

Approximate Splitting for Ensembles of Trees using Histograms

*Chandrika Kamath**, *Erick Cantú-Paz†*, *David Littau‡*

1 Introduction

Ensembles of classifiers have become an active topic of research in the data mining community. They not only provide a simple way of improving the accuracy of the classifier [3, 16, 24, 2], but also have the potential for on-line classification of large databases that do not fit into memory [4]. In addition, some approaches to the generation of ensembles can be easily parallelized, enabling a reduction in the time taken to create the classifier on a multiprocessor system [17]. There are several 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 histograms to determine the split made at each node of a decision tree. The idea of using histograms to approximate the split at a node has been around a long time as a way of reducing the time to create a tree using a very large training set [8]. For each feature, instead of sorting the instances at a node, a histogram is created, and the bin boundaries used as potential split points. The expectation is that the best bin boundary will be close to the best value for a split point chosen using the traditional sorting approach. As a result, the tree created using histograms would

*Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA 94551. kamath2@llnl.gov

†Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA 94551. cantupaz1@llnl.gov

‡Department of Computer Science, University of Minnesota, 200 Union St. S.E., Minneapolis, MN 55455. littau@cs.umn.edu

be close in accuracy to the tree created using sorting. However, since no sorting is done, and fewer potential split points are evaluated, it is likely to take less time to create the tree using histograms than the tree using sorting of the continuous features.

To introduce randomization into this process, once we have found the best bin boundary, we select a split point randomly in an interval around it. As a result of this randomization, for a given training set, different trees are created each time, and their results can therefore be combined through ensembles. Our experimental studies with public domain datasets show that this simple approach reduces the time to create ensembles of trees while leading to improved accuracy on large datasets.

The paper is organized as follows: In Section 2, we briefly discuss the various ways in which we can generate ensembles of classifiers. Next, in Section 3 we describe the ways in which continuous attributes can be discretized to make them more suitable for use in classification. We show how one such discretization technique, namely histograms, can be used 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 are several ways in which ensembles of classifiers can be created [10]. Some of the more popular approaches include:

- **Changing the Instances Used for Training:** In this approach, each classifier in the ensemble is generated using a different sample of the training set. In the *Bagging* algorithm, a new sample of the training set is obtained through bootstrapping with each instance weighted equally [3], and the results of the ensemble obtained by using a simple voting scheme. In the *Boosting* algorithm, a new sample of the training set is obtained using a distribution based on previous results [16]. After each classifier is created, the weights are adjusted to increase the weights of misclassified instances. The results of the ensemble are obtained by weighting each classifier by the accuracy on the training set used to build it. Thus, better classifiers have a greater contribution to the end result. There are several variants of boosting which differ in the way the instances are weighted, the results are combined, and the algorithm is terminated [5, 2, 16]. In the *Pasting* algorithm, the ensemble of trees is grown using a sub-sample of the entire training set [4]. This technique is useful when the entire training set is too large to fit into main memory.
- **Changing the features used in training:** In this approach, each new classifier is created using a subset of the original features. This technique has been used with decision trees in [19] and with neural networks in [9]. It tends to work only when the features are redundant, as poor classifiers could result if some important feature is left out.
- **Changing the output targets:** In this approach, called error correcting output coding [12], a problem with many output classes is first reduced to

several two-class problems by randomly partitioning the classes into two, with labels 0 and 1. A classifier is created with this relabeled data. For each random partitioning of the original set of classes, a new classifier is created. 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.

- **Introducing randomness in the classifier:** Instead of changing the input or output to the classifier to generate the ensemble, we can change the classifier itself. For example, in neural networks, the initial weights are set randomly, creating a new network each time. In decision trees, we can randomly select among the best few splits to create the ensemble [11]. Or, we can randomly select the features used to determine the split at each node of the tree [6]. In earlier work [20], we showed that we can randomize the split at a node of a tree by using a sample of the instances at each node to obtain the potential split points for each feature. We next describe a different way of introducing randomization in the decision tree classifier through the use of histograms.

