Overfitting Explained: A Case Study

David Jensen, Tim Oates, and Paul R. Cohen
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{jensen|oates|cohen}@cs.umass.edu

1 Introduction

Overfitting is a widely observed pathology of induction algorithms. Overfitted models contain unnecessary structure that reflects nothing more than random variation in the data sample used to construct the model. Such models are less efficient to store and use than their correctly-sized counterparts. Using these models requires the collection of unnecessary data. Portions of overfitted models are wrong and mislead users. Finally, overfitting can reduce the accuracy of induced models on new data [8].

For induction algorithms that build decision trees [1, 7, 10], *pruning* is a common approach to correct overfitting. Pruning methods take an induced tree, examine individual subtrees, and remove those subtrees deemed to be unnecessary. Pruning methods primarily differ in the criterion used to judge subtrees. Many criteria have been proposed, including statistical significance tests [10], corrected error estimates [7], and minimum description length calculations [9].

In this paper, we bring together three threads of our research on overfitting and decision tree pruning. First, we show that several common methods for pruning decision trees still allow overfitting to occur. Second, we explain overfitting in terms of statistical decision making with incorrect reference distributions. Third, we present a method that adjusts for using incorrect reference distributions, and we present an experiment that evaluates that method. These topics hold important implications for intelligent data analysis. Our analysis of overfitting indicates that many existing techniques for building decision trees fail to consider the statistical implications of examining many possible subtrees. We show how a simple statistical adjustment can allow such systems to reason correctly in this specific situation.

2 Observing Overfitting

Consider Figure 1, which shows a typical plot of tree size and accuracy as a function of training set size for the UCI `australian` dataset.[1] Moving from left-to-right in the graph corresponds to increasing the number of training instances available to the tree building process. On the left-hand side, no training instances are available and the best one can do with test instances is to assign them a class label at random. On the right-hand side, the entire dataset (excluding test instances) is available to the tree building process. C4.5 [7] and error-based pruning (the c4.5 default) are used to build and prune trees, respectively.

Note that accuracy on this dataset stops increasing at a rather small training set size, thereafter remaining essentially constant. Surprisingly, tree size continues to grow nearly linearly despite the use of error-based pruning to avoid overfitting. The graph clearly shows that overfitting is occurring, and it gets worse as the size of the training set increases. Accuracy stops increasing after only 25% of the available training instances are seen. The tree at that point contains 22 nodes. When 100%

[1] All datasets in this paper can be obtained from the University of California–Irvine (UCI) Machine Learning Repository. `http://www.ics.uci.edu/ mlearn/MLRepository.html`.

of the available training instances are used in tree construction, the resulting tree contains 64 nodes. Despite a 3-fold increase in size over the tree built with 25% of the data, the accuracies of the two trees are statistically indistinguishable.

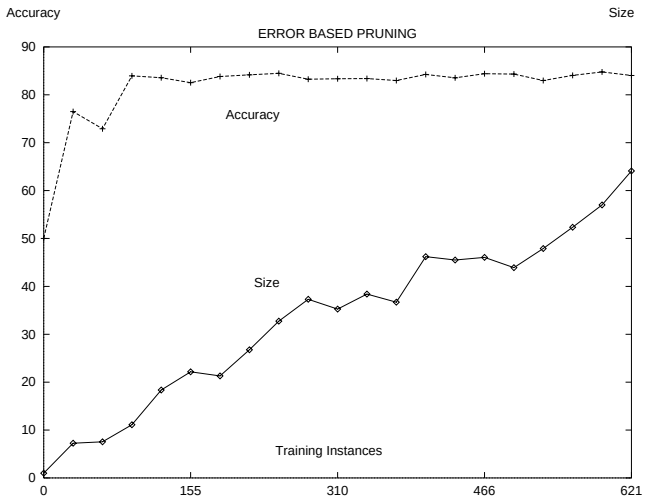

Figure 1: Tree size and accuracy as a function of training set size for the `australian` dataset. C4.5 and error-based pruning were used to build and prune trees.

