Skimmed Classifiers

Rogerio Salvini, Eduardo Aguilar and Inês C. Dutra

Department of Systems Engineering and Computer Science

COPPE, Federal University of Rio de Janeiro, Brazil

{rsalvini,eaguilar,ines}@cos.ufrj.br

Abstract

Most Inductive Logic Programming (ILP) systems use a greedy covering algorithm to find a set of clauses that best model positive examples. This set of clauses is called a theory and can be seen as an *ensemble* of clauses. It turns out that the search for a theory within the ILP system is very time consuming and often yields overly complex classifiers. One alternative approach to obtain a theory is to use the ILP system to non deterministically learn one clause at a time, several times, and to combine the obtained clauses using ensemble methods.

1 Introduction

Inductive Logic Programming (ILP) systems have been quite successful in extracting comprehensible models of relational data. Indeed, for over a decade, ILP systems have been used to construct predictive models for data drawn from diverse domains. These include the sciences [18], engineering [8], language processing [33], environment monitoring [11], software analysis [4], pattern learning and link discovery [23, 24]. Most ILP systems use a greedy covering algorithm that repeatedly examines candidate clauses (the "search space") to find good rules (or theories). Ideally, the search will stop when the rules cover nearly all positive examples with only a few negative examples being covered.

This algorithm poses some challenges, since the search space can grow very quickly, sometimes turning unfeasible the search for a good solution. Several techniques have been proposed to improve search efficiency of ILP algorithms. Such techniques include improving computation times at individual nodes [3, 27], better representations of the search [2], sampling the search space [28, 29, 32], parallelism [6, 15, 22, 31, 32, 9, 13, 12], and utilization of ensemble methods [10].

Ensembles are classifiers that combine the predictions of multiple classifiers to produce a single prediction [7]. Several researchers have been interested in the use of ensemble-based techniques for ILP. To our knowledge, the original work

in this area is Quinlan's work on the use of boosting in FOIL [26]. His results suggested that boosting could be beneficial for first-order induction. More recently, Hoche proposed confidence-rated boosting for ILP with good results [17]. Zemke proposed bagging as a method for combining ILP classifiers with other classifiers [34]. Dutra *et al* [10] studied bagging in the context of the ILP system Aleph [30] learning theories. Their results showed that the applications benefited from ensembles to a limited extent.

In this work we argue that learning **a single clause at a time** rather than learning **whole theories** (or sets of clauses) at a time can be more cost-effective and produce simpler classifiers. Our alternative approach then is to use *ensembles* of **clauses**. To some extent, an induced theory is an ensemble of clauses. However, finding an induced theory is very time consuming and can produce very complex classifiers. In this work we learn single clauses, and use ensemble methods to combine them, which produce classifiers that are better than any single clause or theory, in much less time than finding a theory.

The paper is organized as follows. First, we present in more detail the ensemble method used in this work. Next, we discuss our experimental setup and the applications used in our study. We then discuss how ensembles of clauses compare with ensembles of theories. Last, we offer our conclusions and suggest future work.

2 Ensemble Methods

In general, ensemble methods work by combining the predictions of several (hopefully different) weak classifiers to produce one final strong classifier.

Ensemble generation assumes two distinct phases: (1) training, and (2) combining the classifiers. The ensemble methods vary according to the constraints imposed to the training phase, and to the kind of combination used.

Figure 1 shows the structure of an ensemble of logic programs. This structure can also be used for classifiers other than logic programs. In the figure, each program $P_1, P_2, ..., P_N$ is trained using a set of training instances. At classification time each program receives the same input and executes on it independently. The outputs of each program are then combined and an output classification reached. Figure 1 illustrates that in order to obtain good ensembles one must address three different problems:

- how to generate the individual programs;
- how many individual programs to generate;
- how to combine their outputs.

Regarding the first problem, research has demonstrated that a good ensemble is one where the individual classifiers are accurate and make their errors in