3 Randomizing the Split at a Node using Histograms

Histograms are a way of “discretizing” or “quantizing” continuous attributes in classification algorithms. In addition to converting the input data into a form suitable for classification, discretization techniques may also reduce the CPU time required for training with very large datasets [8, 13, 1, 21]. They have also been used to identify inconsistencies in the data, and select the relevant features while discretizing the data [22].

There are several categories of discretization that can be used in classification [8, 13, 28, 18]. In this paper, we consider the use of simple equal-width histograms as the discretization approach for continuous attributes in decision tree induction. However, instead of just using them to create a single decision tree, we use them to introduce randomization in the split at each node of the tree, and create an ensemble of trees.

We explain this idea by first describing how histogram-based discretization works in the context of a single tree. In the traditional approach to tree induction, for each feature, the instances at each node of the tree are sorted on that feature, and potential split points considered mid-way between all these feature values. The splitting criterion (e.g., Gini or Information Gain [23]) is evaluated at each potential split point. The split point and split feature that are selected are the ones that optimize the splitting criterion [7]. In contrast, in the histogram approach, a histogram is built for each feature and the candidate split points chosen at the bin boundaries. This not only avoids the time consuming sort, but also results in fewer split points being evaluated. However, on the negative side, it may lower the accuracy of the resulting tree as the split is only an approximate split. In addition, while the sorting is done only once at the beginning of the root node, the histograms have to be created at each node of the tree.

Note that when we build the histogram, we need to keep track of the class of the instance that is assigned to each bin. This is because the class is required to evaluate the splitting criterion at the bin boundaries. Also, we do not have to restrict the optimum split point to be at a bin boundary. If the best boundary point is at either of the extreme points of the histogram, we choose the median of the corresponding bin as the split point. Further, to account for the fact that the bins on either side of the boundary may have vastly different number of instances, we select the split point as

$$\text{splitpoint} = \frac{\text{median}_i * \text{num}_i + \text{median}_{i-1} * \text{num}_{i-1}}{\text{num}_i + \text{num}_{i-1}} \quad (1)$$

where the best boundary point is between bins i and $i - 1$, num_i is the number of instances in bin i , and median_i is the median of bin i . This ensures that the split point is biased towards the bin with the larger number of instances. Note that if the two bins on either side of the best boundary point have equal number of instances, the boundary point is selected as the split point.

Once we have obtained an approximate split using histograms, we introduce randomization into the split as follows. Selecting the best bin boundary, we choose a random split point in an interval around this bin boundary. The bin boundary is chosen as the center of the interval and the width of the interval is the width of a bin. In other words, we generate a uniformly distributed random number in the interval $[\text{median}_{i-1}, \text{median}_i]$. As before, we could have weighted the interval by the number of instances in the bins on either side of the best boundary point in order to ensure that a random point selected in this interval has a higher likelihood of being in the bin with the larger number of instances. However, for the experiments reported in this work, we did not apply this weighting.

For the histogram-based ensembles, we combined the results of the ensemble by a simple unweighted voting scheme. We could have also used weighted voting to combine the results, or even bagging or boosting to select the instances used in creating the trees in the ensemble.

Our goal in this paper is to show that this approach to generating ensembles through approximate splits using histograms is a simple, but efficient way of improving the accuracy of decision tree classifiers. On one hand, the use of histograms reduces the CPU time, while potentially sacrificing accuracy. On the other, the use of ensembles improves the accuracy of the resulting classifier.

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 [29]. The details of the five data sets used are summarized in Table 1. In 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.