Under a broad range of circumstances, there is a nearly linear relationship between training set size and tree size, even after accuracy has ceased to increase. The relationship between training set size and tree size was explored with 4 pruning methods and 19 datasets taken from the UCI repository.[2] The pruning methods are error-based (EBP – the C4.5 default) [7], reduced error (REP) [8], minimum description length (MDL) [9], and cost-complexity with the 1SE rule (CCP) [1]. The majority of extant pruning methods take one of four general approaches: deflating accuracy estimates based on the training set (e.g. EBP); pruning based on accuracy estimates from a pruning set (e.g. REP); managing the tradeoff between accuracy and complexity (e.g. MDL); and creating a set of pruned trees based on different values of a pruning parameter and then selecting the appropriate parameter value using a pruning set or cross-validation (e.g. CCP). The pruning methods used in this paper were selected to be representative of these four approaches.

Plots of tree size and accuracy as a function of training set size were generated for each combination of dataset and pruning algorithm as follows. Typically, k-fold cross-validation is used to obtain estimates of the true performance of decision tree algorithms. A dataset, D, with n instances is divided into k disjoint sets, D_i, each containing n/k instances. Then for $1 \leq i \leq k$, a tree is built on the instances in $D - D_i$ and tested on the instances in D_i, and the results are averaged over all k folds [2]. That procedure was augmented for this paper by building trees on subsets of $D - D_i$ of various sizes, and testing them on D_i. Specifically, 20 subsets were created by retaining from 5% to 100% of the instances in $D - D_i$ in increments of 5%; standard k-fold cross-validation corresponds to the case in which 100% of the instances in $D - D_i$ are retained. The order of the instances in D was permuted prior to creating the $k = 10$ folds, and the instances to be retained were gathered sequentially starting with the first instance in $D - D_i$ for each level of data reduction. In this way,

[2]The datasets are the same ones used in [4] with two exceptions. The `crx` dataset was omitted because it is roughly the same as the `australian` dataset, and the `horse-colic` dataset was omitted because it was unclear which attribute was used as the class label. Note that the `vote1` dataset was created by removing the `physician-fee-freeze` attribute from the `vote` dataset.

10-fold cross-validated estimates of tree size and accuracy as a function of training set size were obtained.

This procedure was performed twice for each combination of dataset and pruning method, generating complete size and accuracy curves for two different permutations of the data, and the results were averaged. The goal was to reduce the inherent variability of cross-validated estimates of size and accuracy. Note that the same divisions of a given dataset were used for all of the pruning methods. With 19 datasets, 4 pruning methods, 20 levels of training set size, and 2 runs of 10-fold cross-validation at each level of training set size, the results reported in this section involved running C4.5 30,400 times.

For each plot generated by this procedure, the training set size at which accuracy ceased to grow was found by scanning the accuracy curve from left to right, stopping when the mean of three adjacent accuracy estimates was no more than 1% less than the accuracy of the tree based on all available training data (the right-most point on the accuracy curve). Averaging three adjacent accuracies makes the stopping criterion robust against random variations in the accuracy curve.[3] Bounding the absolute change in accuracy from below by 1% ensures that any reduction in tree size costs very little in terms of accuracy. Then, a linear regression of tree size on training set size was performed on the points in the tree size curve to the right of the training set size at which accuracy ceased to grow.

In general, additional tree structure is welcome as long as it improves classification accuracy, and it is unwelcome otherwise. Ideally, there will be no correlation between tree size and training set size once classification accuracy stops increasing. The linear regression of tree size on training set size indicates the probability, p, of incorrectly rejecting the null hypothesis that there is no such correlation (that the slope of the regression line is zero), the estimated slope of the regression line, and the amount of variance in tree size accounted for by training set size, R^2. When p is significant and R^2 is high, changes in training set size have predictable effects on tree size. When the slope of the regression line is large, the effects are strong.

