Appears in the *Proc. of the Intl. Conf. on Inductive Logic Programming, July 2002, Sydney, Australia.*

An Empirical Evaluation of Bagging in Inductive Logic Programming

Inês de Castro Dutra, David Page, Vítor Santos Costa and Jude Shavlik

Department of Biostatistics and Medical Informatics and
Department of Computer Sciences,
University of Wisconsin-Madison, USA

Abstract. Ensembles have proven useful for a variety of applications, with a variety of machine learning approaches. While Quinlan has applied boosting to FOIL, the widely-used approach of bagging has never been employed in ILP. Bagging has the advantage over boosting that the different members of the ensemble can be learned and used in parallel. This advantage is especially important for ILP where run-times often are high. We evaluate bagging on three different application domains using the complete-search ILP system, Aleph. We contrast bagging with an approach where we take advantage of the non-determinism in ILP search, by simply allowing Aleph to run multiple times, each time choosing "seed" examples at random.

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 [16], engineering [10], language processing [33], environment monitoring [11], and software analysis [5]. In a nutshell, ILP systems repeatedly examine candidate clauses (the "search space") to find good rules. Ideally, the search will stop when the rules cover nearly all positive examples with only a few negative examples being covered.

Unfortunately, the search space can grow very quickly in ILP applications. Several techniques have therefore been proposed to improve search efficiency. Such techniques include improving computation times at individual nodes [4, 26], better representations of the search [3], sampling the search space [27, 28, 32], and parallelism [8, 13, 19]. Parallelism can be obtained from very different alternative approaches, such as dividing the search tree, dividing the examples, or even through performing cross-validation in parallel [31].

An intriguing alternative approach that can lead to better accuracy whilst taking advantage of parallelism is the use of *ensembles*. Ensembles are classifiers

that combine the predictions of multiple classifiers to produce a single prediction [9]. To some extent, an induced theory is an ensemble of clauses. We would like to go one step further and *combine different theories to form a single ensemble*. The main advantage is that the ensemble is often more accurate than its individual components. Moreover, we can use parallelism both in generating and in actually evaluating the ensemble.

Several methods have been presented for ensemble generation. In this work, we concentrate on a popular method that is known to generally create a more accurate ensemble than individual components, *bagging* [6]. Bagging works by training each classifier on a random sample from the training set. In contrast to other well-known techniques for ensemble generation, such as boosting [12], bagging has the important advantage that it is effective on "unstable learning algorithms" [7], where small variations in parameters can cause huge variations in the learned theories. This is the case with ILP. A second advantage is that it can be implemented in parallel trivially.

We contrast bagging with a method we name *different seeds*, where we try to take advantage of the non-determinism in seed-based search by simply combining different theories obtained from experimenting with different seed examples, while always using the original training set. This method is also easily parallelisable.

Several researchers have been interested in the use of ensemble-based techniques for Inductive Logic Programming. To our knowledge, the original work in this area is Quinlan's work on the use of boosting in FOIL [25]. His results suggested that boosting could be beneficial for first-order induction. More recently, Hoche proposed confidence-rated boosting for ILP with good results [15]. Zemke recently proposed bagging as a method for combining ILP classifiers with other classifiers [34]. The contributions of our paper are to experimentally evaluate bagging on three particularly challenging ILP applications, and to compare bagging with the approach of different seeds.

The paper is organised as follows. First, we present in more detail ensemble techniques, focusing on bagging. Next, we discuss our experimental setup and the applications used in our study. We then discuss how ensembles perform on our benchmarks. Last, we offer our conclusions and suggest future work.

2 Ensembles

Ensembles aim at improving accuracy through combining the predictions of multiple classifiers in order to obtain a single classifier. Experience has shown that ensemble-based techniques such as bagging and boosting can be very effective for decision trees and neural networks [24, 21]. On the other hand, there has been less empirical testing with classifiers as logic programs.

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 $P_1, P_2, \ldots, P_N$ 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 classifiers one must address three different problems:

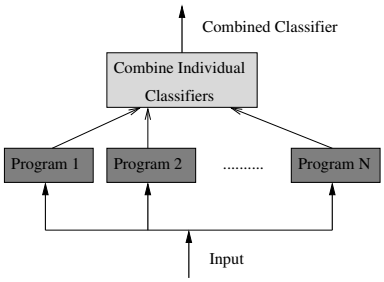

Fig. 1. An Ensemble of Classifiers.

- 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 different parts of the instance space [17, 22]. Obviously, the output of several classifiers is useful only if there is disagreement between them. Hansen and Salamon [14] 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.

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 producing such classifiers, described in the two following paragraphs.

