Concise Papers

An Instance-Weighting Method to Induce Cost-Sensitive Trees

Kai Ming Ting, *Member*, *IEEE Computer Society*

Abstract—We introduce an instance-weighting method to induce cost-sensitive trees. It is a generalization of the standard tree induction process where only the initial instance weights determine the type of tree to be induced—*minimum error* trees or *minimum high cost error* trees. We demonstrate that it can be easily adapted to an existing tree learning algorithm. Previous research provides insufficient evidence to support the idea that the greedy divide-and-conquer algorithm can effectively induce a truly cost-sensitive tree directly from the training data. We provide this empirical evidence in this paper. The algorithm incorporating the instance-weighting method is found to be better than the original algorithm in terms of total misclassification costs, the number of high cost errors, and tree size in two-class data sets. The instance-weighting method is simpler and more effective in implementation than a previous method based on altered priors.

Index Terms—Cost-sensitive, decision trees, induction, greedy divide-and-conquer algorithm, instance weighting.

1 Introduction

COST-SENSITIVE classifications have received much less attention than minimum error classifications in empirical learning research. Classifiers that minimize the number of misclassification errors are inadequate in problems with variable misclassification costs. Many practical classification problems have different costs associated with different types of error. For example, in medical diagnosis, the errors committed in diagnosing someone as healthy when they have a life-threatening disease is usually considered to be far more serious (thus, higher cost) than the opposite type of error—diagnosing someone as ill when they are in fact healthy.

A line of research in cost-sensitive tree induction employing the greedy divide-and-conquer algorithm demands further investigation. Breiman et al. [3] describe two different methods of incorporating variable misclassification costs into the process of tree induction. These methods adapt the test selection criterion in the tree growing process. Pazzani et al. [8] reported negative empirical results when using one of Breiman et al.'s [3] formulations to induce cost-sensitive trees. They found that the cost-sensitive trees do not always have lower misclassification costs, when presented with unseen test data, than those trees induced without cost consideration. Using a postprocessing approach, Webb [17] showed that applying a cost-sensitive specialization technique to a minimum error tree can reduce its misclassification costs by about 3 percent on average. Employing the greedy divide-and-conquer algorithm, the research so far does not show convincingly whether a truly cost-sensitive tree can be effectively learned directly from the training data. We investigate this issue specifically in this paper.

This paper presents the instance-weighting method to induce cost-sensitive trees that seeks to minimize the number of high cost errors and, as a consequence of that, leads to minimization of the total misclassification costs in most cases. This method is inspired by instance weight modification in boosting decision trees developed by Quinlan [10]. Boosting generates multiple classifiers in sequential steps. At the end of each step, the weight of each instance in the training set is adjusted to reflect its importance for the next induction step. These weights cause the learner to concentrate on different instances in each step and so leads to different classifiers. These classifiers are then combined by voting to form a composite classifier. Boosting begins with *equal* initial weights in the first step. The intuition for the cost-sensitive induction in this paper is to have *different* initial weights which reflect the (given) costs of misclassification. This effectively influences the learner to focus on instances which have high misclassification costs. We demonstrate that this is a viable method and can be easily adapted to an existing learning algorithm. We show convincingly that a truly cost-sensitive tree can be effectively learned using this method—an algorithm incorporating the instance-weighting method achieves a substantial reduction in misclassification costs, the number of high cost errors, and tree size over the same algorithm without it in two-class domains. It is also found to be competitive with a recent program C5 [11] and better in some aspects.

The proposed instance-weighting method changes the class distribution such that the tree so induced is in favor of the class with high weight/cost and is less likely to commit errors with high cost. This usually reduces the total misclassification costs as a consequence. Smaller trees are a natural product of the tree induction procedure when presented with training data sets of skewed class distribution, which is a result of weighting instances in data sets with relatively balanced class distribution. We present the proposed instance-weighting method in the next section.

