Preprint
UCRL-JC-142268

Creating Ensembles of Decision Trees through Sampling

C. Kamath, E. Cantú-Paz

This article was submitted to
International Conference on Machine Learning, Williams College, MA, June 28- July 1, 2001

February 2, 2001

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

This work was performed under the auspices of the United States Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Creating Ensembles of Decision Trees Through Sampling

Chandrika Kamath KAMATH2@LLNL.GOV
Erick Cantú-Paz CANTUPAZ1@LLNL.GOV
Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA 94551

Abstract

Recent work in classification indicates that significant improvements in accuracy can be obtained by growing an ensemble of classifiers and having them vote for the most popular class. This paper focuses on ensembles of decision trees that are created with a randomized procedure based on sampling. Randomization can be introduced by using random samples of the training data (as in bagging or arcing) and running a conventional tree-building algorithm, or by randomizing the induction algorithm itself. The objective of this paper is to describe our first experiences with a novel randomized tree induction method that uses a subset of samples at a node to determine the split. Our empirical results show that ensembles generated using this approach yield results that are competitive in accuracy and superior in computational cost.

1. Introduction

Ensembles of classifiers, also referred to as forests in the case of decision tree classifiers, are increasingly gaining acceptance in the data mining community. This is prompted by many factors, including a significant improvement in accuracy (Breiman, 1996a; Freund & Schapire, 1996; Quinlan, 1996; Bauer & Kohavi, 1990), the potential for on-line classification of large databases that do not fit into memory (Breiman, 1996b), and the ease with with these techniques lend themselves to scalable parallelization (Hall et al., 2000). There are different ways in which ensembles can be generated, and the resulting output combined to classify new instances. Implicit in some of these ensembles is the concept of randomness that is introduced either through the randomization of the training set, or the randomization of the classifier itself.

In this paper, we discuss one particular approach to randomization, namely the use of random sampling to determine the split made at each node of a decision tree. As the split made at a node is likely to vary with the sample selected, this technique can be used to generate ensembles of trees. Our objective is to show that this approach not only improves the accuracy of the classifier like other approaches to ensembles, but does so at a relatively low cost. Our experimental studies with public domain datasets indicate that the accuracy obtained is relatively insensitive to the percentage of instances sampled at a node. This allows us to lower the cost of generating each tree in the ensemble, thus ameliorating the cost of generating the ensemble of trees.

The paper is organized as follows: In Section 2, we discuss the various ways in which we can generate ensembles of classifiers. Next, in Section 3 we describe the use of sampling to introduce randomization in the induction of decision trees. We describe our experimental results in Section 4 and conclude in Section 5 with a summary and ideas for future work.

2. Creating Ensembles of Classifiers

There is considerable diversity in the way in which ensembles of classifiers can be created (Dietterich, 2000a). In this section, we briefly discuss some of the more popular approaches.

2.1 Changing the Instances Used for Training

In this approach, each classifier in the ensemble is generated using a different sample of the training set. There are several ways in which this can be accomplished:

- Bagging: In this approach, a new sample of the training set is obtained through bootstrapping with each instance weighted equally (Breiman, 1996a). This technique works very well for un-

stable algorithms such as decision trees and neural networks, where the classifier is sensitive to changes in the training set and significantly different classifiers are created for different training sets. In bagging, the results of the ensemble are obtained by using a simple voting scheme. Each classifier can be generated independent of the other, and randomization is introduced through the random sampling used to create each sample of the training set.

- Boosting: In this case, a new sample of the training set is obtained using a distribution based on previous results (Freund & Schapire, 1996). Unlike the Bagging algorithm, which uniformly weights all the instances in the training set, Boosting algorithms adjust the weights after each classifier is created to increase the weights of misclassified instances. This essentially implies that the training sets for the classifiers have to be created in sequence, instead of in parallel, as in the case of Bagging. The different weights for the ensembles can either be directly incorporated into the classifier by working with weighted instances, or be applied indirectly by selecting the instances with a probability proportional to their weights. Further, in boosting, the results of the ensemble are obtained by weighting each classfier by the accuracy on the training set used to build it. As a result, better classifiers have a greater contribution to the end result than the poorer classifiers. There are several variants of Boosting which differ in the way the instances are weighted, the conditions under which the algorithm stops, and the way in which the results from the ensemble are combined (Breiman, 1998; Bauer & Kohavi, 1990; Freund & Schapire, 1996).

- Pasting: In this approach, the ensemble of trees is grown using a sub-sample of the entire training set (Breiman, 1996b). This technique has been shown to be useful when the entire training set is too large to fit into main memory.

