Preprint
UCRL-JC-142268-REV-1

Creating ensembles of decision trees through sampling

C. Kamath and E. Cantú-Paz

This article was accepted for publication in the Proceedings of the 33-rd Symposium on the Interface: Computing Science and Statistics, Costa Mesa, California, June 13-16, 2001

August 15, 2001

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

Creating Ensembles of Decision Trees Through Sampling

Chandrika Kamath
Erick Cantú-Paz
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
P.O. Box 808, L-561
Livermore, CA 94551
kamath2@llnl.gov, cantupaz1@llnl.gov

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 boosting) 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 sub-sample of instances 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 to boosting and bagging.

1 Introduction

Ensembles of classifiers, sometimes 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 [2, 11, 14, 1], the potential for on-line classification of large databases that do not fit into memory [3], and the ease with these techniques lend themselves to scalable parallelization [12]. There are different ways in which ensembles can be generated, and the resulting output combined to classify new instances. Implicit in many 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 [8]. 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 [2]. This technique works very well for unstable 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 [11]. 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 [4, 1, 11].

- Pasting: In this approach, the ensemble of trees is grown using a sub-sample of the entire training set [3]. 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, this technique has been used with decision trees in [13] and with neural networks in [7]. 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 Changing the output targets

In [10], the authors describe a technique called error correcting output coding which can be applied to a problem with many output classes. The problem is first reduced to a two-class problem by randomly partitioning the classes into two, which are assigned labels 0 and 1. A classifier is created with this relabeled data. The process is repeated, creating a new classifier for each random partitioning of the original set of classes. To classify an unseen instance, each classifier assigns a label to the instance. If the label assigned is 0 (1), each class that was relabeled as a 0 (1) for that classifier, gets a vote. The class with the maximum number of votes is assigned to the instance.

2.4 Introducing randomness in the classifier

Unlike the previous techniques, where the input or output 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 [9]. Another approach would be to randomly select the features used to determine the split at each node of the tree [5]. 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. As the split made at a node is likely to vary with the sample selected, this technique results in different trees which can be combined in ensembles. We explore this approach further in the next section.

3 Sampling Instances at a Node in Tree Induction

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 [6]. The more efficient implementations of decision trees sort all the features once at the beginning. 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, the sorted order of each feature must be maintained for the purpose of efficiency.

There are several different ways in which we can sample the instances at a node. In our work, we use a new sample for each feature, though one can use the same set of sampled instances for all features. We randomly select (with replacement) a fraction p of the n instances at a node to form our sample. We could instead use an explicit number of instances at a node; however, it might be difficult to accurately pre-select this number for each dataset. We also used the same sampling percentage at each node of the tree, though this value can be varied as well. In addition, we

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

Data set	# training (test) instances	# classes	# discrete attributes	# cont attributes
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

must stop the sampling when the number of instances at a node is "small", to ensure enough samples at a node relative to the dimension of the problem. In our work, we stopped sampling when the number of instances was less than twice the number of features at a node.

There are several different ways of maintaining the sorted order of the samples. The simplistic, but expensive, approach is to first select the sample, and then do an additional sort to obtain the sorted order. A different approach is stratified sampling, where we divide the instances into $n * p$ parts, each part containing $1/p$ instances, and then randomly select an instance from each part. The sampled set thus remains ordered. More sophisticated approaches are also possible, such as the one described in [16].

Regardless of how the samples are obtained at each node, the process of sampling introduces randomization which can be used to generate an ensemble of trees. Note that the sampling on the instances at a node does not introduce any special considerations for cases such as missing attributes or nominal attributes. These can be handled as in the case of a single decision tree.