2 Cost-Sensitive Tree Induction via Instance-Weighting

Let N be the total number of instances from the given training set and N_j be the number of class j instances. Similarly, let $N(t)$ and $N_j(t)$ be the number of instances and class j instances in node t of a decision tree. The probability that an instance is in class j given that it falls into node t is given by the ratio of the total number of class j instances to the total number of instances in this node.

$$p(j|t) = \frac{N_j(t)}{\sum_i N_i(t)}. \tag{1}$$

When node t contains instances that belong to a mixture of classes, the standard greedy divide-and-conquer procedure for inducing trees (e.g., [3] and [9]) uses a test selection criterion to choose a test at this node such that the training instances which fall into each branch, as a result of the split, become more homogeneous. One of the commonly used criteria is entropy, that is, $-\sum_j p(j|t)log[p(j|t)]$. At each node, the tree growing process selects a test which has the maximum gain in entropy until the node contains only a single-class collection of instances.

To avoid overfitting, a pruning process is employed to reduce the size of the tree such that the estimated error is a minimum. In short, the standard tree induction procedure seeks to produce a *minimum error* tree.

Our intuition for cost-sensitive tree induction is to modify the weight of an instance proportional to the cost of misclassifying the class to which the instance belonged, leaving the sum of all training instance weights still equal to N. The last condition is important because there is no reason to alter the size of the training set, which is equivalent to the sum of all training instance weights,

• *The author is with the Gippsland School of Computing and Information Technology, Monash University, Churchill, Victoria 3842, Australia. E-mail: kaiming.ting@infotech.monash.edu.au.*

Manuscript received 28 June 1999; revised 13 Sept. 2000; accepted 4 Jan. 2001; posted to Digital Library 7 Sept. 2001.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 110135.

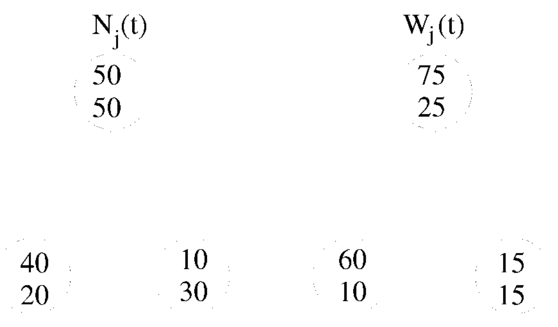

Fig. 1. Splitting on the same test—using unit instance weights (left) and different instance weights (right).

while the individual instance weights are adjusted to reflect the relative importance of instances for making future prediction with respect to cost-sensitive classification.

Let $C(j)$ be the cost of misclassifying a class j instance; the weight of a class j instance can be computed as

$$w(j) = C(j)\frac{N}{\sum_i C(i)N_i} \quad (2)$$

such that the sum of all instance weights is $\sum_j w(j)N_j = N$. For $C(j) \geq 1$, $w(j)$ has the smallest value $0 < \frac{N}{\sum_i C(i)N_i} \leq 1$ when $C(j) = 1$ and the largest value

$$w(j) = \frac{C(j)\sum_i N_i}{\sum_i C(i)N_i} \geq 1$$

when $C(j) = max_i C(i)$.

Similar to $p(j|t)$, $p_w(j|t)$ is defined as the ratio of the total weight of class j instances to the total weight in node t:

$$p_w(j|t) = \frac{W_j(t)}{\sum_i W_i(t)} = \frac{w(j)N_j(t)}{\sum_i w(i)N_i(t)}. \quad (3)$$

The standard greedy divide-and-conquer procedure for inducing minimum error trees can then be used without modification, except that $W_j(t)$ is used instead of $N_j(t)$ in the computation of the test selection criterion in the tree growing process and the error estimation in the pruning process. Thus, both processes are affected due to this change.