2.2 Changing the features used in training

In this approach, each new classifier is created using a subset of the original features. For example, Ho (1995) has illustrated the use of this technique with decision trees and Cherkauer (1996) has used it with neural networks. This approach tends to work only when the features are redundant, as poor classifiers could result if some important feature is left out. The approach used to select the features could introduce randomization to the procedure.

2.3 Introducing randomness in the classifier

Unlike the previous techniques, where the input to the classifier is changed to generate the ensemble, it is possible to create the ensemble by changing the classifier itself. For example, in neural networks, the initial weights are set randomly, thus creating a new network each time. In decision trees, instead of selecting the best split at a node, one can randomly select among the best few splits to create the ensemble (Dietterich, 2000b). Our approach to randomizing the classifier is to use only a sample of the instances at a node of a decision tree in order to make the decision. We explore this approach further in the next section.

3. Sampling in Tree Induction

The idea behind our approach is very simple: at each node of the decision tree, instead of using all instances to determine the best split, we use only a random sample of the instances. This randomized procedure results in different trees which can be combined in ensembles. However, an efficient implementation of the approach can be non-trivial.

In a decision tree, the split at each node is obtained by first sorting each of the continuous features and then selecting a split point that optimizes a certain criterion (Breiman et al., 1984). The more efficient implementations of decision trees sort all the features once at the beginning. Each feature is associated with its instance identification to keep track of which feature in the sorted list is associated with which instance. Then, at each node, using an appropriate split criterion, the optimal split point is found for each feature. The best split across all features is chosen as the split point at the node. When the instances at a node are split among the children nodes, we want to maintain the sorted order of each feature for the purpose of efficiency. In this scenario, if we want to split the instances at a node using a split based on only a sample of the total instances, we have two options:

- **Sample with sorting:** In this case, we randomly select p percentage of the n instances for a feature. The sampling can be done with or without replacement. But, it results in the sampled set of feature values being unsorted. Therefore, after the sampling at a node on each feature, an additional sort is needed.

- **Sample without sorting:** In order to maintain the sorted order of the sampled features at a node, one option would be to divide the instances into $n * p/100$ parts, each part containing $100/p$

Table 1. Benchmark data sets used for studying the effect of sampling on the generation of ensembles.

DATA SET	# TRAINING INSTANCES	# TEST INSTANCES	# CLASSES	# DISCRETE ATTRIBUTES	# CONT ATTRIBUTES
GLASS	214	-	7	-	9
BREAST CANCER	699	-	2	-	9
PIMA INDIAN DIABETES	768	-	2	-	8
GERMAN	1000	-	2	13	7
SATELLITE IMAGE	4435	2000	6	-	36
LETTER RECOGNITION	16000	4000	26	-	16

instances, and randomly select an instance from each part. The sampled set would then remain ordered and no sorting is required.

Once we have decided how to sample the instances, we can either use the same sampled set of instances for all the features, or obtain a new sample for each feature. In our work, we have chosen the latter approach. Regardless of how the samples are obtained at each node, the process of sampling introduces randomization in the induction of the decision tree. This can be used to generate an ensemble of trees and, by appropriately combining the results of these trees, we can obtain classifiers with better generalization accuracy.

4. Experimental Results

In this section, we describe the results of our experiments with the generation of ensembles using the sampling approach. We conducted our experiments on some of the larger datasets available from the repository at the University of California at Irvine (UCI Dataset, 2001). The details of the data sets used are summarized in Table 1. None of the datasets considered have any missing attributes. Note that except for the Glass dataset, the remainder of the data sets are relatively large. This choice was intentional as we believe that the use of sampling is most beneficial for large datasets.

In some cases, the datasets were available with a test set. In such cases, we performed our experiments 10 times to account for the randomization and averaged the results. In cases where there was no test set available, we used 10-fold cross-validation, and averaged the results over 10 runs. Based on the observation by Breiman (1996a) that most of the improvement in bagging is evident within ten replications, we used 10 trees to create the ensemble. This would give us the performance improvement bought by a single order of magnitude increase in the number of trees. The results of the ensemble were combined by a simple unweighted voting.

Table 2. Test error on the benchmark datasets for the generation of a single tree.

DATA SET	ERROR (%)
GLASS	32.81
BREAST CANCER	5.17
PIMA INDIAN DIABETES	27.88
GERMAN CREDIT	29.95
SATELLITE IMAGE	15.75
LETTER RECOGNITION	30.62

We did not use pruning in the generation of the trees as we expected the use of ensembles to eliminate the overfitting. We also used the same sampling percentage at each node of the tree, though it is possible to vary this value at each node. In addition, we stopped the sampling when the number of samples was less than twice the number of features at a node. The sub-tree at this node was then generated without using any sampling. This was done to ensure that there were enough samples at a node relative to the size of the hypothesis space, so that we could obtain an accurate classifier.

