Learning First Order Logic Time Series Classifiers: Rules and Boosting *

Juan J. Rodríguez[1], Carlos J. Alonso[1], and Henrik Boström[2]

[1] Grupo de Sistemas Inteligentes, Departamento de Informática
Universidad de Valladolid, Spain
{juanjo,calonso}@infor.uva.es
[2] Department of Computer and System Sciences
Stockholm University/KTH, Sweden
henke@dsv.su.se

Abstract. A method for learning multivariate time series classifiers by inductive logic programming is presented. Two types of background predicate that are suited for this task are introduced: interval based predicates, such as always, and distance based, such as the euclidean distance. Special purpose techniques are presented that allow these predicates to be handled efficiently when performing top-down induction. Furthermore, by employing boosting, the accuracy of the resulting classifiers can be improved significantly. Experiments on several different datasets show that the proposed method is highly competitive with previous approaches.

1 Introduction

Multivariate time series classification is useful in domains such as biomedical signals [13], continuous systems diagnosis [2] and data mining in temporal databases [6]. This problem can be tackled by extracting features of the series through some kind of preprocessing, and using some conventional machine learning method. However, this approach has several drawbacks [12]: the preprocessing techniques are usually *ad hoc* and domain specific, there are several heuristics applicable to temporal domains that are difficult to capture by a preprocess and the descriptions obtained using these features can be hard to understand. The design of specific machine learning methods for the induction of time series classifiers allows the construction of more comprehensible classifiers in a more efficient way.

When learning multivariate time series classifiers, the input consists of a set of training examples and associated class labels, where each example consists of one or more time series. The series are attributes of the examples, and are often referred to as variables, since they vary over time.

The method for learning time series classifiers that we propose in this work is based on *inductive logic programming* (ILP) and utilises two types of background predicate: i) interval based predicates, such as always(Example, Variable, Region,

* This work has been supported by the Spanish CYCIT project TAP 99–0344.

D.A. Zighed, J. Komorowski, and J. Żytkow (Eds.): PKDD 2000, LNAI 1910, pp. 299–308, 2000.

Beginning, End), where Region is an interval in the domain of the variable, and ii) distance based predicates, such as euclidean_le(Example, Reference, Variable, Value), which considers the euclidean distance between two series.

The proposed system is neither a generic ILP system, nor is it an extension of one of the ILP systems available. In our experience, it is not enough to simply give predicates of the above types as background knowledge to an existing system, but it is also necessary to incorporate knowledge about how to process them efficiently. Hence, specific methods have been developed that allow for an efficient search for hypotheses that include these types of literals.

Moreover, we have also incorporated boosting, which is a method for generating ensembles of classifiers that has been demonstrated to improve accuracy significantly. In our case, the individual classifiers are composed only of rules with one literal in the body. When tested on several previously proposed datasets, the new method achieves better results than all previously published results on these datasets.

The rest of the paper is organized as follows. Section 2 describes the special purpose background predicates for time series classifiers. The proposed method is presented in section 3, including techniques for efficiently handling the special purpose predicates. Section 4 presents experimental results when using the new method. In section 5, it is demonstrated that these results can be improved substantially by employing boosting. Finally, we give some concluding remarks in section 6.

2 Temporal Predicates

2.1 Interval Predicates

The interval predicates make use of *regions*, which are intervals in the domain of the variable. The regions used are independent of the time, as is usual practice when working with series. The interval predicates are the following:

- always(Example, Variable, Region, Beginning, End). It is true, for the Example, if the Variable is always in this Region in the interval between Beginning and End.
- sometime(Example, Variable, Region, Beginning, End).
- true_percentage(Example, Variable, Region, Beginning, End, Percentage). It is true, for the Example, if the percentage of the time between Beginning and End where the variable is in Region is greater or equal to Percentage.

Once that it has been decided to work with temporal intervals, the use of the predicates always and sometime seems natural. Since the former predicate might be too demanding and the latter too flexible, a third one has been introduced, true_percentage.