For each combination of dataset and pruning method, we recorded the percentage of available training instances at which accuracy ceased to grow, results of the linear regression of tree size on training set size (p, slope, and R^2), the percentage decrease in tree size (Δ size) and the absolute difference in accuracy (Δ accuracy) between the tree built from all available training instances and the tree built from the number of instances at which accuracy ceased to grow. A summary of that information is given in Table 1, which shows for each pruning method the number of datasets for which accuracy peaked prior to seeing 100% of the available training instances (% Kept < 100), the number of datasets for which the relationship between tree size and training set size is statistically significant ($p < 0.1$), the number of datasets for which the relationship is both statistically significant and strong ($slope > 0.1$), and the means of R^2, Δ size and Δ accuracy for those datasets with significant p values.

For EBP accuracy peaked prior to seeing 100% of the available training instances for 16 of the 19 datasets. Every one of those 16 datasets exhibited a significant relationship between tree size and training set size beyond the point at which accuracy stopped growing, and 12 of them were highly significant (at the 0.001 level). In 13 of the 16 datasets that exhibit a significant relationship, the slope of the regression line exceeds 0.1, indicating that at least one node is added for every 10% increase in the size of the dataset.

In spite of the fact that accuracy remains basically constant, tree size continues to grow as training set size does (the slope of the regression line is positive in all cases). The most remarkable

[3] We did not use the mean of the final three points on the accuracy curve minus 1% as the accuracy threshold because those points represent different training set sizes, and their mean is therefore not an estimate (robust or otherwise) of the accuracy of trees built on all available training instances.

Pruning Method	% Kept < 100	p < 0.1	Slope > 0.1	Mean R^2	Mean Δ size	Mean Δ accuracy
EBP	16	16	13	0.90	38.29	-0.14
REP	17	17	11	0.75	39.32	-0.32
MDL	18	17	13	0.88	44.03	-0.37
CCP	19	10	4	0.62	30.11	-0.06

Table 1: Summary of the effects of random data reduction for all of the pruning methods.

feature of the EBP row in Table 1 is the R^2 column. Recall that $100 * R^2$ is the percentage of variance in tree size accounted for by training set size. Across 16 datasets, the average R^2 is 0.90. This result is interesting for two reasons. First, it says that training set size has an extremely predictable effect on tree size. Increasing training set size invariably leads to larger trees; decreasing training set size invariably leads to smaller trees. Second, this effect is robust over a large group of datasets with widely varying characteristics. Regardless of the default accuracy, the number and types of attributes, the presence or absence of class and attribute noise, and differences in a number of other features along which the datasets vary, EBP does not appropriately limit tree size as training set size increases.

The results for REP and MDL are qualitatively the same as those for REP. For REP, 17 datasets show a significant relationship between tree size and training set size (12 at the 0.001 level), 11 of the 17 datasets had a slope greater than 0.1, and the mean R^2 is 0.75. The average reduction in tree size obtainable via random data reduction is 39.32% with an average loss in accuracy of less than four tenths of one percent. Accuracy was higher with reduced training sets in 12 of the 17 cases. For MDL, 17 datasets had significant p values (14 at the 0.001 level), 13 of the 17 datasets had a slope greater than 0.1, and the average R^2 was 0.88. Trees based on reduced training sets were on average 44.03% smaller and less than four tenths of one percent less accurate. Note that for one dataset, `hypothyroid`, there is no significant relationship between tree size and training set size past the point at which accuracy stopped growing. In this one case, MDL appropriately limits tree size by not adding structure to the tree unless a concomitant increase in classification accuracy occurs.

The results for CCP indicate that it appropriately limits tree growth much more frequently than the previous three pruning methods. Accuracy peaked for all 19 datasets prior to seeing 100% of the available training instances. However, only about half of the time (10 out of 19 datasets) was there a significant relationship between tree size and training set size after accuracy stopped growing. CCP appropriately limits tree growth for 9 datasets, whereas EBP and REP never did so, and MDL did so once. For the 10 datasets that exhibited significant relationships between tree size and training set size, random data reduction still leads to substantially smaller trees (30.11% on average) with little loss in accuracy (less than one tenth of one percent on average).

3 Explaining Overfitting

Why do three common pruning methods fail to control overfitting? Error-based pruning (EBP), reduced-error pruning (REP), and minimum description length pruning (MDL) almost invariably produce trees whose size increases as a function of training set size. Cost complexity pruning (CCP) produces this pathology as well, but less frequently and to a substantially lesser degree than the

three other methods.