The decision trees in our experiments were generated using the Gini index as the split criterion (Breiman et al., 1984). The test error averaged over multiple runs for the case of a single tree is indicated in Table 2. This gives us the baseline results for comparison with the results obtained using the ensembles.

4.1 Effects of Sorting in Sampling

For our initial experiments, we first selected the second option in Section 3 where the sampling was done without any need for sorting. We believed that the compute time required would be less as there was no sorting on each sample for each feature at each node. In Table 3, we present the results of the accuracy of

Table 3. Test error on three datasets illustrating the effects of sampling with and without sorting

Percentage Sampled	Glass		Pima Indian Diabetes		German Credit	
	Error w/o sorting	Error w/ sorting	Error w/o sorting	Error w/ sorting	Error w/o sorting	Error w/ sorting
0.9	34.38	27.33	26.96	24.35	30.01	28.55
0.8	34.71	26.62	26.59	24.63	30.27	28.59
0.7	37.67	27.47	27.26	25.02	30.35	28.30
0.6	34.43	26.48	28.09	24.72	30.18	28.48
0.5	28.57	27.48	24.67	24.93	28.56	28.68
0.4	31.14	28.00	24.50	24.59	28.80	28.43
0.3	29.52	27.33	24.47	24.65	28.70	27.87
0.2	27.24	27.33	24.59	24.90	28.09	27.51
0.1	28.95	30.14	24.67	25.04	26.77	27.17
0.05	32.81	32.81	25.11	25.04	27.14	26.62
0.01	32.81	32.81	27.88	27.88	29.95	29.95

the ensembles as we varied the percentage of instances sampled at each node of the trees. We present our results for three of the smaller datasets: Glass, Pima Indian diabetes, and German Credit. The error is given for both the case where we sample without sorting and with sorting.

We observe from the table that for a large sample size, the error with sorting was less than the error without sorting. This observation held across the three datasets and the different percentages of the instances sampled at a node. When we investigated this further, we found that, in the case without sorting, the way the instances were divided into parts had an effect on the results. Recall that to obtain a sample at p percentage, we divide the instances n at a node into $n*p/100$ parts, each part containing $100/p$ instances. However, when $p > 50\%$, this means that the number of instances in each part is 1, with the possible exception of the last part, which includes all remaining instances. For example, suppose we have $n = 800$ instances at a node, and we want a $p = 80\%$ sample. This will imply that of the 640 instances to be selected from the 640 parts, the first 639 instances will come from parts that are only a single instance wide. The last sampled instance will be from a part that is $(800 - 640) = 160$ wide, thus resulting in a biased sample and inferior results.

Based on this observation, for the remainder of our experiments, we chose to do the explicit sorting for sampling percentages higher than 50% and opted for the non-sorting version of sampling for lower values of percentage of instances sampled. This enabled us to use the more efficient implementation without any detrimental effect in the generalization error.

4.2 Accuracy for Sampled Ensembles

An important observation from Table 3 is that the accuracy of the results appears to be somewhat independent of the percentage of instances sampled at a node. In Table 4 we present the results for the remainder of the datasets: Breast Cancer, Satellite Image, and Letter Recognition. We notice again that the accuracy is roughly constant as we vary the percentage sampled, with the exception of the results for a sampling percentage of 40%, where the results are less accurate.

In order to explain this behavior, we took a second look at how we split the instances at a node of the tree. For each feature, we need to identify a value such that a split point on that feature at that value will optimize the split criterion. By sampling the instances at a node without sorting, we are essentially selecting a sample that is close to uniformly distributed across the instances. This ensures a split point that is close to the optimal split point. In the case of sampling without sorting, in the worst case, the split point selected could be as far from the true split point as the width of the part from which it is selected. When we introduce randomness through sampling, it is likely that the trees created in the ensemble are very different. However, the decision boundaries, that is the hyperplanes separating the classes, for all the trees will be very close to each other. In contrast with a single tree, the decision boundary for the ensemble will be "soft", leading to a better generalization error.