2.2 Distance Predicates

Several machine learning methods, such as instance-based learning, are based on the use of similarity functions to measure distances between examples. The framework of ILP allows us to include several such definitions in the background knowledge. In our case, we use predicates on the following form:

<*distance*>_le(Example, Reference, Variable, Value)

which is true if the <*distance*>, for one Variable of the examples, between the Example considered and another Reference example is less or equal (_le) than Value. The predicate euclidean_le uses the *euclidean* distance. It is defined, for two univariate series s and t as: $\sqrt{\sum_{i=1}^{n}(s_i - t_i)^2}$. Its execution time is $O(n)$.

Dynamic Time Warping (DTW) aligns a time series to another reference series in a way such that a distance function is minimized, using a dynamic programming algorithm [6]. If the two series have n points, the execution time is $O(n^2)$. The predicate dtw_le uses the minimized value obtained from the DTW as a similarity function between the two series.

3 Top-Down Induction of Time Series Classifiers

The proposed technique follows the scheme of the top-down methods in ILP [5]. The particularities arise in the selection of literals. The selection mechanisms suggested in this section could be incorporated in other ILP systems as well by using the generic method for numerical reasoning described in [18].

Obtaining Regions. In some cases, the definitions of the regions for the interval literals can be obtained from an expert. Otherwise, they can be obtained with a discretization preprocess, which obtains r disjoint, consecutive intervals. The regions considered are these r intervals (equality tests) and others formed by the union of the intervals $1 \ldots i$ (less or equal tests), as suggested by [8].

Selection of Interval Literals. Given an overly general clause that covers both positive and negative examples, the best literal to add to the body must be selected, according to some criterion. Then it is necessary to search over the space of literals. The possible number of intervals, if each series has n points, is $(n^2 - n)/2$. With the objective of reducing the search space, not all the intervals are explored. Only those that are of size power of 2 are considered. The number of these intervals is of $\sum_{i=1}^{k}(n - 2^{i-1}) = kn - 2^k - 1$ where $k = \lfloor \lg n \rfloor$.

If p is the number of predicates considered, and v the total number of regions in the different variables, the possible number of atoms is $pvn \lg n$. In the case of predicates with additional arguments (true_percentage), it is also necessary to consider how many values are possible for them, but its number is a constant. Using a dynamic programming algorithm, the information that needs to be obtained from a window of size $2i$ is computed from two consecutive intervals of size i, with a time of $O(e)$, where e is the number of examples. Hence, the selection of the best literal requires a time of $O(epvn \lg n)$.

```
class( Example, cylinder ) :- % 213, 426
   not true_percentage( Example, x, 1_4, 22, 86, 50 ), % 193, 18
   not true_percentage( Example, x, 3, 78, 110, 30 ), % 182, 0
   !.
class( Example, bell ) :- % 213, 244
   true_percentage( Example, x, 1_4, 3, 35, 95 ), % 213, 1
   not always( Example, x, 1_4, 39, 103 ), % 213, 0
   !.
class( Example, funnel ) :- % 213, 31
   dtw_le( Example, f_35, x, 1.241447 ), % 200, 1
   not euclid_le( Example, c_162, x, 8.629407 ), % 200, 0
   !.
class( Example, cylinder ) :- % 31, 13
   not true_percentage( Example, x, 3, 9, 73, 15 ), % 31, 0
   !.
class( Example, funnel ). % 13, 0
```

Fig. 1. Rules example. The rules are ordered, organized in a decision list. They were obtained from the dataset *CBF* (Sect. 4). At the right of each literal, the number of positive and negative examples covered by the (partial) rule.

Selection of Distance Literals. Calculating the distances between all the examples requires computing $O(e^2)$ distances, which might be too costly. Furthermore, considering all the examples as reference for selecting a literal can also be too slow. Instead, in each iteration, r reference examples are randomly selected from the non-covered examples (it is possible to use only positive reference examples or positive and negative). If $\mathrm{d}(n)$ is the time necessary for calculating the distance between two series with n points (n for the euclidean distance, n^2 for DTW), the best literal for r reference example can be calculated in $O(rv(e\,\mathrm{d}(n) + e \lg e))$. The term $e \lg e$ is the time necessary for ordering the distances to the reference example and selecting the best value according to the criterion. Since the same example can be selected in several iterations, the calculated distances are saved.