3.1 Selecting Among Multiple Trees

The three pruning methods for which overfitting is most pronounced (EBP, REP, and MDL) share a general approach. For each node N_T that forms the root of a subtree, the methods calculate the value of an evaluation function f based on a sample of data $\mathcal{S}$. We call the value of f the *score*, $x_T = f(N_T, \mathcal{S})$. The methods compare the score for the subtree rooted at N_T to a threshold value $\mathcal{T}$. If $x_T > \mathcal{T}$, the subtree is retained, otherwise it is pruned.[4] The threshold $\mathcal{T}$ could be determined in several ways; for EBP, REP, and MDL, the threshold is the score that N_T would receive if it were converted into a leaf node N_L, that is $\mathcal{T} = x_L = f(N_L, \mathcal{S})$.

EBP, REP, and MDL can each be described in terms of this general approach. For example, EBP uses the training sample to calculate an adjusted error rate for a subtree and compares it to the adjusted error rate for a leaf. REP uses a pruning sample to calculate accuracy for the subtree and compares it to the accuracy of the subtree when it is converted into a leaf. MDL uses the training sample to calculate a description length for the subtree and compares it to the description length for the leaf.

Given this approach, when will pruning fail to control overfitting? Pruning will fail if, for a large proportion of subtrees, $x_T > \mathcal{T}$ when the subtree does not improve accuracy. This can occur if the threshold is set too low. With an incorrectly low threshold, subtrees will be judged to be useful when they actually are not, those subtrees will be retained and overfitting will result.

Algorithms that use EBP, REP, and MDL all produce the score x_T in a way that almost guarantees $\mathcal{T}$ will be too low. They: 1) generate n possible subtrees; 2) produce a score for each subtree based on a data sample $\mathcal{S}$; and 3) select the subtree with the maximum score. The pruning methods compare the maximum score $x_{max} = x_T$ to the threshold $\mathcal{T}$.

The procedure above is used to both grow and prune trees. During the *growing* phase, algorithms select each decision node in a subtree by generating and evaluating many possible decision nodes. Each decision node uses a different attribute and a different partitioning of the values of that attribute. For a subtree with D total nodes, a data sample with A attributes, and attributes with an average of P possible partitionings, algorithms select from approximately DAP possible subtrees. During the *pruning* phase, algorithms select each subtree by generating and evaluating many possible pruned subtrees. EBP, REP, and MDL each examine subtrees only after attempting to prune their constituent nodes. For subtrees with D' non-leaf nodes, algorithms select from $2D'$ possible subtrees during pruning, up to and including the entire subtree.

3.2 Why Selection Affects Scores

Why should selecting the maximum score affect the threshold $\mathcal{T}$? Recall that any score x results from applying an evaluation function f to a tree and a data sample $\mathcal{S}$. Suppose an algorithm examines n subtrees with scores $x_1, x_2, \ldots, x_n$. Each score x_i is the value of a random variable. That is, f is a function whose value is a real number determined by a specific sample $\mathcal{S}$. An algorithm examines n subtrees and selects the one with the score $max(x_1, x_2, \ldots, x_n)$.

For simplicity and concreteness, consider a tree-building algorithm that examines two subtrees, with scores x_1 and x_2, and assume that their scores are random variables whose values are drawn from independent uniform distributions of integers $(0, \ldots, 6)$. The distribution of $max(x_1, x_2)$ is shown in Table 2. Each entry in the table represents a joint event with the resulting maximum

[4] In many cases, the score x_T measures error, the inverse of accuracy, and a subtree is retained only if its score is *less than* a threshold. This transformation can be made to the discussion with no loss of generality.

score; for example, $(x_1 = 3 \ \wedge \ x_2 = 4)$ has the result, $max(x_1, x_2) = 4$. Because x_1 and x_2 are independent and uniform, every joint event has the same probability, 1/49, but the probability of a given maximum score is generally higher; for example, $Pr(max(x_1, x_2) = 6) = 13/49$.

		x_1						
		0	*1*	*2*	*3*	*4*	*5*	*6*
	0	0	1	2	3	4	5	6
	1	1	1	2	3	4	5	6
	2	2	2	2	3	4	5	6
x_2	*3*	3	3	3	3	4	5	6
	4	4	4	4	4	4	5	6
	5	5	5	5	5	5	5	6
	6	6	6	6	6	6	6	6

Table 2: The joint distribution of the maximum of two random variables, each of which takes integer values (0...6).