One interesting aspect of Inductive Logic Programming is that the same learning algorithm may lead to quite different theories. More specifically, theories generated by seed-based ILP algorithms may heavily depend on the choice of the seed example. A natural approach to generate ensembles is to take advantage of this property of ILP systems, and combine the rather different theories that were generated just by choosing different sequences of seed examples. We call this approach *different seeds*.

We contrast the random choice of seeds with 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.

Table 1. Example of Bagging Training Sets.

Training Sets	Examples Included					
1	6	2	6	3	2	5
2	1	4	6	5	1	6
3	1	3	3	5	2	3
4	6	4	1	4	3	2
5	6	4	2	3	2	3
6	5	5	2	1	5	4

Table 1 shows six training sets randomly generated from a set with examples numbered from 1 to 6. We can notice that each bagging training set tends to focus on different examples. The first training set will have two instances of the second and sixth examples, while having no instances of the second and fourth example. The second example will have instead two occurrences of the first and sixth example, while missing the second and third example. In general, accuracy for each individual bagging classifier is likely to be lower than for a classifier trained on the original data. However, when combined, bagging classifiers can produce accuracies higher than that of a single classifier, because the diversity among these classifiers generally compensates for the increase in error rate of any individual classifier.

Therefore, the promise of bagging, and of ensembles in general, is that when classifiers are combined the accuracy will be higher than the accuracy for the original classifier. In our case, one particularly interesting result for bagging is that it is effective on "unstable" learning algorithms, that is, on algorithms where a small change in the training set may lead to large changes in prediction. We focus on bagging in this work because it is considered to be less vulnerable to noise than boosting algorithms [12] and it can take advantage of parallel execution.

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 [14]. Larger ensemble sizes have been proposed for decision trees, where gains have been seen up to 25 classifiers. In our experiments we decided to extend our analysis up to 100 classifiers.

The last problem concerns the combination algorithm. An effective combining scheme is often to simply average the predictions of the network [1, 7, 17, 18]. An alternate approach relies on a pre-defined *voting threshold*. If the number of 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.

3 Methodology

We use the ILP system Aleph [29] in our study. Aleph assumes (a) background knowledge B in the form of a Prolog program; (b) some language specification $\mathcal{L}$ 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 $\mathcal{L}$, 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. In the search for any single clause, Aleph selects the first 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.

We have elected to perform a detailed study on three datasets, corresponding to three non-trivial ILP applications. For each application we ran Aleph with random re-ordering of the positive examples and hence of seeds. We call this experiment *different seeds*. Next, we created bagged training sets from the original set, and called Aleph once for each training set. We call this experiment *bagging*. The number of runs of *different seeds* is the same as the number of bags.

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 clauselength, 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 $\mathcal{L}_i$, is 1 plus the maximum chaining depth of all variables that appear in previous literals $\mathcal{L}_j, j < i$. 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 to be the largest clause length that produced run times smaller than 1 hour on Intel 700 MHz machines, running Linux Red Hat 6.2. For our applications this value was either 4 or 5.

For each application, we ran experiments with different lower bounds on the minimum accuracy of an acceptable clause (`minacc`). We chose the values of 0.7,

0.9 and 1.0. In order to keep running times feasible, we first ran our datasets at least 5 times with the three different minaccs, and also with clause length varying from 4 to 10. We then chose the parameters that allowed Aleph to run at most for one hour (on Intel 700 MHz machines). It happened that for all minaccs, the clause length that produced runtimes less than or equal to one hour was the same, though it varied from one application to another. For each application we thus will vary our `minacc` settings among 0.7, 0.9 and 1.0, and we use the appropriate `maxclauselength`.

Next, we organise our discussion of methodology into (a) experimentation and (b) evaluation.

Experimentation. Our experimental methodology employs five-fold cross-validation. For each fold, we consider ensembles with size varying from 1 to 100. The *minacc* parameter used to generate the component theories in an ensemble, and the voting threshold used for the ensembles, are tunable parameters. These parameters are tuned within each fold, using only the training data for that fold. Moreover, rather than holding out a single tuning set from the training data for a fold, we perform 4-fold tuning within the training set. The parameter combination that yields the highest accuracy during this "inner" 4-fold cross-validation on the training set is then used on the entire training set for that fold. The resulting theory is then tested on the test set for that fold. The process is repeated for each of the five folds, and the results are merged in the standard way for cross-validation.

Next, we present the details of the experimental setup, starting with tuning. As explained, our goal in the tuning phase is to estimate what is the best parameter setting (minacc, voting threshold) for the ensembles, in order to use them later during the training/test phase. Tuning proceeds in two steps. First, we repeatedly call Aleph to generate all the theories we need to construct the different ensembles. Because the ILP runs are time-consuming, we do not repeat the ILP runs themselves to learn entirely new theories for each different ensemble size. Rather, for each tuning fold and minacc parameter, we initially learn 100 theories using *different seeds* methodology and 100 theories using *bagging*. Then, for either ensemble approach, and for each ensemble size s between 1 and 99 we randomly select s different theories from the 100. We next use these theories to generate ensembles, and evaluate the results. 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.