Multiclass Problems. When there are more than two classes, it is necessary to learn a theory for each class. The question is how to apply these theories to a new example. We have employed two approaches for dealing with multiclass problems. The first one is the use of ordered rules, also named decision lists [14]. In this case, the first rule that covers the example assigns its label to it. The learning process consists of generating a rule for the class with most uncovered examples and iterate until all the examples are covered. Figure 1 shows an example of such a decision list. The second approach [19] is to learn different theories independently for each class, apply all the rules to the new example and if there is a conflict, solve it by considering the distribution of training examples covered by the rules.

4 Experimental Validation

Datasets for classification of time series are not easy to find [12]. For this reason we have used four artificial datasets and only one "real world" dataset:

- **Cylinder, Bell and Funnel (CBF).** This is an artificial problem [12], in which there are there are 3 classes: cylinder, bell and funnel. Figure 2.a shows two examples of each class. There are 266 examples of each class and each series has 128 points.
- **Control Charts.** In this dataset there are 6 classes of control charts [1]. Figure 2.b shows some examples of three of the classes. The data used were obtained from the UCI KDD Archive [4]. The number of examples is 600, with 60 points in each series.
- **Waveform.** This dataset was introduced by [9]. We used the version from the UCI ML Repository [7]. The number of examples is 900 and the number of points in each series is 21.
- **Wave + Noise.** This dataset was generated in the same way as the previous one, but 19 random points are added at the end of each example, with mean 0 and variance 1. Again, we used the dataset from the UCI ML Repository.
- **Auslan.** Auslan is the Australian sign language, the language of the Australian deaf community. Instances of the signs were collected using an instrumented glove [12]. Each example is composed by 8 series: x, y and z position, wrist roll, thumb, fore, middle and ring finger bend. There are 10 classes and 20 examples of each class. The number of points in each example is variable and currently the system does not support variable length series, so they were resampled to 20 points.

The results for each dataset and setting were obtained using five five-fold stratified cross-validations. The percentages considered for the predicate true_percentage were 5, 15, 30, 50, 70, 85 and 95. For similarity literals, at most 10 positive and 10 negative reference examples were considered when selecting literals.

Results are shown in Table 1. Euclidean is the best for the *Wave* and *Wave + Noise* using decision lists or unordered rules, and for the *Auslan* dataset with decision lists. The performance of the euclidean distance is a bit surprising considering its simplicity. Nevertheless its results are the worst for *CBF* and *Control Charts*. In these cases DTW works much better. This situation is reasonable, because the *CBF* and *Control* datasets are designed specifically to test time series classifers, and hence involve situations like shifts, compressions and expansions. These situations are not present in the *Wave* datasets, and hence euclidean works better than DTW.

The use of always and sometimes gives the best results in the *Control* dataset. In the case of decision lists, the first position is shared with the use of the predicate true_percentage. The advantages of using it over using always and sometimes are only clear for the *CBF* dataset. This is probably due to the fact that the different situations that characterize the examples have a beginning and end for the *CBF* dataset but for the *Control* dataset there are no returns to the

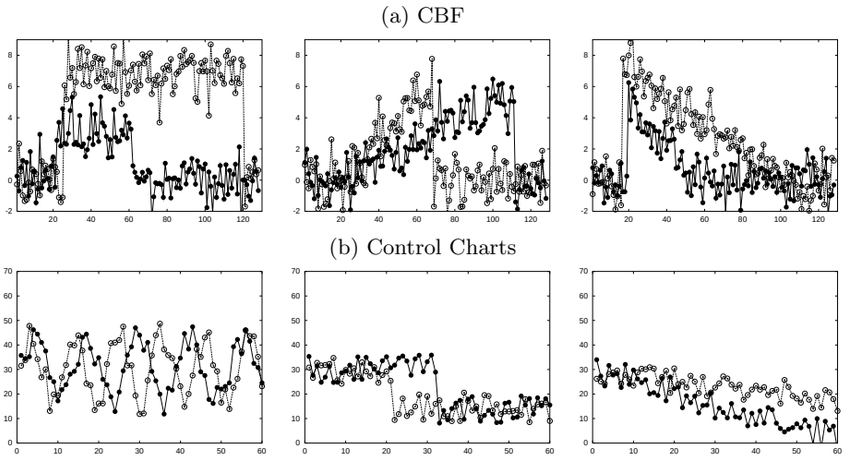

Fig. 2. Some examples of the datasets. Two examples of the same class are shown in each graph.