We present results with both pruned and unpruned trees as the benefits of pruning are very dependent on the dataset [11]. In the case of pruned trees, we use

Table 1. *Benchmark data sets used for studying the effect of histograms 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

pessimistic error pruning described in [26]. We also tried a modified version of the error-based pruning used in the C4.5 software [25], where each pruned subtree was always replaced by a leaf. However, we found that this method (using confidence levels of 10% and 5%) rarely gave better results than the pessimistic error pruning. In all experiments, we used the Gini splitting criterion [7] for all datasets, except for letter recognition, for which we used information gain. This was because it has a large number of classes, and Gini typically does not perform well in such cases, an observation that was validated by our experimental results. For example, a single tree, without pruning, gave test error of 39.57 with the Gini splitting criterion, but 27.35 with information gain. When pessimistic pruning was used, the corresponding errors were 39.05 and 26.90 respectively.

There are several different ways in which we can determine the number of bins in the histogram [8, 15, 21, 13, 14]. While this is an important issue, since our focus is on ensembles, we chose a simple approach. We first used equal-width histograms, with the number of bins equal to the square-root of the number of instances at a node. In this case, we created histograms regardless of the number of instances at a node of the tree. Then, suspecting that histograms were probably not a good idea when the number of instances was small (in our case, equal to the number of features), we stopped using the histograms and used the traditional sorting of all instances. This was done simply by setting the number of bins equal to the number of instances at the node instead of the square-root of the number of instances.

We also include comparisons with the more traditional ensemble techniques such as AdaBoost [16], Bagging [3], and ArcX4 [5]. All experiments were conducted using the Sapphire software [27] developed at Lawrence Livermore National Laboratory.

Our first set of experiments (Tables 2 and 3) focused on a single tree to understand how the accuracy changed when we replaced the exact split at a node (found by sorting the instances in an axis-parallel implementation) by a split determined by using the best bin boundary in the histogram. In this set of results, we did not use ensembles. For the histogram-based tree, two sets of results are given; in the first case, the histograms are used regardless of the number of instances at a node, and in the second case, when the number of instances was less than the number of features, we shifted to the traditional sorting-based technique. For the three smaller

Table 2. Test error (standard error) percentages on the benchmark datasets for a single tree. Histogram-tree (A) uses histograms regardless of the number of instances, histogram-tree (B) uses histograms when the number of instances is larger than the number of features at a node, and then shifts to simple sorting instead of histograms. The lowest error rate in each column is in bold.

Method	Cancer	Diabetes	Credit
Axis-parallel w/o pruning	5.97 (0.29)	29.27 (0.30)	32.10 (0.44)
Axis-parallel w/ pruning	5.69 (0.16)	26.64 (0.49)	27.79 (0.23)
Histogram-tree (a) w/o pruning	5.22 (0.17)	28.41 (0.24)	29.56 (0.16)
Histogram-tree (a) w/ pruning	5.00 (0.18)	24.53 (0.37)	27.37 (0.27)
Histogram-tree (b) w/o pruning	5.16 (0.17)	28.38 (0.24)	29.40 (0.17)
Histogram-tree (b) w/ pruning	5.01 (0.18)	24.71 (0.37)	27.33 (0.26)

Table 3. Test error percentages on the benchmark datasets for a single tree. Histogram-tree (a) uses histograms regardless of the number of instances, histogram-tree (b) uses histograms when the number of instances is larger than the number of features at a node, and then shifts to simple sorting instead of histograms. The lowest error rate in each column is in bold.

Method	Satellite	Letter Recognition
Axis-parallel w/o pruning	15.85	27.35
Axis-parallel w/ pruning	14.80	26.90
Histogram-tree (a) w/o pruning	15.70	16.37
Histogram-tree (a) w/ pruning	15.20	16.27
Histogram-tree (b) w/o pruning	15.75	16.42
Histogram-tree (b) w/ pruning	15.20	16.375

datasets, we report both the test error and the standard error as a result of cross validation. For the larger datasets, since we have separate training and testing data, and no randomization is used in this set of experiments, we report only the test error.