We modified C4.5 [9] to create C4.5CS. We only need to initialize the training instance weights to $w(j)$ since C4.5 has already employed $W_j(t)$ for the computation discussed above.[1]

This modification effectively converts the standard tree induction procedure that seeks to minimize *the number of errors*, regardless of cost, to a procedure that seeks to minimize *the number of errors with high weight or cost*. Note that minimizing the latter does not guarantee that the total misclassification cost is minimized; this is because the number of low cost errors is usually increased as a result.

The advantage of this approach is that the whole process of tree growing and tree pruning is the same as that used to induce minimum error trees. This can be viewed as a generalization of the standard tree induction process where *only the initial instance weights determine the type of tree to be induced—minimum error trees or minimum high cost error trees*.

Fig. 1 shows an example of a split on the same attribute test using unit instance weights (in the left figure) and different instance weights (in the right figure). The sum of the instance weights for each class are shown in each node. With unit weights, each sum is equivalent to the number of instances for each class $N_j(t)$. This example has two equiprobable classes, where $N_1 = N_2 = 50$ at the root of the tree. The right figure shows the result of the same split when $C(1) = 3$ and $C(2) = 1$. Employing (1), the weights of all instances are modified to $w(1) = 1.5$ and $w(2) = 0.5$. As a result, the sums of the class j instance weights at the root are $W_1 = 75$ and $W_2 = 25$. This example shows that initializing the instance weights to $w(j)$ amounts to changing the class distribution of the training data.

To classify a new instance, C4.5CS predicts the class which has the maximum weight at a leaf, as in C4.5.

Here, we describe how the cost of misclassification can be specified in a cost matrix and how the cost matrix is related to $C(j)$ in (2). In a classification task of I classes, the misclassification costs can be specified in a cost matrix of size $I \times I$. The rows of the matrix indicate the predicted class and the column indicates the actual class. The off-diagonal entries contain the costs of misclassifications; and on the diagonal lie the costs for correct classifications, which are zero in this case since our main concern here is total misclassification costs of an induced tree.[2]

Let $cost(i,j)$ be the cost of misclassifying a class j instance as belonging to class i. In all cases, $cost(i,j) = 0.0$ for $i = j$. A cost matrix must be converted to a cost vector $C(j)$ in order to use (2) for instance-weighting. In this paper, we employ the form of conversion suggested by Breiman et al. [3]:

$$C(j) = \sum_i^I cost(i,j). \quad (4)$$

In our experiments, without loss of generality, we impose a unity condition—at least one $cost(i,j) = 1.0$, which is the minimum misclassification cost. The only reason to have this unity condition or normalization[3] is to allow us to measure the number of *high cost errors*, which is defined as the number of misclassification errors that have costs more than 1.0.

Note that the cost matrix to cost vector conversion is expected to work well with the cost-sensitive tree induction, as described in this section, when there are only two classes. But, it might be inappropriate when there are more than two classes because it collapses $I \times I$ numbers to I. In order to investigate the potential problem due to this conversion, we explicitly divide the experimental data sets into two groups: two-class and multiclass. Any performance discrepancy between these two groups is due to this conversion.

3 EXPERIMENTS

Four measures are used to evaluate the performance of the cost-sensitive tree induction algorithm in this paper. They are total misclassification costs (i.e.,

$$\sum_l^{N'} cost(\text{predicted-class}(l), \text{actual-class}(l)),$$

where N' is the number of instances in the unseen test set), pruned tree size (i.e., total number of internal nodes and leaves), the number of high cost errors, and the total number of misclassification errors on unseen data. The first and the third are the most important measures. While the aim of cost-sensitive classification is to minimize the total misclassification costs, it is important to measure the number of high cost errors since the instance-

1. C4.5 uses fractional weights for the treatment of missing values. See [9] for details.

2. In general, the costs of correct classifications can be nonzero. Minimizing the costs of correct classifications is a different issue outside the scope of this paper.