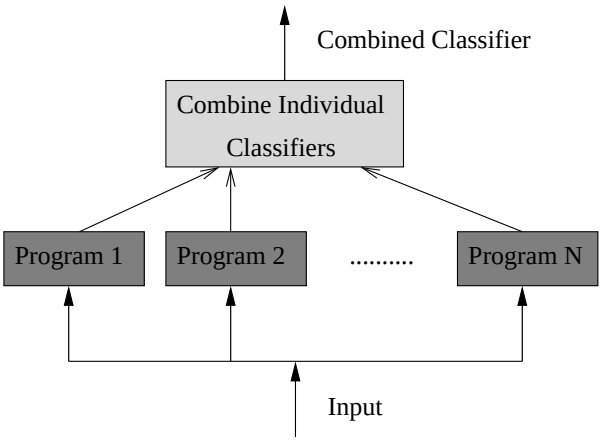

Figure 1: An Ensemble of Classifiers

different parts of the instance space [20, 25]. Obviously, the output of several classifiers is useful only if there is disagreement between them. Hansen and Salamon [16] proved that if the average error rate is below 50% and if the component classifiers are independent, the combined error rate can be reduced to 0 as the number of classifiers goes to infinity. In this work, we argue that ensembles of clauses produce rather independent classifiers than ensembles of theories as studied by Dutra *et al* [10].

Methods for creating the individual classifiers therefore focus on producing classifiers with some degree of diversity. In the present work, we follow two approaches to produce such classifiers. We produce clauses and theories (sets of clauses). We believe that clauses have a greater degree of diversity than theories.

The second issue we had to address was the choice of how many individual classifiers to combine. Previous research has shown that most of the reduction in error for ensemble methods occurs with the first few additional classifiers [16]. Larger ensemble sizes have been proposed for decision trees, where gains have been seen up to 25 classifiers.

The last problem concerns the combination algorithm. An effective combining scheme is often to simply average the predictions of the classifiers [1, 5, 20, 21]. An alternate approach relies on a pre-defined *voting threshold*: if the number of clauses or theories that cover an example is above or equal to the threshold, we say that the example is positive, otherwise the example is negative. Thresholds may range from 1 to the ensemble size. A voting threshold of 1 corresponds to a classifier that is the disjunction of all theories. A voting threshold equal to the ensemble size corresponds to a classifier that is the conjunction of all theories. This voting scheme was used in [10].

Individual classifiers that compose an ensemble can be obtained from different samples of the dataset or from the same dataset. They can also be obtained

from one single ILP algorithm (homogeneous classifiers) or from different ILP algorithms (heterogeneous classifiers). In this work we use homogeneous classifiers and obtain classifiers from different samples of the datasets. The classifiers can be independent or dependent, depending on the ensemble method employed.

Several methods have been presented for ensemble generation. The two most popular are *bagging* and *boosting* [14]. Bagging works by training each data set on a random sample from the training set. Classifiers in this case are totally independent from each other. Boosting works by assigning penalties to misclassified examples, and refining the search for clauses that try to cover the misclassified examples. Therefore, in boosting, the classifiers are dependent. The current classifier is dependent on the previous in the training sequence.

In this work we evaluate bagging. Bagging classifiers are obtained by training each classifier on a random sample of the training set. Each classifier's training set is generated by randomly, uniformly drawing K examples with *replacement*, where K is the size of the original training set. Thus, many of the original examples may be repeated in the classifier's training set. We then contrast the results obtained with clause-based learning to the results obtained with theory-based learning.

3 Methodology

We use the ILP system Aleph [30] in our study. Aleph assumes (a) background knowledge B in the form of a Prolog program; (b) some language specification L describing the hypotheses; (c) an optional set of constraints I on acceptable hypotheses; and (d) a finite set of examples E. E is the union of a nonempty set of "positive" examples E^+, such that none of the E^+ are derivable from B, and a set of "negative" examples E^-.

Aleph tries to find one hypothesis H in L, such that: (1) H respects the constraints I; (2) The E^+ are derivable from B, H, and (3) The E^- are not derivable from B, H. By default, Aleph uses a simple greedy set cover procedure that constructs such a hypothesis one clause at a time. The final classifier is a collection of clauses (a theory).

In the search for any single clause, Aleph randomly selects an uncovered positive example as the seed example, saturates this example, and performs an admissible search over the space of clauses that subsume this saturation, subject to a user-specified clause length bound. As it was mentioned before, this is a very time-consuming process. We contrast this approach with the approach of generating one single clause and stopping, using a randomly chosen example as a seed.

Aleph allows the user to set a number of parameters. We always set the following parameters as follows:

- search strategy: `search`. We set it to be breadth-first search, `bf`. This enumerates shorter clauses before longer ones. At a given clause length, clauses are re-ordered based on their evaluation. This is the Aleph default

strategy that favours shorter clauses to avoid the complexity of refining larger clauses.

- evaluation function: `evalfn`. We set this to be coverage. Clause utility is measured as $P - N$, with P and N being the number of positive and negative examples covered by the clause, respectively.