For independent and identically distributed (i.i.d.) random variables $x_1, x_2, \ldots, x_n$, it is easy to specify the relationship between cumulative probabilities of individual scores and cumulative probabilities of maximum scores:

$$\text{If } Pr(x_i < \mathcal{T}) = q, \text{ then } Pr(max(x_1, x_2, \ldots, x_n) < \mathcal{T}) = q^n. \tag{1}$$

For example, in Table 2, $Pr(x_1 < 4) = 4/7$ (and $Pr(x_2 < 4)$ is identical, because x_1 and x_2 are i.i.d.), but $Pr(max(x_1, x_2) < 4) = (4/7)^2 = 16/49$. It is also useful to look at the upper tail of the distribution of the maximum:

$$\text{If } Pr(x_i \geq \mathcal{T}) = p, \text{ then } Pr(max(x_1, x_2, \ldots, x_n) \geq \mathcal{T}) = 1 - (1 - p)^n. \tag{2}$$

These expressions and the distribution in Table 2 make clear that the distribution of any individual random variable x_i from i.i.d. variables $x_1, x_2, \ldots, x_n$ underestimates the distribution of the maximum of all the variables $x_{max} = max(x_1, x_2, \ldots, x_n)$. $Pr(x_i \geq \mathcal{T})$ underestimates $Pr(max(x_1, x_2, \ldots, x_n) \geq \mathcal{T})$ for all values $\mathcal{T}$ if the distributions are continuous. Said differently, the distribution of x_i has a lighter upper tail than the distribution of x_{max}.

This disparity increases with the number of random variables, $x_1, x_2, \ldots, x_n$. Consider three variables distributed in the same way as the two in Table 2. Then,

$$\begin{aligned} Pr(x_i \geq 4) &= 3/7 = 0.43 \\ Pr(max(x_1, x_2, x_3) \geq 4) &= 1 - (1 - 3/7)^3 = 0.81. \end{aligned}$$

The distribution of $Pr(x_i \geq 4)$ underestimates x_{max} by almost half its value. In comparison, $Pr(max(x_1, x_2) \geq 4) = 0.67$.

We have examined this relationship, and its effects on overfitting, in greater detail elsewhere [3].

3.3 What About Cost-Complexity Pruning?

This analysis applies to EBP, REP, and MDL. However, cost-complexity pruning (CCP) is unaffected by the difference in the distributions of x_{max} and x_i. Like the other pruning methods, CCP selects among multiple trees: for each of ten cross-validation test sets, it creates a large number of pruned trees (where each tree is characterized by a pruning parameter α) and selects the one with the maximum score, x_{max}. However, CCP does not compare x_{max} to any threshold. Instead, it combines the α values from each of the cross-validation folds and uses that parameter value to prune the tree grown on the entire dataset. Trees are not selected based on their α values, nor is the selected tree's α compared to some threshold. Therefore, the difference in the distributions of a single score and a maximum score do not affect CCP.

4 Controlling Overfitting

Understanding the mechanism that causes overfitting suggests a solution: a pruning method could set $\mathcal{T}$ such that it adjusts for the number of subtrees examined by an algorithm. Specifically, equation 2 can be used to set a threshold $\mathcal{T}$ so that there is a specified probability that $x_{max} \geq \mathcal{T}$ for any value of n, given that the null hypothesis is true.

Using equation 2 requires knowing the number of independent scores n and the probability that a single score will exceed $\mathcal{T}$. The number of scores n can be set equal to the number of subtrees examined, where the scores of those subtrees are assumed to be independent. The probability that a given score will exceed $\mathcal{T}$ can be obtained from a standard statistical significance test, where the evaluation function f is a statistic such as chi-square or G.