3. Note that an arbitrary cost matrix can be normalized to become a cost matrix satisfying this unity condition.

TABLE 1
Details of the Data Sets Used in the Experiment

Datasets	# Instances	# Classes	# Attr			Default Accuracy %
Echocardiogram	131	2	1B		6C	67.2
Hepatitis	155	2	13B		6C	79.4
Heart(Statlog)	270	2			13C	55.6
Heart	303	2			13C	54.1
Horse	368	2	3B	12N	7C	63.0
Credit	690	2	4B	5N	6C	55.5
Breast-W	699	2			9C	65.5
Diabetes	768	2			8C	65.1
GermanCredit	1000	2			24C	70.0
Euthyroid	3163	2	18B		7C	90.7
Hypothyroid	3163	2	18B		7C	95.2
Coding	20000	2		15N		50.0
Lymphography	148	4	9B	9N		54.7
Glass	214	6			9C	35.7
Waveform	300 5000	3			40C	33.3
Soybean	683	19	16B	19N		23.8
Annealing	898	6	19B	13N	6C	76.2
Vowel	990	11			10C	9.5
Splice	3177	3		60N		51.9
Abalone	4177	3		1N	7C	52.7
Nettalk(s)	5438	5		7N		40.1
Satellite	6435	6			36C	23.8

N-nominal; B-binary; C: Continuous.

weighting method is designed to achieve the aim through high cost error minimization. All other factors being equal, one tree induction algorithm is better than another if it induces smaller trees.

We conduct experiments using 20 data sets obtained from the UCI repository of machine learning databases [1] and two data sets with specified cost matrices (i.e., Heart_S and German) used in the Statlog project [5]. The data sets are selected to cover a wide variety of different domains with respect to data set size, the number of classes, the number of attributes, and the types of attributes. They consist of 12 two-class data sets and 10 multiclass data sets. The details of these data sets are given in Table 1.

Ten 10-fold cross-validations[4] [3] are carried out in each data set, except in the Waveform data set where randomly generated training data size of 300 and test data size of 5,000 are used in the 100 trials.

Random cost assignments with the unity condition are used in all data sets except the Heart_S and German data sets. In the latter cases, the costs (i.e., $cost(1,2) = 1.0$ and $cost(2,1) = 5.0$) specified in [5] are used. In the former cases, a cost matrix is randomly generated at the beginning of each trial. Each nondiagonal entry in the cost matrix is assigned an integer randomly generated between 1 to 10.

We first compare C4.5CS with C4.5 to evaluate whether trees induced by C4.5CS are more cost sensitive than those produced by C4.5. Note that *the only difference between C4.5CS and C4.5 is the initial weight setting*, so any performance differences are due to this initial weight setting. Then, we compare C4.5CS to C5 [11] and other algorithms reported in the Statlog project.

3.1 Can C4.5CS Induce Cost-Sensitive Trees Effectively?

Given a training set and a cost matrix, C4.5CS induces a cost-sensitive tree which seeks to minimize the number of high cost errors and total misclassification costs. C4.5 produces a tree which seeks to minimize the total misclassification errors. Both trees are then tested using a separate test set and the total misclassification costs are measured according to the given cost matrix.

Table 2 presents averages, over 100 trials, for the misclassification costs, the tree size, the number of high cost errors, and the total errors for both C4.5CS and C4.5 in each data set . The ratio (C4.5CS/C4.5) for each of these measures is also presented—a value less than 1 represents an improvement due to C4.5CS. The means of these ratios are given for the 12 two-class data sets as well as the 10 multiclass data sets.

In terms of misclassification costs, C4.5CS achieves a mean reduction of 38 percent compared to C4.5 in two-class data sets, but a mean reduction of only 2 percent in multiclass data sets.