		EUC	DTW	AST	TRP	ALL
Decision Lists	CBF	6.67 2.39	2.16 1.22	4.51 1.38	2.76 1.25	**1.65** 1.10
	Control	4.63 1.38	3.23 1.71	**3.07** 1.64	**3.07** 1.66	3.20 1.55
	Wave	**19.64** 2.82	24.38 3.65	22.29 2.85	21.44 2.96	19.95 2.72
	Wave+Noise	**21.07** 3.04	26.78 2.78	24.80 3.35	24.62 2.44	22.31 3.17
	Auslan	**15.00** 4.21	21.60 6.33	21.40 7.00	20.10 4.97	16.60 5.49
Unordered Rules	CBF	6.87 2.16	2.11 1.02	4.31 2.03	2.88 1.28	**1.10** 0.89
	Control	6.03 2.20	4.10 1.89	**2.97** 1.23	4.13 1.98	3.60 1.64
	Wave	**19.47** 2.73	25.49 3.75	23.04 3.18	22.64 3.09	20.65 3.10
	Wave+Noise	**21.64** 2.16	28.84 2.51	24.67 2.53	25.58 3.39	22.91 2.54
	Auslan	24.60 7.49	24.20 5.94	22.80 6.55	23.80 6.13	**21.50** 7.74

Table 1. Experimental Results. The predicates considered are EUClidean, DTW, Always / SomeTime, TRue Percentage and ALL together. For each combination, the averaged error (in %) and the standard deviation of the 25 executions are shown. In boldface, the best results.

normal situation. The results obtained using all the predicates are the best for the *Auslan* dataset with unordered rules and for the *CBF* dataset. In the rest of the cases, the results are close to the best.

With respect to the use of decision lists or unordered rules, there is no clear winner regarding the error, although for the *Auslan* dataset decision lists work much better than unordered rules. Probably the inclusion of pruning methods could alter this situation.

5 Improving Accuracy by Boosting

At present, an active research topic is the use of *ensembles* of classifiers. They are obtained by generating and combining base classifiers, constructed using other machine learning methods. The target of these ensembles is to increase the accuracy with respect to the base classifiers. One of the most popular methods for creating ensembles is boosting [17], a family of methods, of which AdaBoost is the most prominent member. They work by assigning a weight to each example. In each iteration a base classifier is constructed, according to the distribution of weights. Afterwards, the weights are readjusted according to the result of the example in the base classifier. The final result is obtained by weighted votes of the base classifiers. Boosting inductive logic programming is described in [15].

Inspired by the good results of works using ensembles of very simple classifiers [17], *stumps*, we have studied base classifiers consisting only of one literal. The criterion used for selecting the best literal is to select the one with the smallest error, relative to the weights.

Multiclass Problems. There are several methods of extending AdaBoost to the multiclass case [17]. We have used AdaBoost.OC [16] since it can be used with any weak learner which can handle binary labeled data. It does not require that the weak learner can handle multilabeled data with high accuracy. The key idea is, in each round of the boosting algorithm, to select a subset of the set of labels, and train the binary weak learner with the examples labeled positive or negative depending if the original label of the example is or is not in the subset. In our concrete case, the rule inducer searches for a rule with the head:

$$\mathsf{class}(\ \mathsf{Example},\ [\mathsf{class}_1,\ \ldots\ \mathsf{class}_k]\ )$$

This predicate means that the Example is of one of the classes in the list. Figure 3 shows an ensemble of these classifiers.

The classification of a new example is obtained from a weighted vote of the results of the weak classifiers. For each rule, if its antecedent is true the weights of all the labels in the list are incremented by the weight of the rule, if it is false the weights of the labels out of the list are incremented. Finally, the label that has been given the highest weight is assigned to the example.

5.1 Results

The results obtained with the boosting process are summarized in Table 2. For all the datasets, the benefits of using boosting are substantial. Moreover, the evolution of the error with the number of iterations is rather good. Until the limit of the number of iterations considered, the increment of the number of iterations do not cause more overfitting.