Tuning thus requires 12,000 theories per application: one theory for *bagging* plus one theory for *different seeds*, times 5 test set folds, times 4 tuning folds, times 3 minacc values, times the 100 different theories we create for producing ensembles.

Once the 12,000 theories are generated, two tables are produced. The first one, *minaccs_for_rocs*, is used to calculate the ROC curves and contains the best minacc for each ensemble size and for each voting threshold. The second one, *best_thresholds* is used to obtain the accuracies and contains the best combination

of minacc and voting threshold for each ensemble size. This second table is a subset of *minaccs_for_rocs*.

We next move to the 5-fold cross-validation by doing a training/test per fold, per each minacc. First, we produce 600 theories per fold: one for bagging plus one for *different seeds*, times 3 minaccs, times the 100 different theories used to create ensembles. We are now ready to evaluate the ensembles.

Evaluation. For the evaluation phase (b), we used two techniques to evaluate the quality of the ensembles. First, we studied how average accuracy varies with ensemble size. We present accuracy as the average between accuracy on the positive examples and accuracy on the negative examples.

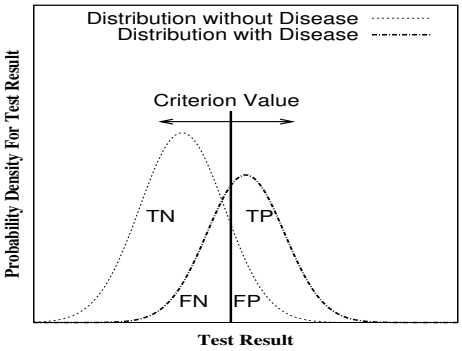

Fig. 2. Example of Probability Density Functions for Two Populations: with Disease, without Disease.

Second, we studied Receiver Operating Characteristic (ROC) curves for the ensembles. Provost and Fawcett [23] have shown how ROC curve analysis [20, 35] can be used to assess classifier quality. When we consider the results of a particular test in two populations, say positive and negative examples, we will rarely observe a perfect separation between the two groups. Indeed, the distribution of the test results can overlap, as shown in Figure 2. For every possible cut-off point or criterion value we select to discriminate between two populations, there will be some cases with the classifier correctly reporting positive examples to be positive (TP = True Positive fraction), but some cases incorrectly reported negative (FN = False Negative fraction). On the other hand, some negative examples will be correctly classified as negative (TN = True Negative fraction), but some negative examples will be classified as positive (FP = False Positive fraction).

In an ROC curve the true positive rate (sensitivity, or $\frac{TP}{TP+FN}$) is plotted against the false positive rate (100-specificity, or $\frac{FP}{FP+TN}$) for different cut-off points. Each point on the ROC plot represents a sensitivity/specificity pair corresponding to a particular decision threshold. A test with perfect discrimination (no overlap in the two distributions) has a ROC plot that passes through the

upper left corner (100% sensitivity, 100% specificity). Therefore the closer the ROC plot is to the upper left corner, the higher the overall accuracy of the test [35].

When we have many ROC curves to be analysed, we can look instead to the area under those curves. The value for the area under the ROC curve can be interpreted as follows: an area of 0.84, for example, means that a randomly selected individual from the positive group has a test value larger than that for a randomly chosen individual from the negative group 84% of the time. When the variable under study cannot distinguish between the two groups, i.e. where there is no difference between the two distributions, the area will be equal to 0.5 (as is the case when the ROC curve coincides with the diagonal). When there is a perfect separation of the values of the two groups, i.e. there is no overlapping of the distributions, the area under the ROC curve equals 1 (the ROC curve will reach the upper left corner of the plot).

We wish to test the effectiveness of different sizes of ensembles, from 1 to 99. 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.

Accuracies across the folds are obtained by averaging the sum of all positives and negatives for every fold. Areas across the folds are obtained by simply averaging areas computed for each ensemble size. ROC curves across the folds are obtained by averaging the rates of positives and negatives of each fold.

All experiments were performed using Condor, a tool for managing heterogeneous resources developed by the Condor Team at the UW-Madison [2]. Without the utilisation of such a tool, our experiments would have taken years to be concluded. Our jobs occupied about 53,380 hours of CPU, with an average peak of 400 jobs running simultaneously on Intel/Linux and Sun4u/Solaris machines.

3.1 Benchmark Datasets

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

Carcinogenesis. Our first application concerns the prediction of carcinogenicity test outcomes on rodents [30]. 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; experience suggests that a fairly large search space needs to be examined to obtain a good clause.

