur in the right branch of that node.

An example of a FOLDT is:

```
machine(A, B)
worn(A, C) ?
---yes: not_replaceable(C) ?
|      ---yes: [sendback] [6.0/6.0]
|      ---no:  [fix] [6.0/6.0]
---no:  [ok] [3.0/3.0]
```

This tree can be interpreted as follows: we first test whether a machine A has a part C that is worn. If so then we test whether C is non replaceable. If so then the class of the machine (argument B) is sendback, otherwise it is fix. If A does not have a part C that is worn, that the class is ok.

We use the following notation: a FOLDT T is either a leaf of class c , in which case we write $T = leaf(c)$, or it is an internal node with conjunction $conj$, left child l and right child r , in which case we write $T = inode(conj, l, r)$.

A FOLDT can be used to classify an example $p(\mathbf{x}, C)$ in two ways: by adopting a specialized classification function that employs the tree or by first translating the FOLDT into a logic program and then running the query $p(\mathbf{x}, C)$ against it plus the background knowledge, where \mathbf{x} stands for a vector of constants.

Tilde learns FOLDT by upgrading to a first order setting the c4.5 [7] learning algorithm. Therefore it performs a simultaneous covering algorithm: it does not try to cover a number of examples in order to remove them from the training set but it tries to cover all the examples at once.

The algorithm performs a standard recursive partitioning approach and is shown below (we assume that $p(\mathbf{X}, C)$ is the target predicate):

```
function Tilde(E: set of examples)
T:=GrowTree(E, true)
return Prune(T)
```

```
function GrowTree(E: set of examples, Q: query)
Qb := OptimalSplit(ρ(Q), E)
if StopCrit(Qb, E) then
return leaf(Info(E))
else
conj := Qb - Q
E1 := {e ∈ E | B ∪ {p(X, C) ← Qb} ⊨ e}
E2 := {e ∈ E | B ∪ {p(X, C) ← Qb} ⊭ e}
T := inode(conj,
GrowTree(E1, Qb), GrowTree(E2, Q))
return T
```

The function $\rho(Q)$, given a conjunction Q , returns the set of all the specializations of Q according to the refinement operator. The function $OptimalSplit$, given a set of conjunctions and a set of example E , returns the conjunction that best discriminates the examples of the various classes. The amount of discrimination is computed by using the gain ratio heuristic function [7] of c4.5.

The function $StopCrit$ evaluates the split of the examples generated by Q_b and decides whether it is the case of stopping the tree growth. A typical case in which the growth is stopped is when one of the sets obtained from the split contains less than a predefined number of examples (2 by default).

The function $Info(E)$ returns the most common class in the set of examples E .

The function $Prune(T)$ returns the tree T after the same pruning as in c4.5 is performed.

6 Experiments

We consider a learning task that consists in predicting the class of the clients. The class is represented in table `taska` and can assume the values:

- ‘0’: not applicable;
- ‘1’: belonging to the class of interests;
- ‘2’: not belonging to the class of interests;

We are particularly interested in correctly classifying clients of class ‘1’.

The distribution of values is shown in table 3. To this

Class	Examples	%
1	10.723	62,10%
2	2.599	15,05%
0	3.945	22,85%
Total	17.267	

Table 3: Class distribution in the Sisyphus dataset.

task we applied Aleph version 5 and Tilde version 2.2. Aleph is implemented in Prolog and uses the Yap Prolog

compiler version 4.4.4. The implementation of Tilde that has been used is the one contained in the ACE suite of ILP systems version 1.2.6.

All the experiments have been performed on a personal computer with a 1133 MHz Pentium III Mobile, 512 MB of Ram and the Windows 2000 operating system.

No other transformation was necessary because the dataset is already distributed as prolog facts directly usable by Aleph and Tilde.

For Aleph, we decided to consider examples belonging to class 1 as positive examples and those belonging to classes 0 and 2 as negative, instead of opting for a separate classification of the three classes. The predicate to be learned is therefore $\text{taska}(X)$ where X is the identifier of the client. For Tilde, we learned the predicate $\text{taska}(X, C)$ where X is the identifier of the client and C is the class. C is indicated as the argument to be predicted by Tilde.

Let us now discuss the settings used for the experiments. The parameters of the two systems were chosen by running repeatedly each system on a sample of the data with different parameters and by testing the learned theory on the rest of the data. The parameters that gave the best results for each system were chosen.