Based on the results presented in Tables 2 and 3, we observe that pruning is helpful in some data sets (e.g., Pima Indian Diabetes and German Credit), but doesn't significantly affect the accuracy in other data sets (e.g., breast cancer and letter recognition). This supports the observation made in [11] that pruning makes a difference in some cases, but not in others. We also observe that using the histogram-based tree, where the splits are made on bin boundaries, does not result in a degradation in the accuracy, but may even improve the accuracy substantially in some cases, such as the letter recognition data set. This observation has also been made by other authors [8, 13]. In addition, we observe that the two variants of the histogram-based tree do not differ significantly in accuracy.

Table 4. Test error (standard error) percentages on the benchmark datasets for histogram-based ensembles. Histogram-tree (A) uses histograms regardless of the number of instances, histogram-tree (B) uses histograms when the number of instances is larger than the number of features at a node, and then shifts to simple sorting instead of histograms. The lowest error rate in each column is in bold.

Method	# trees	Cancer	Diabetes	Credit
Histogram-tree (a) w/o pruning	10	4.73 (0.17)	25.50 (0.31)	28.95 (0.18)
	20	4.53 (0.13)	24.96 (0.18)	29.19 (0.31)
	50	4.76 (0.17)	24.76 (0.33)	29.04 (0.22)
	100	4.56 (0.15)	24.04 (0.18)	29.62 (0.18)
Histogram-tree (a) w/ pruning	10	5.09 (0.10)	23.74 (0.25)	26.96 (0.14)
	20	5.09 (0.13)	23.95 (0.24)	27.01 (0.19)
	50	4.88 (0.11)	23.60 (0.12)	26.81 (0.19)
	100	5.03 (0.13)	23.26 (0.13)	27.09 (0.19)
Histogram-tree (b) w/o pruning	10	4.62 (0.16)	25.57 (0.28)	25.66 (0.44)
	20	4.51 (0.13)	25.05 (0.16)	29.10 (0.33)
	50	4.78 (0.18)	24.74 (0.30)	28.96 (0.25)
	100	4.56 (0.15)	24.18 (0.14)	29.57 (0.18)
Histogram-tree (b) w/ pruning	10	5.01 (0.18)	23.66 (0.27)	27.33 (0.26)
	20	5.06 (0.13)	23.66 (0.22)	27.00 (0.19)
	50	4.79 (0.12)	23.41 (0.19)	26.82 (0.17)
	100	4.98 (0.12)	23.26 (0.15)	27.10 (0.19)

4.1 Accuracy for Histogram-based Ensembles

We next present the results of ensembles of trees, where each tree is created using a split randomly chosen in an interval around the best bin boundary. The results are presented in Tables 4 and 5 as the number of trees in the ensemble is varied. As before, we report the test error and standard error for 10 runs of 10-fold cross-validation for the smaller data sets. For the larger data sets with the training and test data, since the algorithm is randomized, we report the test error and standard error of 10 runs. Unlike the case of a single tree reported in Table 3, the randomization in the ensemble gives a different result for each run. In addition, for the larger datasets, we did not run the experiments with 100 trees in the ensemble as it took too much time.

Based on these results, we observe that the use of the ensembles does improve the accuracy of the histogram-based classifier, though the improvement is dependent on the dataset. For example, comparing the best error rate in the columns of Table 2 and Table 4, we find that the error rate for the Pima Indian Diabetes dataset reduces from 24.53 to 23.26, while for the German Credit data set, it reduces from 27.33 to 25.66. A similar reduction can be observed in the larger data sets by comparing Tables 3 and 5. The accuracy often improves as the number of trees in the ensemble is increased, though in some cases, this improvement is small and perhaps not worth the extra computation. We also observe that in the case of

Table 5. Test error (standard error) percentages on the benchmark datasets for an ensemble of histogram-based trees. Histogram-tree (A) uses histograms regardless of the number of instances, histogram-tree (B) uses histograms when the number of instances is larger than the number of features at a node, and then shifts to simple sorting instead of histograms. The lowest error rate in each column is in bold.