Smuggling. Our second dataset concerns data on smuggling of some materials. The key element of our data is a set of smuggling *events*. Different events may be related in a variety of ways. They may share a common location, they may involve the same materials, or the same person may participate. Detailed data on people,

locations, organisations, and occupations is available. The actual database has over 40 relational tables. The number of tuples in a relational table vary from 800 to as little as 2 or 3 elements.

The ILP system had to learn which events were *related*. We were provided with a set of related examples that we can use as positive examples. We can assume all other events are unrelated and therefore compose a set of negative examples. We assume *related* is comutative. Therefore we changed Aleph to assume `related(B,A)` if `related(A,B)` was proven, and vice-versa.

The smuggling problem is thus quite challenging in that it is a heavily relational learning problem over a large number of relations, whereas most traditional ILP applications usually require a small number of relations.

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 our ILP system.

Table 2. Datasets Characteristics.

	Dataset Sizes	Max Clause Length
Carcinogenesis	182+/148-	4
Smuggling	143+/517-	5
Protein	172+/690-	5

Table 2 summarises the main characteristics of each application. The second column corresponds to the size of the full datasets, where `P+/N-` represents number of positive examples and number of negative examples. Bags are created by randomly picking elements, with replacement, from the full dataset. Therefore the number of positives or negatives of each bag are not the same as of the full dataset used for *different seeds*, although the total size is the same. The second column indicates the clause length used for each dataset. The test sets for each 5-fold cross-validation experiment is obtained by a block distribution of the full dataset. For example, application Carcinogenesis will have 5 positive test sets of sizes: 36, 36, 36, 36 and 38. These test sets are not used during the tuning phase.

4 Results

This section presents our results and analyses the performance of each application. For each application we show the average accuracy for positives and

negatives, and the area under ROC curves built from 1 to 25 ensemble sizes. The results from 26 to 99 are essentially horizontal lines and are not shown. We report results for *different seeds* and *bagging*. We also show the ROC curve for an ensemble size of 25.

The accuracies presented in the graphs are averaged across all folds, and for each ensemble size, a different voting threshold and minacc are used. This combination of voting threshold and minacc is the one that produced the best accuracy during the tuning phase. For clarity's sake, these parameter values are not shown in the curves.

The areas under the ROC curves were computed by (1) computing an ROC curve, per fold, for each ensemble size using the theories learned for the best pair *<minacc, voting threshold>*, (2) computing the area under each ROC curve, and (3) averaging the areas for each ensemble size across the folds.

Figure 3 shows the average accuracies for the three applications, for *different seeds* and *bagging*, when varying the ensemble sizes. Figure 4 shows the areas under the ROC curves, averaged across five folds, for the three applications, when varying the ensemble sizes. Figure 5 shows ROC curves at ensemble size 25 for every application.

The results show that ensembles do provide an improvement both in accuracies and in ROC areas. Most of the improvement is obtained for smaller ensemble sizes, up to 5 or 10 elements. Performance does not seem to benefit much from using larger sizes.

The best results were obtained in the smuggling application, with *bagging* and *different seeds* obtaining similar performance. The most irregular application is carcinogenesis. We discuss the individual applications in more detail in the next sections.

4.1 Carcinogenesis

Single-theory accuracy results for carcinogenesis are not very good. On average, accuracy for a single theory is around 60% for *different seeds* and 59% for *bagging*. Our results show that ensembles can improve accuracy to around 64%. Unfortunately, our results also show huge variations, as we discuss next.

Figure 3(a) shows the average among the accuracy curves for the five folds. At first, the curves show that both *bagging* and *different seeds* obtain significant improvements. As ensemble size grows, *different seeds* tends to obtain better results, whereas *bagging* on average tends to achieve performance closer to the single-theory case. Both curves show a number of peaks.

The maximum average accuracy for *different seeds* is 64.5%, which represents a significant improvement over the single-theory accuracy. Studying *different seeds* in more detail, we found the system tends to be pretty reasonable at classifying positive examples. It achieves a maximum of 78.7% acceptance rate for *different seeds*, at ensemble size 31. It does not perform so well at rejecting negative examples: the best result is a minimum probability of 48.3% of rejecting a negative example, at ensemble size 12.