- chaining of variables: `i`. This Aleph parameter controls variable chaining during saturation: chaining depth of a variable that appears for the first time in a literal L_i, is 1 plus the maximum chaining depth of all variables that appear in previous literals $L_j, j < i$. We used a value of 5 instead of the default value of 2 in order to obtain more complex relations between literals in a clause.

- max number of nodes allowed: `maxnodes`. This corresponds to the number of clauses in the search space. We set this to be 100,000.

- maximum number of literals in a clause: `maxclauselength`. This was chosen based on some previous experiments and was set to 5.

- minimum clause accuracy: this is the minimum accuracy acceptable when generating a new clause. It was set to 0.9 (i.e., accuracy of 90%).

We used the same set of parameters used by Dutra *et al* [10] in order to compare some results.

Our experimental methodology employs five-fold cross-validation. For each fold, we consider ensembles with size varying from 1 to 25, when learning theories. Thus each application ran 25 times. This step generates 25 files with one theory per file (set of clauses). For the experiment that learns clauses, we varied the size of the ensemble from 1 to the average size of the theory, per application, per fold. Therefore we can compare the performance of obtaining one theory using a greedy covering algorithm with the performance of obtaining an ensemble of clauses (with the same theory size), without using a greedy covering algorithm.

For the evaluation phase, we used one popular metric to evaluate the quality of the ensembles, the accuracy. We studied how average accuracy varies with ensemble size. We present accuracy as $\frac{T_p+T_n}{Total\_of\_exs}$, where T_p and T_n, are respectively, the number of positive and negative covered examples, and $Total\_of\_exs$ is the dataset size.

We wish to test the effectiveness of different sizes of ensembles. Again, we do not repeat the ILP runs themselves to learn entirely new theories for each different ensemble size. Rather, we use the theories from the previous step. Because our results may be distorted by a particularly poor or good choice of these theories, we repeat this selection process 30 times and average the results.

Figure 2 shows the general algorithm to perform the evaluation step. The loop that goes from line 2 to 8 computes the points necessary to produce the accuracies, where Tn is the rate of true negatives, Fn is the rate of false negatives, Fp is the rate of false positives and Tp is the rate of true positives. This

```
1. for fold = 1 to numFolds do
2.   for ensSize = 1 to numIterations do
3.     randomly select 30 sets of size ensSize
4.     for threshold = 1 to ensSize do
5.       for s = 1 to 30 do
6.         errorSum[fold,ensSize,threshold] += (Tn,Fn,Fp,Tp)
7.       endfor
8.     endfor
9.   endfor
10. endfor
```

Figure 2: General algorithm used in the evaluation step

is done for 30 sets, where each set contains $ensSize$ theories that are selected randomly from the 25. This builds a table per fold, per ensemble size, where each line represents the error sum for the 30 sets, for each voting threshold.

This is repeated twice, once for a tuning set, from where we extract the best threshold for each ensemble size, and again for the validation phase, where the voting threshold used is the one that was the best for the tuning phase. The final results are obtained by averaging the numbers across all folds.

The same procedure is repeated for the experiment that generates clauses and do not use the greedy covering algorithm, but at this time, the ensemble sizes can vary from 1 to the average size of the theories.

Bagging was implemented straightforwardly. We only needed to generate the bags and run the training/test experiments independently.

All experiments were performed on the same machine, AMD 2.8 GHz with 512 MBytes of RAM, using Yap Prolog 5, and Aleph 3.0 running Mandriva Linux 2007.

3.1 Benchmark Datasets

Our benchmark set is composed of five datasets that correspond to non-trivial ILP applications that are also used in other works. We next describe the characteristics of each dataset with its associated ILP application, and present a dataset summary table.

Amine

Our first learning task is to predict amine re-uptake inhibition to discover new drugs for the Alzheimer disease [19]. We were given some positive examples of drugs that were effective against the disease, and some negative examples. The task is to build a model for an active drug that distinguishes between active and inactive drugs.

Carcinogenesis

Our second application concerns the prediction of carcinogenicity test outcomes on rodents. This application has a number of attractive features: it is an important practical problem; the background knowledge consists of large numbers of non-determinate predicate definitions; previous experience suggests that a fairly large search space needs to be examined to obtain a good clause.

Choline

This application is also related to drug discovery for the Alzheimer disease. The learning task is to identify the inhibition of the aceto-choline-esterase enzyme.