This approach is fairly common in statistics, where equation 2 is known as a *Bonferroni equation*, and the process of using it to determine a threshold is referred to as a Bonferroni adjustment. In this section, we describe an experiment that tests the utility of Bonferroni adjustment in the same way as EBP, REP, MDL, and CCP were evaluated in section 2.

4.1 Building and Pruning Trees with Bonferroni Adjustment

We developed an algorithm — Tree-building with Bonferroni Adjustment (TBA) — that grows and prunes decision trees by using Bonferroni-adjusted significance tests. TBA's basic algorithm is similar to nearly all other algorithms for top-down induction of decision trees [10]. It differs from many other algorithms in its use of Bonferroni-adjusted significance tests, its evaluation function, how it selects partitions during tree construction, how it selects attributes during tree construction, how it selects among possible subtrees during tree pruning, and how it handles missing values.

Bonferroni-adjusted Significance Tests TBA uses Bonferroni-adjusted significance tests in three contexts: selecting the partitioning of an attribute, selecting the attribute to use in a decision node, selecting between subtrees and leaf nodes during pruning. Each of these is covered in more detail below. For all of TBA's significance tests, the adjusted significance level $\alpha_t = 0.10$.

Evaluation function TBA uses the G statistic to evaluate contingency tables during both tree construction and pruning. The G statistic is used because it has a known reference distribution, a requirement for using Bonferroni adjustment. G is computed for contingency tables as follows:

$$G = 2 \sum_{cells} f_{ij} \ln \left(\frac{f_{ij}}{\hat{f}_{ij}} \right), \tag{3}$$

where f_{ij} is the number of occurrences, or frequency, in the cell i, j of the table and $\hat{f}_{ij}$ is the expected value of that cell. In this case, the expected value is $f_{i.}f_{.j}/f_{..}$, where $f_{i.}$ is the total frequency in row i, $f_{.j}$ is the total frequency in column j, and $f_{..}$ is the total of all cells in the table.

Selecting partitions During tree construction, attribute partitions are selected using an approach suggested by Kass [5] and Kerber [6]. For each attribute, a contingency table is constructed with a row for each class value and a column for each of k attribute values — every possible value for discrete attributes or every unique interval for discretized continuous attributes. Then, the pair of columns in the table with the least significant difference is merged. The merging process is repeated until all columns are significantly different. For continuous attributes, only adjacent columns, corresponding to adjacent numeric intervals, can be merged. The result is a node that partitions the sample into j subsamples, where $1 \leq j \leq k$.

The test that determines whether pairs of columns are significantly different uses a Bonferroni adjustment. The exponent n is the number of column pairs. Without this adjustment, an inappropriately large number of partitions would be produced.

Selecting attributes TBA selects attributes based on their probability values. These values are calculated by comparing the G value for the merged contingency table to an appropriate reference distribution, and applying a Bonferroni adjustment. For a table with r rows and c columns, the appropriate reference distribution is chi-square with $(r-1)(c-1)$ degrees of freedom. Following Kass, the Bonferroni exponent n is the number of possible ways of combining k initial categories into j final categories. The calculations differ by attribute type because only adjacent intervals can be merged in continuous attributes:

$$n_{continuous} = \left(\begin{array}{c} j-1 \\ k-1 \end{array} \right); \qquad n_{discrete} = \sum_{i=0}^{k-1} (-1)^i \frac{(k-i)^j}{i!(k-i)!} \tag{4}$$

While these estimates of n are approximate, at best, they provide a rough balance between the total number of possible tables (certainly an overestimate because the tables are highly correlated) and an exponent of 1 (certainly an underestimate). In Kass' experiments with randomly-generated data, they adjusted appropriately for the bias introduced by the merging process.

TBA forms a decision node using the attribute with the lowest probability value, regardless of whether that value exceeds some threshold α. The algorithm uses the decision node to partition the sample into j subsamples based on the attribute's values, and repeats the attribute selection process for each subsample. Tree growth stops when no partition can improve accuracy on the training set.