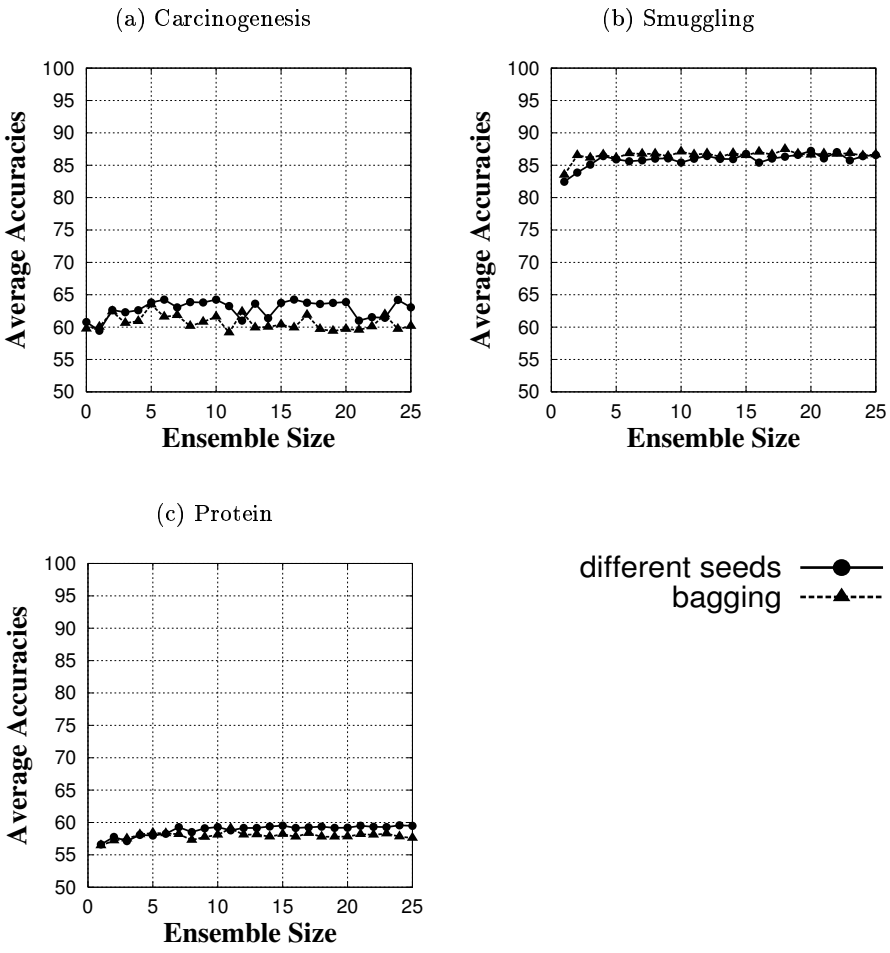

Fig. 3. Average Accuracies for the Three Applications.

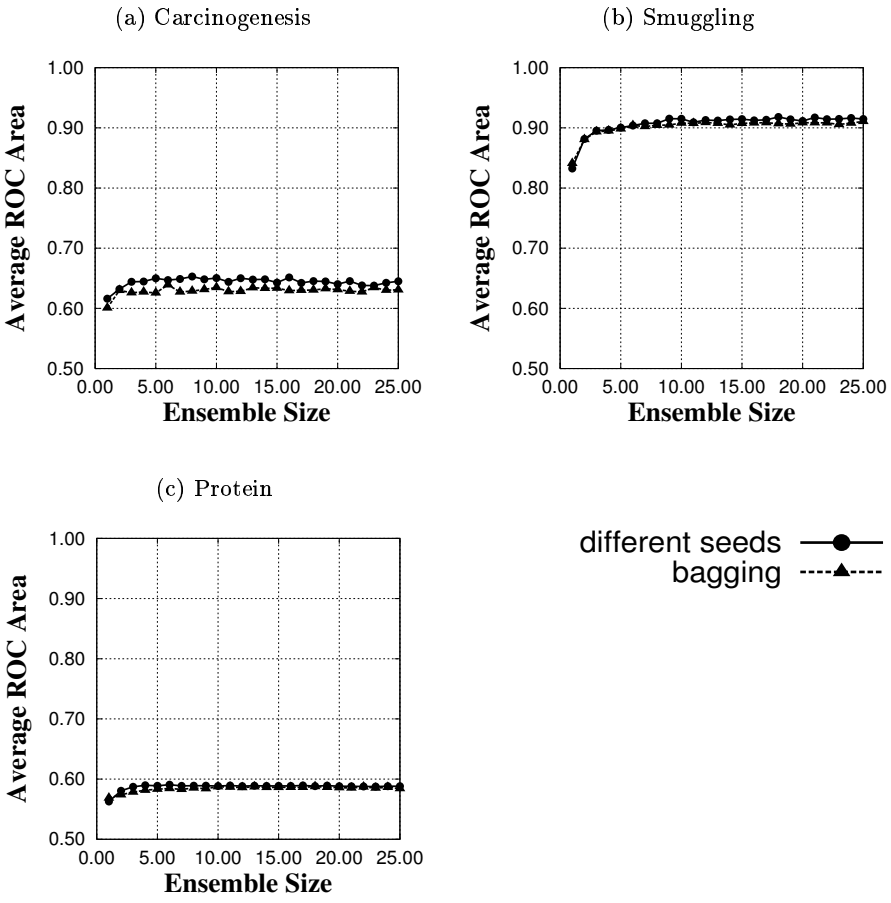

Fig. 4. Areas under the ROC curves for the Three Applications.