For Aleph we set i to 6, minpos to 2, clauselength to 8 and noise to 2. The parameter i indicates the maximum depth of the new variables that can appear in the body of clauses. minpos sets the minimum number of positive examples that a clause can cover in order to be added to the current hypothesis. clauselength is the maximum number of literals that can be present in a clause. noise is the maximum number of negative examples that a clause can cover. All the other parameters assume their default value.

For Tilde each parameter assumes its default value, since these settings were found to yield the best results.

The language bias that is used in both systems allows the database relations to be chained according to the foreign key links represented in the database schema. Moreover, additional background predicates were used, namely the binary relations $\text{equal}(e\text{q}/2)$, smaller or equal than ($\text{smeq}/2$) and greater or equal than ($\text{greq}/2$). These predicates are intensionally defined in the background and are used in order to compare the attributes of database relations that are not keys (primary or foreign) with constants. In particular, for nominal attributes, only the predicate equal is used, while for numeric attributes (reals or integers) all the three predicates are used. The constants that can appear as the second argument of these predicates are taken from those appearing in the extensional database relations. In the case of Tilde, they are obtained by running a query on the database and collecting the results, while in the case of Aleph they are obtained by building the “bottom clause”.

For Tilde, it was necessary to specify also lookahead statements. They are used by the refinement operator in order to specialize the current node by adding a conjunc-

tion of literals instead of a single literal. In particular, with the lookahead statements we force Tilde to add, besides a database predicate, also a test on one of its arguments (equal, smaller or equal or greater or equal).

Without these lookahead statements, Tilde would not be able to learn because the addition of a database predicate alone would not produce any improvement in the gain ratio.

We tried to run the two systems over the whole dataset. Unfortunately, the training over the whole dataset was such a hard task that after 24 hours of CPU time no system had given an answer. As a consequence, we decided to extract small random samples from the whole dataset, to apply the algorithm to them and then to average the results, in order to filter away variations due to randomness.

We considered samples containing 360 partners. The number was chosen for historical reasons: we had already available from previous experiments one such sample on which the learning times were acceptable.

We have randomly selected 360 facts from the relation taska . The remaining facts were included in the testing set. The background knowledge of each sample has been obtained by including in it all the facts that were related to the chosen partners. Five samples were extracted.

The average dimension of the file containing just the examples is 5 Kilobytes. The average dimension of the file containing the background knowledge is 463 Kilobytes.

The theory learned on the reduced dataset was tested on the examples from the testing set. The background knowledge used for testing was the complete database (excluding the relation taska).

The average number of positive and negative examples of the training and testing sets is reported in table 4. The

Examples	Training Sets		Test Sets	
$ E^+ $	223.4	62%	10,499.6	62%
$ E^- $	136.6	38%	6,407.4	38%
$ E $	360		16907	

Table 4: Example distribution for the experiments.

results of experiments are compared in terms of accuracy. Such a measure is defined as the number of positive test examples covered by the theory plus the number of negative test examples not covered by the theory over the total number of test examples.

The average learning times obtained by Aleph and Tilde on the five training sets is shown in table 5 together with the average accuracies over the testing sets. In parentheses is indicated the standard deviation. The testing time was 11.60 hours on average for the theory learned by Aleph and 5.78 hours on average for the theory learned by Tilde. Testing was performed using Yap 4.4.4.

Let us also distinguish the types of errors performed by the two systems. Table 6 shows the rate of errors of commission (negative examples classified as positive over the

Algorithm	Av. Time (hours)	Av. Accuracy (%)
Aleph	9.13 (3.73)	68.92% (2.56%)
Tilde	0.96 (0.07)	86.82% (0.84%)

Table 5: Average execution time and accuracy obtained by applying Aleph and Tilde to the Sisyphus dataset (standard deviation in parenthesis).

total number of negative examples) and the rate of errors of omission (positive example classified as negative over the total number of positive examples).

Algorithm	Commission (%)	Omission (%)
Aleph	35.8%	28.2%
Tilde	20.9%	8.4%

Table 6: Average commission and omission error rates.