Pruning After constructing a tree, TBA prunes the tree by examining the probability values calculated during tree construction. Recall that those probability values are adjusted to account for multiple comparisons *within* an attribute. However, they are not yet adjusted to account for multiple comparisons *among* the many attributes that could be used at an individual node, nor are they adjusted to account for the many possible nodes available in other parts of the tree. The latter two adjustments are made at this stage, where the Bonferroni exponent n is the number of attributes considered at that node and the total number of decision nodes at the same tree depth as that node, respectively.

TBA examines each frontier node of the tree — decision nodes that have only leaf nodes as children. Frontier nodes where $p \leq \alpha_t$ are retained; frontier nodes where $p > \alpha_t$ are converted to leaf nodes and labeled with the majority class of the appropriate training subsample. The process

continues until all frontier nodes are significant. Note that this process cannot eliminate non-frontier nodes for which $p > \alpha_t$. This could potentially "trap" insignificant nodes in the interior structure of the tree, but it guards against eliminating potentially useful subtrees.

Handling Missing Values TBA handles missing values by assigning a default class to each decision node (the majority class of the training instances at that node). When the decision tree is used to classify instances, this class is assigned to any instance that reaches a decision node for which it lacks a value for the appropriate attribute. While this approach is easy to implement, it lacks the sophistication of the approaches in other algorithms. For example, C4.5 sends instances that lack attribute values down *all* relevant branches of a node, weighted according to overall frequency in the training set, and then makes a prediction based on the weighted class labels of all branches.

4.2 Results Using TBA

We repeated the experiments from section 2 on TBA. The results are summarized in table 3 along with results reproduced from table 1 for comparison.

Pruning Method	% Kept < 100	p < 0.1	Slope > 0.1	Mean R^2	Mean Δ size	Mean Δ accuracy
EBP	16	16	13	0.90	38.29	-0.14
REP	17	17	11	0.75	39.32	-0.32
MDL	18	17	13	0.88	44.03	-0.37
CCP	19	10	4	0.62	30.11	-0.06
TBA	**16**	**11**	**2**	**0.68**	**36.72**	**-0.18**

Table 3: Summary of the effects of random data reduction for TBA and the other pruning methods.

Based on the results in table 1, TBA performs better than EBP, REP, and MDL and performs similarly to CCP. Its accuracy peaked prior to seeing 100% of the available training instances for 16 of the 19 datasets. Eleven datasets exhibited a significant relationship between tree size and training set size beyond the point at which accuracy stopped growing. However, the slope of the regression line exceeds 0.1 in only two of those datasets. While TBA still exhibits a significant relationship between training set size and tree size in many datasets, the relationship is a relatively weak one in all but two of those cases.

5 Acknowledgments

The authors would like to thank Donato Malerba, Floriana Esposito, and Giovanni Semeraro of the Dipartimento di Informatica, Università degli Studi, Bari Italy for supplying their implementations of reduced error pruning and cost-complexity pruning. M. Zwitter and M. Soklic of the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia provided the `breast cancer` and `lymphography` datasets, and Dr. William H. Wolberg of the the University of Wisconsin Hospitals provided the `breast-cancer-wisc` dataset.

This research was supported by Sterling Software, Inc. subcontract #7335-UOM-001 (DARPA F30602-95-C-0257), and by a National Defense Science and Engineering Graduate Fellowship. The

U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of the Advanced Research Projects Agency, Rome Laboratory or the U.S. Government.

References

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International, 1984.

[2] Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. The MIT Press, Cambridge, 1995.

[3] Paul R. Cohen and David Jensen. Overfitting explained. In *Preliminary Papers of the Sixth International Workshop on Artificial Intelligence and Statistics*, pages 115–122, 1997.

[4] George H. John. Robust decision trees: Removing outliers from databases. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, 1995.

[5] G.V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29(2):199–127, 1980.

[6] Randy Kerber. Chimerge: Discretization of numeric attributes. In *Proceedings of the Tenth National Conference on Artificial Intelligence*. MIT Press, 1992.

[7] J. R. Quinlan. *C4.5 : programs for machine learning*. Morgan Kaufmann Publishers, Inc., 1993.

[8] J. Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

[9] J. Ross Quinlan and R. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

[10] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.