4 Experimental Results

In this section, we describe the results of our experiments conducted on some of the larger datasets available from the repository at the University of California at Irvine [15]. The details of the five data sets used are summarized in Table 1. In some cases, where the datasets included a test set, we performed our experiments 10 times to account for the randomization and averaged the results. When no test set was available, we used 10-fold cross-validation, and averaged the results over 10 runs. Based on the observation that most of the improvement in bagging is evident within ten replications [2], 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. In this set of experiments, we used unpruned trees as we expected the use of ensembles to eliminate the overfitting [3]. We used the Gini index as the split criterion [6]. Table 2 gives the estimate of the error averaged over multiple runs for the case of a single tree (without pruning). We also include results with C4.5 (with pruning and default options), Boosting and Bagging; these were taken from [11]. This data allows us to compare our approach with other techniques for creating ensembles.

Table 2: Test error percentages on the benchmark datasets for competitive techniques.

Data set	Single Tree w/o pruning	C4.5 w/ pruning	Boosting w/ C4.5	Bagging w/ C4.5
Breast Cancer	5.17	5.0	3.3	3.2
Pima Indian Diabetes	27.88	28.4	25.7	24.4
German Credit	29.95	29.4	25.0	24.6
Satellite Image	15.75	14.8	8.9	10.6
Letter Recognition	30.62	13.8	3.3	6.8

Table 3: Test error (standard error) on three datasets illustrating the effects of sampling in the creation of ensembles

Percentage Sampled	Breast Cancer	Pima Indian Diabetes	German Credit	Satellite Image	Letter Recognition
0.9	3.41 (.08)	25.17 (.28)	28.47 (.18)	11.76 (.11)	11.71 (.31)
0.8	3.43 (.09)	24.88 (.24)	28.55 (.20)	11.61 (.12)	12.05 (.33)
0.7	3.13 (.11)	24.59 (.17)	28.44 (.15)	11.66 (.07)	11.41 (.18)
0.6	3.38 (.09)	24.75 (.31)	28.39 (.16)	11.68 (.12)	12.13 (.21)
0.5	3.87 (.17)	24.49 (.19)	28.68 (.15)	11.67 (.12)	12.56 (.23)
0.4	3.51 (.11)	24.87 (.18)	28.43 (.29)	11.39 (.10)	11.27 (.31)
0.3	3.25 (.13)	24.47 (.17)	27.87 (.23)	11.61 (.19)	11.64 (.36)
0.2	3.48 (.07)	25.13 (.22)	27.51 (.20)	11.29 (.14)	11.46 (.26)
0.1	3.38 (.08)	24.97 (.22)	27.17 (.35)	11.09 (.14)	11.36 (.25)
0.05	3.40 (.09)	25.21 (.16)	26.62 (.26)	11.65 (.14)	11.87 (.31)
0.01	5.17 (.15)	27.88 (.36)	29.95 (.29)	15.75 (0)	13.62 (.32)

4.1 Accuracy for Sampled Ensembles

Table 3 summarizes our results as we vary the percentage of samples at each node. We observe that the accuracy is roughly constant as the percentage sampled is varied. Also, the error is smaller than for the single unpruned tree. To explain this, we considered how we split the instances at each node. For each feature, we need to identify a value such that a split on that feature at that value will optimize the split criterion. By sampling the instances at a node with a uniform distribution, we are essentially selecting a split that is likely to be close to the optimal split. 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 happens when the sample size at the root node of the tree is less than twice the number of features. At this point, no sampling is performed, and each tree in the ensemble becomes identical to the tree created on the entire training set. When the sample size at the root node is a little bit larger than twice the number of features, 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

Table 4: Timing results (in seconds) comparing 10 runs of the training of a single tree without sampling vs. an ensemble of 10 trees with sampling at 10%.

	Satellite Image	Letter Recognition
Single Tree	63 s	263 s
Ensemble (1)	488 s	1279 s
Ensemble (2)	434 s	1222 s
Ensemble (3)	51 s	123 s

for a single tree. This observation could be used as a rough rule of thumb to bound the smallest fraction of the samples to use in creating the ensembles to be np ¿ 2 * number of features.