Method	# trees	Satellite	Letter Recognition
Histogram-tree (a) w/o pruning	10	13.70 (0.13)	12.03 (0.24)
	20	13.41 (0.12)	11.96 (0.16)
	50	13.20 (0.08)	11.69 (0.11)
Histogram-tree (a) w/ pruning	10	14.55 (0.13)	13.36 (0.17)
	20	14.68 (0.07)	13.15 (0.17)
	50	14.33 (0.08)	12.88 (0.11)
Histogram-tree (b) w/o pruning	10	13.72 (0.12)	11.94 (0.24)
	20	13.27 (0.07)	11.79 (0.16)
	50	13.17 (0.08)	11.64 (0.14)
Histogram-tree (b) w/ pruning	10	14.56 (0.11)	13.22 (0.24)
	20	14.65 (0.06)	12.97 (0.14)
	50	14.32 (0.08)	12.89 (0.11)

histogram-based ensembles, pruning can some times help improve the accuracy, while in other data sets, it can lower the accuracy. In addition, the two different options for using the histograms do not differ in accuracy.

To further experiment with the idea of creating an approximate split at a node of the tree, we tried combining the idea of sampling the instances [20] with the use of histograms. So, for each feature, we first randomly sample the instances, and use these sampled instances to create the histogram. Randomization is now introduced in two ways - first, due to the sampling and second, due to the random split in the histogram. For the two larger datasets, the results of this experiment are reported in Table 6. 10% of the instances at a node are sampled, unless the number of instances is less than twice the number of features at a node, in which case no sampling is done. This is done to ensure that there are enough instances at a node to learn a hypothesis in a space of dimension equal to the number of features. We also restrict the experiments to the case where histograms are used regardless of the number of instances at a node, as our earlier results indicated that this option had no influence on the accuracy.

Our experiments in Table 6 indicate that combining sampling with histograms can result in additional reduction in the errors for large datasets.

4.2 Results for Other Ensemble Techniques

For comparison, we also include the results with other competitive techniques in Table 7. These are result of 10 runs. For the smaller datasets, we use 10-fold cross-validation; for the larger datasets (letter and satellite), it is just training and

Table 6. Test error (standard error) percentages on the benchmark datasets for an ensemble of trees combining sampling and histograms. The sampling is done at 10%. Results are for histogram-based trees where histograms are used regardless of the number of instances. The lowest error rate in each column is in bold.

Method	# trees	Satellite	Letter Recognition
Histogram-tree (a) w/o pruning	10	11.18 (0.12)	7.85 (0.11)
	20	13.41 (0.12)	6.81 (0.06)
	50	10.21 (0.08)	6.31 (0.04)
Histogram-tree (a) w/ pruning	10	12.16 (0.17)	9.25 (0.13)
	20	11.75 (0.07)	8.20 (0.08)
	50	11.55 (0.08)	7.72 (0.06)

Table 7. Test error (Standard error) percentages on the benchmark datasets for competitive techniques. All results are with pruning for 10 runs. The lowest error rate in each column is in bold.

Method	# trees	Cancer	Diabetes	Credit	Satellite	Letter
AdaBoost	10	4.69 (0.18)	25.26 (0.47)	26.20 (0.31)	12.65 (-)	22.55 (-)
	20	4.69 (0.18)	24.50 (0.37)	26.03 (0.50)	12.65 (-)	22.55 (-)
	50	4.69 (0.18)	24.50 (0.37)	25.81 (0.53)	12.65 (-)	22.55 (-)
Bagging	10	3.59 (0.11)	24.50 (0.32)	26.60 (0.19)	12.91 (0.13)	14.71 (0.36)
	20	3.66 (0.09)	23.30 (0.25)	26.00 (0.31)	12.34 (0.12)	12.45 (0.19)
	50	3.37 (0.08)	23.79 (0.19)	26.21 (0.18)	11.99 (0.10)	11.18 (0.08)
ArcX4	10	3.96 (0.08)	25.77 (0.44)	27.92 (0.23)	11.75 (-)	17.60 (-)
	20	3.87 (0.11)	26.05 (0.40)	27.86 (0.22)	11.05 (-)	15.50 (-)
	50	3.87 (0.20)	26.32 (0.36)	28.34 (0.26)	10.70 (-)	11.65 (-)

testing, as a separate test set is available. Also, for the larger data sets, for the AdaBoost and ArcX4 methods, we report only the test error. Since there is no randomization in these techniques, we get the same accuracy for each of the 10 runs. However, since there is randomization in bagging, we get different results for each run and therefore report both the test and standard error. In the case of the satellite and letter datasets, AdaBoost stops after 8 and 7 trees, respectively [16]. As a result, there is no improvement in the accuracy by increasing the number of