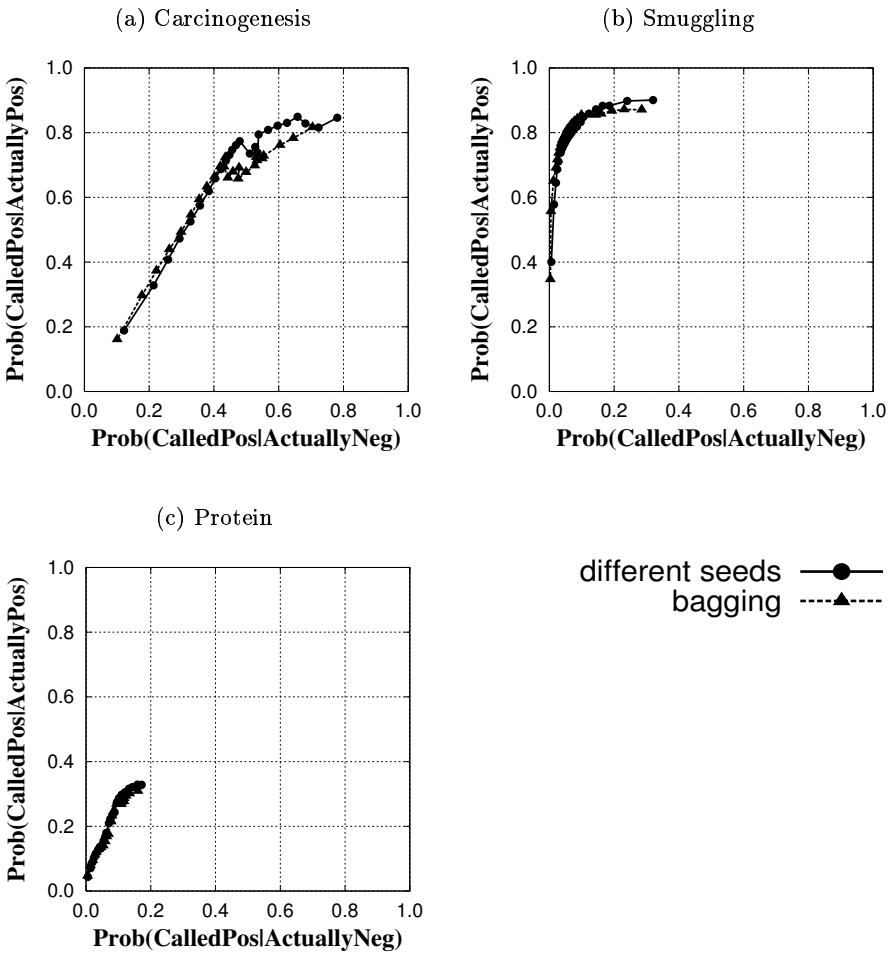

Fig. 5. ROC curves at N=25 for the Three Applications.

Different seeds is particularly interesting in that it shows several spikes, which are most obvious in the accuracy curve. This effect is caused by our tuning algorithm that has some difficulties with a very unstable application, such as carcinogenesis. More precisely, study of the data shows that often the tuning algorithm will choose the same minacc in a row for a series of consecutive ensemble sizes in a fold, and then all of a sudden select a different minac, and immediately return to choosing the initial parameter. The points where there is an abrupt change of parameters are usually the points where we have spikes.

The accuracy curve for *bagging* has several spikes on smaller ensembles, and then stabilises rather quickly. Again, most of the variation is caused by the choice of parameters. Interestingly enough, whereas most spikes in *different seeds* are negative, in this case most spikes are improvements. This suggests that accuracy results may be suffering from a wrong choice, either of thresholds or of Aleph's minimal training accuracy parameter.

Next, we look at the areas under the ROC curves as ensemble size varies from size 1 to 25. The area under a ROC curve gives a good estimate of classifier quality.

Figure 4(a) compares the ROC areas for *different seeds* and for *bagging*. We can notice that both techniques obtain a significant improvement with the smaller ensembles. *Bagging* stabilises very quickly, though, whereas *different seeds* obtains improvements up to ensemble size 5. As a result, *different seeds* has a somewhat better result than *bagging*.