Some of the clauses learned by Aleph are shown in Figure 2. Part of the tree learned by Tilde in the same fold is shown in Figure 3. In order to test the theory learned by Tilde on the testing set, we used the Prolog program equivalent to the learned tree. Moreover, in order to compare the results of Tilde with those of Aleph, we added to the equivalent Prolog program the following clause

`taska(X) :- taska(X, Y), Y=1.`

In this way, we could use testing sets composed of facts for the `taska(X)` predicate.

The application of Aleph and Tilde to the Sisyphus dataset shows that, even if the dataset can fit in main memory, the execution times are too large to apply the systems to the whole dataset. This suggests that the current limitations of ILP systems regard the execution times rather than the memory space. Even with a dataset that is 2 % of the original dataset, the execution times are of the order of a few hours.

The comparison between Aleph and Tilde over the considered sample shows that Tilde is superior both in terms of the accuracy of the learned theory and of the execution times. In particular, if we compute the two series of accuracies with a t -test [3], we obtain 19.2 as the value of the t statistics, that means that the difference in accuracy is significant at more than a 99.99% level.

The superiority in terms of execution time is not surprising, since Tilde implements a “simultaneous covering” algorithm in which the conditions of multiple rules are chosen at once. Moreover, once a condition is chosen, it is never retracted. On the contrary, Aleph implements a “sequential covering” algorithm where the conditions of each rule are chosen separately. Moreover, Aleph keeps a set of clauses in its search, therefore, after the addition of a condition to a clause, it can still consider clauses without that condition.

The superiority in terms of accuracy is more surprising, especially since the two system are given the same

language bias. It can be probably explained by the way in which the two system generate the constants to be inserted in comparison operators. While Aleph uses the constants appearing in the bottom clause generated from a single example, Tilde extracts the constants directly from the database. So, while Aleph can insert in a single clause only constants relative to a single example, Tilde can insert constants appearing in more than one example.

Looking at the error rates, we can observe that both systems have higher commission error rates, thus showing that the learned theories in both cases are over-general. This is probably due to the fact that the number of negative examples in the training sets is not sufficient to accurately model the negative concept.

7 Related Works

The application of data mining techniques to the Sisyphus dataset has been the subject of [4]. This paper reports the application of a number of propositional learning system to the dataset. In order to apply propositional system to Sisyphus, the dataset has been transformed into a propositional form, i.e., into a database containing a single table where each original example is represented by a single row. The paper does not describe the details of the propositionalization performed but the standard technique consists in aggregating the attributes of the tables connected with the example table by a many to one relationship. For example, the table `tfrol` is connected by a many to one relationship to the table `parrol` that in its turn is connected by a many one relationship to the table `taska`. The attribute `TRTECEINAL` of table `tfrol` is the age at contract agreement: in the single table, for each client, the minimum, the maximum and the average of this attribute for all the tuples related to the client will be included.

In [4] the experimentation on the Sisyphus dataset was performed by first removing the examples with class 0. In this way, roughly 80% of the examples are positive and 20% are negative. Then the authors randomly split the examples in a training set containing 70% of the instances and a testing set containing 30 % of the instances. The accuracy obtained by a number of propositional learner is shown in table 7. The AllPos algorithm is an algorithm

Algorithm	Accuracy (%)
J48	89.7%
Naive Bayes	81.8%
Linear SVM	89.9%
WBC _{SVM}	80.0%
OneR	83.4%
AllPos	80.5%

Table 7: Accuracy of propositional learner on the Sisyphus dataset.

that classifies all the examples as positive.

```

taska(A) :-
    part(A,B,C,D,E,F,G,H), smeq(B,55), eq(G,2).
taska(A) :-
    parrol(B,A,C,D,E), eq(E,1), tfrol(F,B,G,H,I,J,K,L), eq(H,32).
taska(A) :-
    parrol(A,B,C,D,E,F,G,H), eq(D,1941).

```

Figure 2: Some of the rules learned by Aleph from one of the samples.

```

taska(A,B)
parrol(C,A,D,E,F),eq(E,11) ?
+--yes: parrol(G,A,H,I,J),eq(I,17) ?
|      +--yes: [2] [16.0/16.0]
|      +--no:  tfkomp(L,D,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,A1,B1,C1,D1,E1,F1,G1,H1),eq(M,6) ?
|      +--yes: padr(A,I1,J1,K1),eq(K1,89411) ?
|      |      +--yes: [1] [2.0/2.0]
|
. . . .
+--no: [0] [76.0/76.0]

```

Figure 3: Part of the tree learned by Tilde from one of the samples.