Table 8. *Timing results (in seconds) comparing 10 runs for different ensemble-based techniques*

Data set	# trees	AdaBoost	Bagging	ArcX4	Histogram-based
Satellite	10	405 s	568 s	500 s	304 s
	20	405 s	1134 s	1025 s	609 s
	50	400 s	2892 s	2715 s	1522 s
Letter	10	4245 s	4785 s	6475 s	714 s
	20	4250 s	9590 s	13190 s	1442 s
	50	4255 s	25528 s	35730 s	5346 s

trees.

Comparing the data in Table 7 with the histogram-based ensembles in Tables 4 and 5, we observe that our techniques are competitive in accuracy to other techniques for creating ensembles. In some cases (Pima Indian Diabetes, German Credit, and letter), the accuracy of the histogram-based ensembles is the same as the best of the other ensembles. For the other two data sets, the histogram-based ensembles are not as accurate as the best of the other methods, but are still competitive. When we combine our histogram-based ensembles with sampling for the larger datasets, we tend to improve the performance beyond the traditional ways of creating ensembles such as boosting and bagging. These accuracy results must also be viewed in light of the computational cost of the various ensemble-based techniques, an issue we discuss next.

4.3 Computational Costs

Our next set of experiments explores the reduction in compute time through the use of histograms. The total computational cost for trees created using our approach is determined by several competing factors. In the traditional axis-parallel tree, a single sort of the instances is done at the beginning for each feature. While this sort is replaced by a histogram, reducing the compute time, we now need to create the histogram at each node of the tree, which increases the time. Also, the splitting criterion needs to be evaluated only at the bin boundaries of the histograms instead of between all the values for a feature.

Our initial timing results are presented in Table 8. We focus on the two larger datasets, satellite and letter recognition. The times given are in seconds on a 1.5GHz Pentium III system with 512MB of memory. The comparison is among the average of 10 runs of various ensemble-based approaches, where each classifier is created with pruning. The histogram-based ensembles use histograms regardless of the number of instances at a node. For comparison, the time for a single axis-parallel tree generated using the traditional approach with sorting, is 71 seconds for the satellite dataset and 585 seconds for the letter recognition dataset.

Note that the timing increases proportionately for all ensembles as the number of trees in the ensemble is increased. The exception is the AdaBoost algorithm,

which in both datasets, converges early, after creating 8 trees for the satellite dataset and 7 for the letter dataset. As a result, when the number of trees in the ensemble is varied beyond this number, it has no effect on the accuracy (Table 7) or the time (Table 8). Given the current implementation of the algorithms, the histogram-based techniques are among the fastest on the larger datasets in our study. The exception is the AdaBoost technique, which as we have observed, converges early. If we combine our histogram-based technique with sampling (as reported in Table 6) the compute time reduces slightly.

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 histograms. Once we have found the best bin boundary, we select the split point randomly in an interval around this bin boundary. Our experiments with public-domain data show that this simple approach is competitive in accuracy, but can be superior in time to other ensembles-based approaches such as boosting and bagging. We can improve the accuracy still further by combining sampling with the histograms. In our work, we used simple equal-width histograms and a number of bins equal to the square-root of the number of instances, though more sophisticated techniques are certainly possible. In our future work, we plan to explore the use of such techniques as well as other ways of combining the results of the ensembles. We will also look into improving the compute time for all algorithms through code optimizations such as generating the histogram at the root node only once for all the trees in the ensembles.