In terms of the accuracy obtained by our ensembles relative to other similar approaches, our results are competitive with those obtained through Boosting and Bagging as listed in Table 2. However, for the larger datasets (Satellite Image and Letter Recognition), our algorithm is worse than Boosting or Bagging. This may be due to the fact that we are working with unpruned trees, which give higher errors than pruned trees for these datasets even in the case of a single tree (for example, compare columns 2 and 3 in Table 2). As observed by other authors, e.g. [9], the decision to prune or not prune the tree should be made independently for each dataset.

4.2 Computational Costs

The total computational cost for trees created using our approach is determined by several competing factors. While fewer instances have to be considered at each node, there is an additional sort required, unless we use a smarter approach for the sequential random sampling such as the one in [16]. We can also exploit the fact that since the same training set is used for all the trees in the ensemble, we can reduce the time further by doing the initial sort only once for all the trees. This approach to reducing the cost of ensembles can also be applied through clever coding in the case of Bagging or Boosting.

Our initial timing results are presented in Table 4. 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 time for training alone, as our timing results indicate that training is more time consuming.

We consider three different implementations. In the first implementation, we use a slightly more sophisticated way of sampling at each node that preserves the sorted order of the instances. The instances at a node are divided into 10 equal parts (corresponding to a sampling fraction of 0.1) and then an instance is randomly selected from each part. This avoids the additional sort at each node, but the initial sorting is repeated for each tree in the ensemble. In the second implementation, we do the initial sort across the trees in the ensemble only once, but at each node of the tree, the sample is obtained randomly and then sorted. In the third implementation, we combine the two time-saving aspects of the first two implementations, and create the ensembles by doing the initial sort only once, and using the more sophisticated stratified sampling at each node.

Using the first implementation, we note that building the ensemble of 10 trees is slower than building the single tree by a factor of 4.88 and 8.0 for the letter recognition and satellite datasets, respectively. In the second case, these factors are 4.65 and 6.7, respectively. For the third implementation, the ensembles are slower by a factor of 0.47 and 0.81, respectively. In other words, if we combine the initial "sort once across all trees" with the no additional sorting at each node, we can create ensembles of trees in less time than it would take to create a single tree.

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 sampling the features using the approach in [16] and considering other ways of stratified sampling. 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.

6 Acknowledgements

UCRL-JC-142268-REV-1 - 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

[1] Bauer, E., and Kohavi, R. An empirical comparison of voting classification algorithms: Bagging boosting and variants. *Machine Learning 36*, 1/2 (1999), 105–139.

[2] Breiman, L. Bagging predictors. *Machine Learning 26*, 2 (1996), 123–140.

[3] Breiman, L. Pasting bites together for prediction in large data sets and - on-line. Tech. rep., Statistics Department, University of California, Berkeley, 1996. ftp.stat.berkeley.edu/pub/users/breiman/pastebite.ps.Z.

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

[5] Breiman, L. Random forests - random features. Tech. Rep. Technical Report 567, Statistics Department, University of California, Berkeley, 1999.

[6] Breiman, L., Friedman, J., Olshen, R., and Stone, C. *Classification and Regression Trees.* Chapman and Hall/CRC Press, Boca Raton, Florida, 1984.

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

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

[9] DIETTERICH, T. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning 40*, 2 (2000), 139–158.

[10] DIETTERICH, T., AND BAKIRI, G. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research 2* (1995), 263–286.

[11] FREUND, Y., AND SCHAPIRE, R. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference* (1996), pp. 148–156.

[12] HALL, L., BOWYER, K., KEGELMEYER, W., MOORE, T., AND CHAO, C. Distributed learning on very large data sets. In *Workshop on Distributed and Parallel Knowledge Discover, in conjunction with KDD2000* (2000).

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

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

[15] UCI Knowledge Discovery in Databases Archive, 2001. http://kdd.ics.uci.edu/.

[16] VITTER, J. An efficient algorithm for sequential random sampling. *ACM Transactions on Mathematical Software 13*, 1 (1987), 58–67.

University of California
Lawrence Livermore National Laboratory
Technical Information Department
Livermore, CA 94551