Note that as we reduce the percentage of instances sampled, the error increases again, some times reverting to the value in Table 2. This is because at some point, the sample size at the base node of the tree becomes small enough such that the number of samples

Table 4. Test error on three datasets illustrating the effects of sampling in the creation of ensembles

Percentage Sampled	Breast Cancer	Satellite Image	Letter Recognition
0.9	3.56	11.64	11.62
0.8	3.63	11.55	11.73
0.7	3.37	11.67	11.58
0.6	3.26	11.56	11.97
0.5	4.03	12.36	13.05
0.4	4.72	13.37	35.76
0.3	4.04	12.06	19.84
0.2	3.76	11.84	12.71
0.1	3.67	11.65	13.25
0.05	3.56	11.83	12.50
0.01	5.17	15.75	13.79

is at least twice the number of features. As explained earlier, at this point, each tree in the ensemble becomes identical to the tree created without sampling, and the error is the same as that for a single tree. For sample percentages just a little higher than this value, the nodes at the higher levels of the tree use sampling, but those at the lower levels take the "no-sampling" path, resulting in a higher error value for the ensembles, but lower than the error for a single tree. This observation could be used as a rough rule of thumb to determine the smallest percentage of the samples to use in creating the ensembles.

In terms of the accuracy obtained by the our ensembles relative to other similar approaches, our results are competitive with those obtained through Boosting and Bagging (Freund & Schapire, 1996; Quinlan, 1996).

4.3 Computational Costs

We would expect that the use of sampling in the tree induction would reduce the total time required in comparison with tree induction without sampling. This is because fewer instances have to be considered in determining the split value at each node of the tree. Further, the process of sampling without sorting adds little to the overhead due to the sampling itself. Our initial timing results are presented in Table 5. The times given are in seconds on a 800MHz Pentium III system with 512MB of memory. Since we generate 10 trees in the ensemble, we considered a sampling rate of 0.1. The comparison is between the average of 10 runs of the single tree generated without sampling vs. 10 runs of the ensemble. We compare the total time for training and testing, though our timing results indicate that a large part of this is training time.

We note that the ensembles are slower by a factor of 5 and 8 for the letter recognition and satellite datasets,

Table 5. Timing results (in seconds) comparing 10 runs of the training/testing of a single tree without sampling vs. an ensemble of 10 trees with sampling at 10%.

	Satellite Image	Letter Recognition
Single Tree	61 s	262 s
Ensemble	488 s	1279 s

respectively. If we had not used any sampling, the time for the ensembles of 10 trees would have been a factor of 10 greater. Our timing results show that a substantial amount of time in the generation of the decision tree is in the initial sorting of the features. Since the same training set is used for all the trees in the ensemble, we expect that the time for the ensembles can be reduced further by doing the sort only once for all the trees. This approach to reducing the cost of ensembles is not applicable in the case of bagging or boosting, the training set varies across the classifiers in the ensemble and the sorting has to be done in each case.

5. Summary and Future Work

In this paper, we have introduced an approach to the generation of ensembles where randomization is introduced in the decision tree induction through the use of sampling. Our early experimental results using public domain datasets show that this is a promising approach, both in terms of accuracy and computational cost. However, much remains to be done. We want to improve the performance of the ensembles by sorting the features of the training set only once for each tree in the ensemble. Further, we are interested in seeing if the results would be improved by using more trees in

the ensemble. In addition, we plan to conduct studies with additional datasets to see if the results carry over to these datasets as well.

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

References

Bauer, E., & Kohavi, R. (1990). An empirical comparison of voting classification algorithms: Bagging boosting and variants. *Machine Learning, 36*, 105–139.

Breiman, L. (1996a). Bagging predictors. *Machine Learning, 26*, 123–140.

Breiman, L. (1996b). *Pasting bites together for prediction in large data sets and on-line* (Technical Report). Statistics Department, University of California, Berkeley. ftp.stat.berkeley.edu/pub/users/breiman/pastebite.ps.Z.

Breiman, L. (1998). Arcing classifiers. *Annals of Statistics, 26*, 801–824.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and Regression Trees.* Boca Raton, Florida: Chapman and Hall/CRC Press.

Cherkauer, K. (1996). *Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks* (Technical Report). Working notes of the AAAI Workshop on Integrating Multiple Learned Models. http://www.cs.fit.edu/ imlm/.

Dietterich, T. (2000a). Ensemble methods in machine learning. *Proceedings of the First International Workshop on Multiple Classifier Systems* (pp. 1–15). Springer Verlag.

Dietterich, T. (2000b). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning, 40*, 139–158.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference* (pp. 148–156).

Hall, L., Bowyer, K., Kegelmeyer, W., Moore, T., & Chao, C. (2000). Distributed learning on very large data sets. *Workshop on Distributed and Parallel Knowledge Discover, in conjunction with KDD2000.*

Ho, T. K. (1995). Random decision forests. *Proceedings of the 3rd International Conference on Document Analaysis and Recognition* (pp. 278–282).

Quinlan, J. (1996). Bagging, boosting, and C4.5. *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (pp. 725–730). AAAI Press and MIT Press.

UCI Dataset (2001). UCI Knowledge Discovery in Databases Archive. http://kdd.ics.uci.edu/ .