Mutagenesis

The prediction of mutagenesis is important as it is relevant to the understanding and prediction of carcinogenesis. Not all compounds can be empirically tested for mutagenesis, e.g. antibiotics. The considered dataset has been collected with the intention to search for a method for predicting the mutagenicity of aromatic and hetero-aromatic nitro compounds.

Protein

Our last dataset consists of a database of genes and features of the genes or of the proteins for which they code, together with information about which proteins interact with one another and correlations among gene expression patterns. This dataset is taken from the function prediction task of KDD Cup 2001. While the KDD Cup task involved 14 different protein functions, our learning task focuses on the challenging function of "metabolism": predicting which genes code for proteins involved in metabolism. This is not a trivial task for an ILP system.

Table 1 summarizes the main characteristics of each application. The second and third columns correspond to the size of the full datasets. Bags are created by randomly picking elements, with replacement, from the full dataset. The last column indicates the size of the background knowledge (number of facts and rules).

Table 1: Datasets Characteristics

Dataset	E+	E-	BK Size
Amine	343	343	232
Carcinogenesis	182	148	24988
Choline	663	663	232
Mutagenesis	125	63	15113
Protein	172	690	6913

4 Results

This section presents our results and analyses the performance of each application.

Accuracies presented are averaged across all folds, and are related to the test sets. For each ensemble size, accuracy values for the test sets are obtained varying the voting threshold. The results showed uses the threshold that produced the best accuracy in the tuning set where we trained on 3 of the 4 training data of each fold to obtain the best threshold. For clarity's sake, these parameter values are not shown in the curves.

Tables 2 and 3 shows a summary of execution time and best accuracy achieved (at a given ensemble size), for three applications. The execution times correspond to the total time to produce ensembles of all sizes.

Table 2: Ensemble of Theories

Dataset	Time	Acc.	Ens.size	Number of clauses	Total clauses
Amine	10h41min	0.82	16	60	960
Choline	5h03min	0.81	25	139	3475
Protein	3h42min	0.81	25	63	1575

Table 3: Ensemble of Clauses

Dataset	Time	Acc.	Ens.size	Number of clauses	Total clauses
Amine	3min	0.81	46	1	46
Choline	34min	0.78	139	1	139
Protein	24min	0.80	63	1	63

We can observe that the accuracies obtained with ensembles of theories are slightly better than the ones obtained with ensembles of clauses, however the execution time to obtain the theories are much higher. On the other hand, we can obtain accuracies (with ensembles of clauses) very close to the best achieved by ensembles of theories in far less time.

Moreover, the length of the classifier obtained with ensemble of clauses is much shorter than the one obtained with ensemble of theories. This can be of utmost importance for the expert that will analyse the generated rules.

With these results, we wanted to answer the question: is it possible to improve a classifier of rules even further, not changing the ILP learner? The answer is yes. It suffices to find the smallest set of clauses that models all data, without considering exceptions that do not generalize using the given learner. Examples that do not generalize well should neither be part of the training set

or of the test set, because they distort the quality of the learned rules. We then decided to change the methodology used so far and use all positive examples of the training set to generate clauses, instead of using a sample of the examples as is done with the bagging method. Our new methodology also removes redundant or equivalent clauses from the final ensemble as well as the Aleph specialized clauses that cover only one example.

The algorithm that performs the selection is very simple: we generate a bitmap of clause coverage, where each row is an example and each column is a clause. If a clause covers an example, the bitmap position is filled with a 1. Otherwise, it is filled with a 0. Let $U = (a, b, c)$ and $V = (x, y, z)$, and 3 examples. For each pair of clauses U and V, we then compute the number of $1\,'s$ of U (tU) and the number of $1\,'s$ of V (tV). Let W be $a * x + b * y + c * z$. If $tU = tV = W$, clauses U and V cover the same set of examples. We then randomly remove one of them. If $tU = W$, remove U. If $tV = W$, remove V.

We used this new methodology with the datasets mentioned before, and added two other popular datasets: mutagenesis and carcinogenesis, with 5-fold cross-validation. The results are shown in table 4.

Table 4: Skimmed classifier results

Dataset	Time	Acc.	Avg. number of selected clauses
Amine	5min	0.83	34
Carcinogenesis	52min	0.57	71
Choline	12min	0.73	48
Mutagenesis	47min	0.80	13
Protein	5min	0.77	54

The times shown in table 4 are the total time to generate one clause per each positive example that is used as a seed to Aleph, for all folds. The accuracy is the average accuracy of all folds, after the clauses are selected using the new filtering methodology, where redundant and equivalent clauses are removed from the set of generated clauses. The number of clauses selected also correspond to the average across all folds.