ROC curves provide a more detailed picture of the behaviour of this application concerning rates of true positives against false positives. We chose to plot an ROC curve for ensemble size of 25, the largest ensemble size for which we present other results. Figure 5(a) shows an ROC curve for ensemble size 25, for the application Carcinogenesis, for *different seeds* and *bagging*, when varying the voting threshold from 1 to 25. The curves show very clearly the spikes caused by the different minacc chosen for each point.

The curves show large variations for small ensemble sizes. This corresponds to choosing different minaccs. Both *different seeds* and *bagging* can achieve a very good improvement on positive examples. However, at a cost of increasing the rate of false positives. The best result for positives, for *different seeds* is achieved at voting threshold 1, with acceptance rate of 88%. *Bagging* also achieves its best performance on positives at threshold 1, but it performs a little worse than *different seeds* achieving maximum of 85% of acceptance rate. As the voting threshold increases, the chance of better classifying a positive or negative example decreases. Note that the voting threshold increases as we decrease along the X axis, but not in a consecutive order of points in the ROC curve. For example, at ensemble size 9, the false positive rate is 0.66, and the true positive rate is 0.81, while at ensemble size 10, the false positive rate is 0.70 and the true positive rate is 0.80.

Our main conclusion is that *different seeds* performs better for small thresholds, that is, it can recognise most positive examples. *Bagging* tends to perform better for large thresholds, that is, it is better at recognising negative examples.

4.2 Smuggling

The smuggling problem is quite challenging in that it is a heavily relational learning problem over a large number of relations, whereas most traditional ILP applications usually require a small number of relations. It was therefore quite interesting to find that Aleph can achieve quite good accuracy for this problem. We found average accuracy to be about 82% for *different seeds* and 83% for *bagging*, for a single theory. Single accuracy for negative examples for a single theory is around 89% for *different seeds* and 91% for *bagging*, while accuracy for positive examples are around 76%, for both *different seeds* and *bagging*.

Of course, it would be quite nice to achieve an even better accuracy. Figure 3(b) shows the variation of accuracy averaged across folds, as we range the size of the ensembles between 1 and 25. Accuracy for both curves increases quickly for smaller ensembles and then is largely stable as we increase ensemble size. *Different seeds* and *bagging* achieve a maximum average accuracy of around 87%, with *different seeds* behaving slightly better than *bagging* for larger ensemble sizes. Our results thus correspond to a significant improvement over the single-theory accuracy. Accuracies stabilise at around 92%, for both *different seeds* and *bagging*, at classifying negative examples, and at around 82% for both *different seeds* and *bagging*, at classifying positive examples.

This application illustrates the benefit of using *bagging* at smaller ensemble sizes, and stresses the advantage of using ensemble methods to improve the accuracy of a single theory.

Figure 4(b) shows the areas under the ROC curves from ensemble size 1 to 25 averaged across the five test set folds. The results on ROC areas are very impressive. Performance is excellent for both *different seeds* and *bagging*, with a small advantage for *different seeds*. Both curves show a substantial improvement up to size 10, and then stabilise. Also notice that *bagging* and *different seeds* achieve very similar results.

Figure 5(b) shows average of ROC points across five folds, for ensemble size 25 varying the threshold from 1 to 25. Both classifiers perform in much the same way. The combined classifier is very good at classifying negative examples: even in the worst case, it only misclassifies up to 30% of all negative examples. Results are also quite good on the positive examples, although some examples are never covered. *Different seeds* does have some advantage in this case. Last, notice that there are much less spikes than for Carcinogenesis: the tuning algorithm seems to perform quite well here.

4.3 Protein

The Protein application has average single-theory accuracy of around 56% for both *different seeds* and *bagging*. Figure 3(c) shows the average accuracies between positives and negatives for *different seeds* and *bagging*, as we increase the ensemble size from 1 to 25. *Bagging* and *different seeds* have very similar performance for this application. The improvement over the single-theory is between 2

and 3 percentage points, and does show that the ensemble methods can improve performance even in this case.

Different seeds achieves maximum average accuracy of 60% at ensemble size 47, while *bagging* achieves maximum accuracy of 59% at ensemble size 28.

In order to understand better what happens with this application we draw the ROC curve for ensemble size 25. Figure 5(c) shows the ROC curves for *different seeds* and *bagging* averaged across five folds. The performance of this application on positives improves as we increase the ensemble size for both *different seeds* and *bagging*. For low thresholds, *different seeds* does better for positives. The learned theories seem to have difficulty in generalising: we cannot cover most examples. This results in bad accuracy for positives, and in good accuracy for negatives. The results also show that most of the improvement happens for smaller ensembles.

Our analysis provides more insight when we look at the areas under the ROC curves, varying our threshold from 1 to 25. Figure 4(c) shows the areas under the ROC curves. The results essentially confirm what we obtained with accuracy. *Different seeds* and *bagging* have the same performance up to ensemble size 24. After that *different seeds* surpasses the performance of *bagging*.