In terms of tree size, C4.5CS produces trees 34 percent smaller than those produced by C4.5 in two-class data sets, and 15 percent smaller in multiclass data sets. In only two data sets (Hypothyroid and Euthyroid) does C4.5CS produce trees which are larger than those produced by C4.5. This is because the two data sets have very skewed class distribution; 95.2 percent and 90.7 percent of the total instances belong to one of the two classes in these two data sets, respectively. A high cost $C(j)$ assigned to the class which has a small number of instances effectively reduces the class distribution skewness, leading to larger trees. Although the costs are randomly assigned without reference to the original class distribution, reduction in skewness seems to have a larger effect than increase in skewness in these two data sets.

C4.5CS makes 65 percent fewer high cost errors than C4.5 in two-class data sets, but 2 percent more high cost errors in multiclass data sets. On the other hand, C4.5CS has 41 percent more errors than C4.5 in two-class data sets, but only 6 percent more errors in multiclass data sets.

Hypothyroid is the only two-class data set in which C4.5CS has higher misclassification costs (by 3 percent) than C4.5. While C4.5CS is able to reduce the number of high cost errors by 19 percent in this highly skewed class distribution data set, the 74 percent increase in total errors outweighs this reduction, resulting in a net increase in total misclassification costs.

3.2 Minimum Expected Cost Criterion

A simple method of using a minimum error tree for cost-sensitive classifications is to employ the minimum expected cost criterion in selecting a predicted class during classification [5]. It is interesting to find out how the proposed method compares to this simple method.

The expected misclassification cost for predicting class i with respect to the instance x is given by:

$$EC_i(x) \propto \sum_j W_j(t(x)) cost(i,j), \tag{5}$$

where $t(x)$ is the leaf of the tree that instance x falls into and $W_j(t)$ is the total weight of class j training instances in node t.

To classify a new instance x using a minimum error tree with the minimum expected cost criterion, $EC_i(x)$ is computed for every class. The instance x is assigned to class i with the smallest value for $EC_i(x)$; that is, $EC_i(x) < EC_{i'}(x)$ for all $i' \neq i$.

A comparison between C4.5CS_mc and C4.5_mc, both using the minimum expected cost criterion, is presented here. The results in Table 3 show that it is still better to induce a cost-sensitive tree than a minimum error tree for cost-sensitive classifications in two-class data sets, even using the minimum expected cost criterion. Note that this criterion affects only the classification process, not the induction process. Thus, the tree size does not change for C4.5CS and C4.5CS_mc, as well as for C4.5 and C4.5_mc.

It is interesting to know what the effect of the criterion is on C4.5CS. An experiment using the 12 two-class data sets shows that the criterion increases, average misclassification cost of C4.5CS by 5 percent but significantly reduces the average number of high cost

4. In a 10-fold cross-validation, the given data set is randomly divided into 10 equal size subsets. In each fold, nine subsets are used as the training and the remaining subset as the test set. This is repeated 10 times and each subset is used as the test set only once.

TABLE 2
C4.5CS vs. C4.5 in Terms of Misclassification Cost, Tree Size, Number of High Cost Errors and Total Number of Errors