We need to stress some important issues regarding this methodology:

- all examples that are not generalized by Aleph are not part of the classifier. In other words, the final classifier will never classify these examples. This contrasts to the previous classifier generated with bagging, where examples that are not generalized are part of the final ensemble. As we choose all positive examples to generate a clause, we can discard the examples that are not generalized by Aleph.

- The algorithm that constructs the final classifier searches for the smallest set of clauses that has maximum coverage of positives. We are not yet

looking at false positives, therefore, the accuracy maybe not so good. The voting mechanism used to obtain the previous classifier helps to reduce the false positive rate, which may increase the accuracy.

- There is one subset of examples that concentrates all information. Depending on the dataset, there can be examples that can not be generalized. In other words, these examples have characteristics that do not appear in any other example, and do not have characteristics that appear in other examples. The performance of any classifier will depend on where these examples appear in the training set or in the test set.

Although the issues above may be an impediment to use the methodology we are using, we observe gains related to the ensemble generation used before. The classifiers obtained with this methodology are simpler and have equivalent performance to the ones generated using bagging.

5 Discussion

The results shown demonstrate several important facts in ILP learning:

- Ensembles of clauses learned with ILP are very powerful methods to obtain good accuracies in a small amount of time;
- Bagging is effective on obtaining good quality classifiers, even when the individual classifier is as weak as a single clause;
- Ensemble sizes greater than 25 may produce better accuracies for methods that learn clauses;
- Learning theories can be unfeasible depending on the application, and learning clauses can produce equal or better results;
- Theories do not benefit much from ensembles. We believe that this is because a theory is already an ensemble, and theories tend to be repeated across folds;
- Very weak individual classifiers such as single clauses can benefit significantly from ensembles, and producing a final classifier takes time that is several orders of magnitude less than the time spent to learn theories;
- Clauses are classifiers simpler than theories. Consequently, an ensemble of clauses is simpler than an ensemble of theories, which may help an expert to interpret the results.

6 Conclusions and Future Work

This work presented an empirical study of ensemble methods, in the Inductive Logic Programming setting. We chose to apply ensembles to two different individual classifiers: (1) classifiers composed of one single clause, faster to obtain; and (2) classifiers composed of one or more clauses (theory), that take a huge amount of time to obtain. We applied ensemble methods to classifiers obtained with the Aleph system. We tested three ILP applications already used in the literature. Our results show that ensembles built from single clauses are more cost-effective than ensembles built from theories. Clauses are much faster to be learned, and an ensemble of clauses produces more readable classifiers.

We also explored further the ability of obtaining simpler classifiers by using a very simple methodology, where we generate clauses using all positive examples, in contrast to bagging, and generate an ensemble by removing redundant and equivalent clauses. In this experiment we used 5 datasets, including the three mentioned before. This approach showed to be better than using ensembles whose clauses were generated using bagging. The classifiers are simpler, and their quality is equivalent to that of classifiers based on ensemble of clauses generated from bagging.

References

[1] E. Alpaydin. Multiple networks for function learning. In *IEEE International Conference on Neural Networks*, pages 9–14, 1993.

[2] H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, J. Ramon, and H. Vandecasteele. Executing query packs in ILP. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 60–77. Springer-Verlag, 2000.

[3] H. Blockeel, B. Demoen, G. Janssens, H. Vandecasteele, and W. Van Laer. Two advanced transformations for improving the efficiency of an ILP system. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 43–59, 2000.

[4] I. Bratko and M. Grobelnik. Inductive learning applied to program construction and verification. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 279–292. J. Stefan Institute, 1993.

[5] L. Breiman. Stacked Regressions. *Machine Learning*, 24(1):49–64, 1996.

[6] L. Dehaspe and L. De Raedt. Parallel inductive logic programming. In *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, 1995.

[7] T. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli, editors, *First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2000.

[8] B. Dolšak and S. Muggleton. The application of ILP to finite element mesh design. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 225–242, 1991.

[9] I. C. Dutra, D. Page, V. Santos Costa, J. Shavlik, and M. Waddell. Toward automatic management of embarrassingly parallel applications. In *Euro-Par03, Klagenfurt, Austria, August 26 - 29*, pages 509–516, 2003.

[10] I. C. Dutra, D. Page, and J. Shavlik V. Santos Costa. An empirical evaluation of bagging in inductive logic programming. In *Proceedings of the 12th International Conference on Inductive Logic Programming*, Lecture Notes in Artificial Intelligence. Springer-Verlag, September 2002.