As with the other applications, both ensemble methods improve performance over the single-theory.

5 Conclusions and Future Work

This work presents an empirical study of *bagging*, a well-known ensemble building mechanism, in the Inductive Logic Programming setting. In our approach, we use bagging to combine theories built from random variations of the original training set. We contrast bagging to *different seeds*, an approach where we always use the same training set, and randomly select different seeds to build the different theories in the ensemble. We evaluated *bagging* and *different seeds* with three non-trivial applications of ILP.

Our results show that ensembles built through *bagging* can indeed achieve a sizable improvement in performance, both measured through accuracy and through ROC curves. Most of the gain is achieved with ensembles of size up to 20. Exploiting different seeds can also achieve very good gains. In fact, simply using *different seeds* worked as well, or arguably better, than *bagging* in our experiments. We believe this is because *different seeds* learns theories using the whole set of examples. Our results confirm the advantages of using ensemble methods in ILP.

We believe that *bagging* and *different seeds* can have a substantial impact on ILP. We can often achieve an interesting improvement in performance, with little implementation work. Moreover, we found that accuracy may improve, even in the cases where ILP is obtaining very good results, as is the case of the dataset Smuggling. On the other hand, resulting theories are more complex and thus harder to understand.

We have thus far used *bagging* and *different seeds* with ensembles of theories. An interesting alternative we are researching is to use ensembles of clauses. As discussed before, *boosting* can also be employed to improve accuracy of classifiers by penalizing examples that are misclassified. One disadvantage of *boosting* is that it can not be as easily parallelisable as *bagging* or *different seeds*. We have been investigating a method to perform *boosting* in parallel. Last, we are investigating other tuning algorithms to improve the interaction between different settings (e.g., clause length, minimum clause accuracy) for the ILP search and the *bagging*/*different seeds* process.

Acknowledgments

This work was supported by DARPA EELD grant number F30602-01-2-0571, the NSF grant 9987841 and by NLM grant NLM 1 R01 LM07050-01. Vítor Santos Costa and Inês de Castro Dutra were partially supported by CNPq. We would like to thank the Biomedical Group support staff for their invaluable help with Condor. We also would like to thank Ashwin Srinivasan for his help with the Aleph system and the Carcinogenesis benchmark. Vítor Santos Costa and Inês Dutra are on leave from COPPE/Sistemas, Federal University of Rio de Janeiro.

References

1. E. Alpaydin. Multiple networks for function learning. In *IEEE International Conference on Neural Networks*, pages 9–14, 1993.
2. J. Basney and M. Livny. Managing network resources in Condor. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, Pennsylvania*, pages 298–299, Aug 2000.
3. 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.
4. 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.
5. 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.
6. L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.
7. L. Breiman. Stacked Regressions. *Machine Learning*, 24(1):49–64, 1996.
8. 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.
9. 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.
10. 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.
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. 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.
13. J. Graham, D. Page, and A. Wild. Parallel inductive logic programming. In *Proceedings of the Systems, Man, and Cybernetics Conference*, 2000.
14. L. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, October 1990.
15. 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.
16. R. King, S. Muggleton, and M. Sternberg. Predicting protein secondary structure using inductive logic programming. *Protein Engineering*, 5:647–657, 1992.
17. 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.

18. N. Lincoln and J. Skrzypek. Synergy of clustering multiple backpropagation networks. In *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 1989.
19. 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.
20. J. Metz. The epidemic in a closed population with all susceptibles equally vulnerable; some results for large susceptible populations and small initial infections. *Acta Biotheoretica*, 27:75–123, 1978.
21. D. W. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.
22. D. W. Opitz and J. W. Shavlik. Actively searching for an effective neural-network ensemble. *Connection Science*, 8(3/4):337–353, 1996.
23. F. J. Provost and T. Fawcett. Robust classification systems for imprecise environments. In *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 706–713, 1998.
24. J. R. Quinlan. Bagging, boosting, and c4.5. In *Proceedings of the 14th National Conference on Artificial Intelligence*, volume 1, pages 725–730, 1996.
25. J. R. Quinlan. Boosting first-order learning. *Algorithmic Learning Theory, 7th International Workshop, Lecture Notes in Computer Science*, 1160:143–155, 1996.
26. 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.
27. 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.
28. A. Srinivasan. A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery*, 3(1):95–123, 1999.
29. A. Srinivasan. *The Aleph Manual*, 2001.
30. A. Srinivasan, R. King, S. Muggleton, and M. Sternberg. Carcinogenesis predictions using ILP. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 273–287. Springer-Verlag, 1997.
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 *The Twelfth 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.
35. M. Zweig and G. Campbell. Receiver-operative characteristic. *Clinical Chemistry*, 39:561–577, 1993.