Datasets	Cost			Tree Size			No. HC Errors			No. Errors		
	C4.5CS	C4.5	ratio	C4.5CS	C4.5	ratio	C4.5CS	C4.5	ratio	C4.5CS	C4.5	ratio
Echo	7.9	18.2	.44	6.0	10.8	.56	0.5	2.5	.22	5.4	4.7	1.15
Hepa	6.0	12.3	.48	9.5	17.0	.56	0.4	1.6	.27	3.9	3.4	1.15
Heart_S	10.9	17.1	.64	16.7	35.6	.47	1.0	2.8	.37	6.8	5.9	1.15
Heart	14.2	25.2	.56	18.2	39.5	.46	1.1	3.3	.34	8.7	6.6	1.32
Horse	16.4	21.8	.75	8.6	11.6	.74	1.1	2.9	.39	11.6	5.8	2.00
Credit	21.3	36.5	.58	10.5	33.2	.32	1.5	4.9	.31	13.7	10.2	1.34
Breast	9.9	14.4	.69	14.8	23.8	.62	1.0	2.0	.51	4.9	3.6	1.36
Diab.	35.8	69.6	.51	18.4	41.9	.44	2.0	9.4	.21	27.5	19.8	1.39
German	30.3	72.8	.42	2.2	149.3	.01	0.1	11.4	.01	29.9	27.2	1.10
Hypo	9.0	8.7	1.03	24.6	12.2	2.02	0.9	1.1	.81	4.0	2.3	1.74
Euthyr	20.7	24.2	.85	39.8	25.3	1.57	2.1	3.3	.64	10.1	6.3	1.60
Coding	942.2	2058.6	.46	302.7	2805.6	.11	22.2	278.2	.08	880.8	554.0	1.59
Mean			.62			.66			.35			1.41
Lympho	17.4	18.2	.96	19.0	27.4	.69	2.9	2.9	1.00	3.3	3.2	1.03
Glass	35.8	38.4	.93	39.2	45.5	.86	6.0	6.3	.95	6.9	7.1	.97
Wave	7119.0	7709.3	.92	42.7	51.0	.84	1177.9	1232.4	.96	1522.1	1522.1	1.00
Soybean	33.8	30.4	1.11	91.1	96.4	.95	5.7	5.2	1.11	6.2	5.6	1.11
Anneal	35.4	35.7	.99	76.7	76.6	1.00	6.4	6.0	1.06	7.3	6.7	1.09
Vowel	115.1	111.8	1.03	175.2	187.0	.94	19.1	18.2	1.05	21.1	20.3	1.04
Splice	95.7	95.6	1.00	157.1	171.6	.92	16.2	15.3	1.06	22.7	18.6	1.22
Abalone	708.7	799.2	.89	402.1	579.2	.69	124.6	129.2	.96	168.4	161.9	1.04
Net(s)	507.9	514.2	.99	1650.4	2061.6	.80	86.0	84.2	1.02	99.6	94.0	1.06
Sat.	475.3	478.5	.99	472.4	561.2	.84	78.4	78.5	1.00	88.1	88.1	1.00
Mean			.98			.85			1.02			1.06

errors by 60 percent. A direct comparison between C4.5CS and C4.5_mc shows, that to minimize cost, the instance-weighting method does more than substitute the minimum expected cost criterion as the cost ratio for C4.5CS versus C4.5_mc stays at about the same level shown in Table 3.

3.3 MultiClass Problems

In this section, we study the behavior of C4.5CS using three different types of cost matrix in the 10 multiclass data sets. We first define three types of cost matrix, which give rise to different definitions of $C(j)$. They are defined as follows:

1. $cost(i,j) > H$ only for a single value of $i = I$ and $cost(i \neq I, j) = H$ for all $i \neq j$. We define

$$C(j) = cost(I,j) \text{ for } j \neq I \text{ and } C(I) = H.$$

2. This type of cost matrix has a constant misclassification cost for each class, i.e., $cost(i,j) = H_j \geq 1.0$ for each $j \neq i$. Thus, it can be expressed as a cost vector

$$C(j) = H_j.$$

3. In this more general case, $cost(i,j) \geq 1.0$ for all $i \neq j$. This is the type of matrix we used for the previous experiments and a possible form of $C(j)$ is defined earlier in (4).

Examples of these matrices are shown in Table 4.

For experiments in this section, without loss of generality, we impose the following unity condition: For Type 1 cost matrix, $H = 1.0$, for Type 2 cost matrix, at least one $H_j = 1.0$, and for Type 3 cost matrix, at least one $cost(i,j) = 1.0$, where the unit cost is the minimum cost.