[11] S. Džeroski, L. Dehaspe, B. Ruck, and W. Walley. Classification of river water quality data using machine learning. In *Proceedings of the 5th International Conference on the Development and Application of Computer Techniques to Environmental Studies*, 1995.

[12] Nuno A. Fonseca, Rui Camacho, and Fernando Silva. Strategies to Parallelize ILP Systems. In *To appear in Proceedings of the 15th International Conference on Inductive Logic Programming*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2005.

[13] Nuno A. Fonseca, Fernando Silva, Vitor Santos Costa, and Rui Camacho. A pipelined data-parallel algorithm for ILP. In *To appear Proceedings of 2005 IEEE International Conference on Cluster Computing*. IEEE, September 2005.

[14] Y. Freund and R. Shapire. Experiments with a new boosting algorithm. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 148–156. Morgan Kaufman, 1996.

[15] J. Graham, D. Page, and A. Wild. Parallel inductive logic programming. In *Proceedings of the Systems, Man, and Cybernetics Conference*, 2000.

[16] L. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, October 1990.

[17] S. Hoche and S. Wrobel. Relational learning using constrained confidence-rated boosting. In Céline Rouveirol and Michèle Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 51–64. Springer-Verlag, September 2001.

[18] R. King, S. Muggleton, and M. Sternberg. Predicting protein secondary structure using inductive logic programming. *Protein Engineering*, 5:647–657, 1992.

[19] R. King, M. Sternberg, and A. Srinivasan. Relating chemical activity to structure: An examination of ilp successes. *New Generation Computing Journal*, 13(3&4):411–433, 1995.

[20] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. The MIT Press, 1995.

[21] N. Lincoln and J. Skrzypek. Synergy of clustering multiple backpropagation networks. In *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 1989.

[22] T. Matsui, N. Inuzuka, H. Seki, and H. Ito. Parallel induction algorithms for large samples. In S. Arikawa and H. Motoda, editors, *Proceedings of the First International Conference on Discovery Science*, volume 1532 of *Lecture Notes in Artificial Intelligence*, pages 397–398. Springer-Verlag, December 1998.

[23] R. Mooney, P. Melville, L. P. Rupert Tang, J. Shavlik, I. Dutra, D. Page, and V. Santos Costa. Relational data mining with inductive logic programming for link discovery. In *Proceedings of the National Science Foundation Workshop on Next Generation Data Mining*, Baltimore, Maryland, USA, 2002.

[24] R. J. Mooney, P. Melville, L. R. Tang, J. Shavlik, I. C. Dutra, D. Page, and V. S. Costa. Relational data mining with inductive logic programming for link discovery. In Hillol Kargupta, Anupam Joshi, K. Sivakumar, and Yelena Yesha, editors, *Data Mining: Next Generation Challenges and Future Directions*. AAAI/MIT Press, Berlin, 2003.

[25] D. W. Opitz and J. W. Shavlik. Actively searching for an effective neural-network ensemble. *Connection Science*, 8(3/4):337–353, 1996.

[26] J. R. Quinlan. Boosting first-order learning. *Algorithmic Learning Theory, 7th International Workshop, Lecture Notes in Computer Science*, 1160:143–155, 1996.

[27] V. Santos Costa, A. Srinivasan, and R. Camacho. A note on two simple transformations for improving the efficiency of an ILP system. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 225–242. Springer-Verlag, 2000.

[28] M. Sebag and C. Rouveirol. Tractable induction and classification in first-order logic via stochastic matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–893. Morgan Kaufmann, 1997.

[29] A. Srinivasan. A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery*, 3(1):95–123, 1999.

[30] A. Srinivasan. *The Aleph Manual*, 2001.

[31] J. Struyf and H. Blockeel. Efficient cross-validation in ILP. In Céline Rouveirol and Michèle Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 228–239. Springer-Verlag, September 2001.

[32] F. Zelezny, A. Srinivasan, and D. Page. Lattice-search runtime distributions may be heavy-tailed. In *Proceedings of the 12th International Conference on Inductive Logic Programming*. Springer Verlag, July 2002.

[33] J. Zelle and R. Mooney. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 817–822, Washington, D.C., July 1993. AAAI Press/MIT Press.

[34] S. Zemke. Bagging imperfect predictors. In *Proceedings of the International Conference on Artificial Neural Networks in Engineering, St. Louis, MI, USA*. ASME Press, 1999.