– **Cylinder, bell and funnel.** The best previously published result, according to our knowledge, with this dataset is an error of 1.9 [12], using 10 fold cross

```
class( Example, [ decreasing, downward, increasing ] ) :-    % 240, 240
   not true_percentage( Example, x, 5, 43, 59, 5 ).    % 197, 26
   % 0.456818
class( Example, [ decreasing, upward, normal ] ) :-    % 240, 240
   true_percentage( Example, x, 1_4, 14, 22, 70 ).    % 229, 84
   % 0.422232
class( Example, [ cyclic, decreasing, downward ] ) :-    % 240, 240
   sometime( Example, x, 2, 22, 54 ).    % 240, 0
   % 0.700627
class( Example, [ cyclic, decreasing, normal ] ) :-    % 240, 240
   dtw_le( Example, cyclic_54, x, 59.898940 ).    % 159, 0
   % 0.536895
...
```

Fig. 3. Initial fragment of an ensemble of classifiers, obtained with AdaBoost.OC, for the dataset *control charts* (Sect. 4). The weights of each individual classifier are below each classifier.

Iterations	5	10	20	30	40	50
CBF	2.41 1.21	1.50 0.85	0.75 0.60	0.60 0.53	0.58 0.65	**0.50** 0.60
Control	22.30 2.58	8.70 5.38	2.27 1.62	1.80 1.22	1.60 1.25	**1.47** 1.19
Wave	19.58 2.55	18.36 2.79	15.64 2.46	15.49 2.08	**15.00** 2.32	15.04 2.36
Wave + Noise	23.87 3.13	20.56 3.24	18.29 2.73	17.60 2.41	17.18 2.36	**16.78** 2.11
Auslan	61.50 7.14	37.90 7.90	16.70 5.72	11.50 4.39	8.40 3.88	7.60 3.27
+100 iter.		3.60 2.51	3.80 2.71	3.20 2.34	3.10 2.20	2.90 1.87
+150 iter.		3.00 1.77	3.10 2.20	2.50 1.91	2.80 2.20	2.40 2.22
+200 iter.		2.40 2.22	2.30 2.16	**1.90** 2.20	2.20 1.95	2.00 1.77

Table 2. Results obtained with boosting, for several numbers of iterations, using all the background predicates. For the *Auslan* dataset, the table includes results with more iterations (110, 120 ... 150; 160 ... 200 and 210 ... 250). In some cases, it is possible to improve these results using not all the predicates (see discussion for each dataset).

validation. From iteration no. 10, the results shown in table 2 are better than this result, and from iteration no. 20 the results are smaller than 1.0. Moreover, the results using rules without boosting (Table 1) are also better than 1.9 (1.65 and 1.10).

- **Control charts.** The only previous result we are aware of regarding this dataset is for similarity queries [1], and not for supervised classification. To check if this dataset was trivial, we tested it with C4.5, over the raw data, and obtained an average error of 8.6 (also using five five-fold cross validation). It should also be noted that obtained error is reduced to 0.82 using 100 iterations and only dtw_le.
- **Waveform.** The error of a Bayes optimal classifier on this dataset is approximately 14 [9]. The best previous result we are aware of for this dataset is an error of 15.21 [11]. That result was obtained using boosting, with decision

trees as base classifiers, which are much more complex than our base classifiers (clauses with one literal in the body). The obtained error is reduced to 14.42 using 100 iterations and euclid_le.

- **Wave + Noise.** Again, the error of an optimal Bayes classifier on this dataset is 14. This dataset was tested with bagging, boosting and variants over decision trees [3]. Although their results are given in graphs, their best error is apparently approximately 17.5. The obtained error is reduced to 14.69 using 100 iterations and euclid_le.
- **Auslan.** This is the dataset with the highest number of classes (10). In order to distinguish between this large number of classes, it turned out that 50 one-body-literal clauses were too few, so we incremented the number of iterations for this dataset, until 250. The best result previously published is an error of 2.50 [12], which is greater than the results obtained after 200 iterations, as shown in Table 2.