6 Acknowledgements

The work of David Littau was performed while he was a summer student at the Center for Applied Scientific Computing at the Lawrence Livermore National Laboratory.

UCRL-JC-145576: 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.

Bibliography

- [1] ALSABTI, K., RANKA, S., AND SINGH, V. CLOUDS: A decision tree classifier for large datasets. In *The Fourth International Conference on Knowledge Discovery and Data Mining* (1998), pp. 2–8.
- [2] BAUER, E., AND KOHAVI, R. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning* 36, 1/2 (1999), 105–139.
- [3] BREIMAN, L. Bagging predictors. *Machine Learning* 26, 2 (1996), 123–140.
- [4] 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>.
- [5] BREIMAN, L. Arcing classifiers. *Annals of Statistics* 26 (1998), 801–824.
- [6] BREIMAN, L. Random forests - random features. Tech. Rep. Technical Report 567, Statistics Department, University of California, Berkeley, 1999.
- [7] BREIMAN, L., FRIEDMAN, J., OLSHEN, R., AND STONE, C. *Classification and Regression Trees*. Chapman and Hall/CRC Press, Boca Raton, Florida, 1984.
- [8] CATLETT, J. *Megainduction: machine learning on very large databases*. PhD thesis, Basser Department of Computer Science, University of Sydney, 1991.
- [9] 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/>.
- [10] DIETTERICH, T. Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems* (2000), Springer Verlag, pp. 1–15.
- [11] 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.

- [12] DIETTERICH, T., AND BAKIRI, G. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research* 2 (1995), 263–286.
- [13] DOUGHERTY, J., KOHAVI, R., AND SAHAMI, M. Supervised and unsupervised discretization of continuous features. In *Machine Learning: Proceedings of the Twelfth International Conference* (1995).
- [14] ELOMAA, T., AND ROUSU, J. General and efficient multisplitting of numerical attributes. *Machine Learning* (1999), 1–49.
- [15] FAYYAD, U., AND IRANI, K. On the handling of continuous values attributes in decision tree generation. *Machine Learning* 9 (1992), 87–102.
- [16] FREUND, Y., AND SCHAPIRE, R. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference* (1996), pp. 148–156.
- [17] 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 Discovery, in conjunction with KDD2000* (2000).
- [18] HO, K., AND SCOTT, P. A global method for discretization of continuous variable. In *Proceedings of KDD-97, The Third International Conference on Knowledge Discovery and Data Mining* (1997).
- [19] HO, T. K. Random decision forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition* (1995), pp. 278–282.
- [20] KAMATH, C., AND CANTÚ-PAZ, E. Creating ensembles of decision trees through sampling. In *Proceedings of the 33-rd Symposium on the Interface: Computing Science and Statistics* (2001).
- [21] KERBER, R. Chimerge: Discretisation of numeric attributes. In *Proceedings of the Tenth National Conference on Artificial Intelligence* (1992), AAAI Press and MIT Press, pp. 123–128.
- [22] LIU, H., AND SETIONO, R. Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of the 7th International Conference on Tools with Artificial Intelligence* (1995), pp. 388–391.
- [23] MURTHY, K. V. S. *On Growing Better Decision Trees from Data*. PhD thesis, Johns Hopkins University, 1997.
- [24] 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.
- [25] QUINLAN, J. R. *C4.5 Programs for machine learning*. Morgan-Kaufmann, 1993.

- [26] QUINLAN, R. Simplifying decision trees. *Intl. J. Man-Machine Studies* 27 (1987), 221–234.
- [27] Sapphire: Large-Scale Data Mining and Pattern Recognition. <http://www.llnl.gov/casc/sapphire>.
- [28] SUSMAGA, R. Analyzing discretizations of continuous attributes given a monotonic discrimination function. *Intelligent Data Analysis 1* (1997), 157–179.
- [29] UCI Knowledge Discovery in Databases Archive, 2001. <http://kdd.ics.uci.edu/>.