The results can be compared with ours with caution for two reasons. The first is that the authors use a dataset with a different ratio of positive and negative examples. In fact, by classifying all the examples as positive they get an accuracy of 80.5%, while we get an average accuracy of 62.1%. The second is that propositionalization requires a manual intervention in order to choose the different aggregating functions to be applied, while ILP looks for interesting features by using brute force.

[4] reports also the result of applying Tilde to the dataset with the same settings (no 0 examples, 70% training, 30% testing): the achieved accuracy is 94.7% (they do not report execution time). The difference with our results is due to two factors: the different experimental settings and the fact that we do not have used the discretization feature of Tilde. The reason why we did not use discretization is that we wanted to try the simplest approach and also that Aleph does not offer it, so the comparison would not have been totally fair.

8 Discussion and Conclusion

Sisyphus is an interesting data because of its size, that is one order of magnitude larger than that of the average ILP dataset.

To this dataset we have applied two state of the art ILP systems: Aleph and Tilde. Aleph learns logic programs, while Tilde learns first order logical decision trees.

The application of the two systems to the whole dataset was impossible: after 24 hours of CPU time none of the systems responded. Therefore we have applied the systems to samples from the dataset. We extracted five samples composed of 360 examples, we ran Aleph and Tilde on each sample and we averaged the results over the five

samples.

The results show that Tilde is both faster (around 1 hour on average against around 9 hours) and significantly more accurate (86.82% against 68.92%). The speed of Tilde is not surprising, since in building the tree it applies a greedy algorithm for choosing the conjunction that gives the optimal split: once a conjunction is selected, it is never retracted. The difference in accuracy is instead surprising given that in [1] Tilde is reported to have accuracy results that are comparable (not superior) with those of FOIL and Progol.

The experiments show that the main limitations of ILP systems regards execution times rather than memory requirements. A preliminary analysis shows that most of the time is spent checking the coverage of examples, therefore we think that methods of stochastic matching, such as those used in [9], are particularly useful and should be integrated with these ILP systems.

Another approach that could speed up Tilde (but not Aleph) is the use of its discretization algorithm that should reduce the number of constants that are tried for inclusion in conjunctions.

An interesting line of future research would be to investigate how many more examples are needed for obtaining good classifiers (i.e., with more than 90% accuracy). This requires the execution of multiple experiments with increasing number of examples. For this to be accomplished a faster machine and speeding up techniques for the two systems are needed.

REFERENCES

- [1] H. Blockeel. *Top-down Induction of First Order Logical Decision Trees*. PhD thesis, Department of Com-

- puter Science, Katholieke Universiteit Leuven, 1998.
- [2] Hendrik Blockeel and Luc De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June 1998.
 - [3] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998.
 - [4] Thomas Gärtner, Shaomin Wu, and Peter A. Flach. Data mining on the sisyphus dataset: Evaluation and integration of results. In Christophe Giraud-Carrier, Nada Lavrač, and Steve Moyle, editors, *Integrating Aspects of Data Mining, Decision Support and Meta-Learning*, pages 69–80. ECML/PKDD’01 workshop notes, September 2001.
 - [5] S. Muggleton. Inductive logic programming. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 43–62. Ohmsma, Tokyo, Japan, 1990.
 - [6] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
 - [7] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, USA, 1988.
 - [8] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
 - [9] Michele Sebag and Celine Rouveirol. Resource-bounded relational reasoning: Induction and deduction through stochastic matching. *Machine Learning*, 38(1/2):41–62, January 2000.
 - [10] Ashwin Srinivasan. Aleph, 2004. http://web.comlab.ox.ac.uk/oucl/research/areas/machine/Aleph/aleph_toc.html.

9 Contacts

Fabrizio Riguzzi
Dipartimento di Ingegneria, Università di Ferrara,
Via Saragat 1, 44100 Ferrara, Italy
++39 0532974836
friguzzi@ing.unife.it

10 Biography

He is Assistant Professor at the Dipartimento di Ingegneria, Università di Ferrara. He graduated in Computer Engineering at the University of Bologna in 1995 and attained his Ph.D. from the University of Bologna in 1999. He has been a visiting researcher at the University of Cyprus and at the New University of Lisbon. His research interests include: data mining (and in particular methods for learning from multirelational data), machine learning and software engineering. He is a member of the AI*IA.