6 Conclusions and Future Work

A multivariate time series classification system has been developed, using ILP techniques. Two types of background predicate were considered: those based on intervals and regions and those based on similarity functions. The use of similarity literals allows a smooth integration of Instance Based Learning and ILP, leading to that similarity functions can be defined by the user and incorporated as background knowledge and that these functions appear explicitly in the rules instead of being an external element to the learned theory. Special purpose techniques have been presented that allow these predicates to be handled efficiently when performing top-down induction. Furthermore, we have demonstrated that by employing boosting, the accuracy of the resulting time series classifiers can be improved significantly. Experiments on several different datasets show that the proposed method is highly competitive with previous approaches. On all data sets, the proposed method achieves better than all previously reported results.

Boosting generic learners, which are not designed for time series, such as decision trees, can produce good results for time series classification. Nevertheless, the incorporation of temporal predicates, in the background knowledge, improves the performance of boosting, especially with respect to the size of the classifiers.

Another method to improve the accuracy of a classifier is the use of pruning techniques. Pruning also enhances the comprehensibility of classifiers, while boosting worsens it. Nevertheless, pruning and boosting are not incompatible, e.g., when boosting rules it is possible to prune them [10]. Hence, it would be convenient to incorporate pruning techniques in the rule induction system, and also to apply boosting over pruned rules.

Acknowledgements. To the maintainers of the ML [7] and KDD [4] UCI Repositories. To Mohammed Waleed Kadous, David Aha and Robert J. Alcock for, respectively, donating the *Auslan*, *Wave* and *Control Charts* datasets.

References

[1] R.J. Alcock and Y. Manolopoulos. Time-series similarity queries employing a feature-based approach. In *7th Hellenic Conference on Informatics*, Ioannina, Greece, 1999.

[2] C.J. Alonso and J.J. Rodríguez. A graphical rule language for continuous dynamic systems. In *Computational Intelligence for Modelling, Control and Automation, CIMCA-99*. IOS Press, 1999.

[3] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning*, 36(1/2), 1999.

[4] S.D. Bay. The UCI KDD archive, 1999. `http://kdd.ics.uci.edu/`.

[5] F. Bergadano and D. Gunetti. *Inductive Logic Programming: from machine learning to software engineering*. The MIT Press, 1995.

[6] D.J. Berndt and J. Clifford. Finding patterns in time series: a dynamic programming approach. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*. AAAI Press / MIT Press, 1996.

[7] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[8] H. Blockeel and L. De Raedt. Lookahead and discretization in ILP. In *7th International Workshop on Inductive Logic Programming, ILP'97*. Springer, 1997.

[9] L. Breiman, J.H. Friedman, A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1993.

[10] W.W. Cohen and Y. Singer. A simple, fast, and effective rule learner. In *16th National Conference on Artificial Intelligence, AAAI-99*, 1999.

[11] T.G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, 1999.

[12] M.W. Kadous. Learning comprehensible descriptions of multivariate time series. In 16th *International Conference of Machine Learning (ICML-99)*. Morgan Kaufmann, 1999.

[13] M. Kubat, I. Koprinska, and G. Pfurtscheller. Learning to classify biomedical signals. In R.S. Michalski, I. Bratko, and M. Kubat, editors, *Machine Learning and Data Mining*. John Wiley & Sons, 1998.

[14] R.J. Mooney and M.E. Califf. Induction of first-order decision lists: results on learning the past tense of english verbs. *JAIR*, 3, 1995.

[15] J.R. Quinlan. Boosting first-order learning. In S. Arikawa and A. Sharma, editors, *Algorithmic Learning Theory, 7th International Workshop, ALT'96*. Springer, 1996.

[16] R.E. Schapire. Using output codes to boost multiclass learning problems. In *14th International Conference on Machine Learning (ICML-97)*, 1997.

[17] R.E. Schapire. A brief introduction to boosting. In *16th International Joint Conference on Artificial Intelligence (IJCAI-99)*. Morgan Kaufmann, 1999.

[18] A. Srinivasan and R.C. Camacho. Numerical reasoning with an ILP system capable of lazy evaluation and customized search. *Journal of Logic Programming*, 40(2–3), 1999.

[19] W. Van Laer, L. De Raedt, and S. Dzeroski. On multi-class problems and discretization in inductive logic programming. In *10th International Symposium on Methodologies for Intelligent Systems (ISMIS97)*. Springer, 1997.