TABLE 3
Mean Ratios for C4.5CS_mc against C4.5_mc

	Two-class	Multi-class
Misclassification Cost ratio	.86	.99
Tree Size ratio	.66	.85
No. High Cost Errors ratio	.36	1.02
No. Errors ratio	1.57	1.07

TABLE 4
Examples of Three Types of Cost Matrix, $cost(i,j)$

		Type (a)				Type (b)				Type (c)		
			j				j				j	
		1	2	3		1	2	3		1	2	3
	1	0.0	3.0	6.0	1	0.0	3.0	7.0	1	0.0	3.0	2.0
i	2	1.0	0.0	1.0	2	1.0	0.0	7.0	2	1.0	0.0	7.0
	3	1.0	1.0	0.0	3	1.0	3.0	0.0	3	10.0	5.0	0.0

TABLE 5
Mean Ratios for C4.5CS against C4.5 Over 10 Multiclass Data Sets

	Multi-class		
Cost matrix type	(a)	(b)	(c)
Misclassification Cost ratio	.80	.87	.98
Tree Size ratio	.71	.70	.85
No. High Cost Errors ratio	.33	.90	1.02
No. Errors ratio	1.30	1.30	1.06

Table 5 presents the mean ratios for C4.5CS against C4.5 using the three types of cost.

In comparison to C4.5, C4.5CS performs better in terms of misclassification costs, tree size, and the number of high cost errors for all three types of cost matrix, except that it is making 2 percent more high cost errors when using Type 3 cost matrix. Among these cost matrices, C4.5, and its cost-sensitive counterpart seem to be performing most similarly using Type 3 cost matrix, and least similarly using Type 1 cost matrix. This is evident since the mean ratios for misclassification costs and the number of high cost errors increase from Type 1 to Type 2 and to Type 3 cost matrices. This suggests that the conversion from $cost(i,j)$ to $C(j)$ is most effective for Type 1 cost matrix but least effective for Type 3 cost matrix.

To maintain generality, we have used Type 3 cost matrix as the default cost matrix in other parts of this paper.

3.3.1 Summary

We summarize the findings so far as follows:

- In terms of misclassification costs and the number of high cost errors, C4.5CS performs better than C4.5 in two-class data sets, but is only comparable for multiclass data sets.
- The relatively poor performance of C4.5CS in multiclass data sets is due to the conversion of cost matrix to cost vector.
- C4.5CS always makes fewer high cost errors than C4.5 in two-class data sets, but, in data sets with highly skewed class distribution, C4.5CS might have higher total misclassification costs than C4.5.
- C4.5CS produces smaller trees than C4.5 because instance weighting effectively increases the skewness of the otherwise more balanced class distribution.
- Even using the minimum expected cost criterion, it is better to induce a cost-sensitive tree than to induce a minimum error tree for cost-sensitive classifications in two-class data sets. The instance-weighting method produces an effect that is significantly better than simply applying the criterion.

3.4 How Does C4.5CS Compare to C5?

In this experiment, we compare C4.5CS to the improved version of C4.5, C5 [11], which can produce a cost-sensitive tree when given a cost matrix.

The results in Table 6 show that C4.5CS performs comparably to C5 in terms of misclassification costs; the mean ratio is .99 over 22 data sets. In only six data sets are the ratios for misclassification costs outside the range $1.00 \pm .10$.

In terms of tree size, C4.5CS produces smaller trees than those produced by C5 in 20 out of the 22 data sets. On average, C4.5CS induces trees which are 14 percent smaller. It is important to note that the two exceptions, the Annealing and Nettalk(s) data sets, are

TABLE 6
Ratios for C4.5CS vs. C5

Datasets	Cost ratio	Tree Size ratio	No.HC Errors ratio	No.Errors ratio
Echocardiogram	1.04	.94	1.09	1.00
Hepatitis	.98	.87	1.00	.97
Heart(Statlog)	.94	.80	.83	1.03
Heart	.90	.80	.82	1.00
Horse	1.00	.84	1.14	.94
Credit	1.01	.83	.98	1.02
Breast-W	.91	.85	.87	.98
Diabetes	.99	.83	.94	1.02
GermanCredit	1.00	.88	.71	1.00
Hypothyroid	.98	.91	1.02	.97
Euthyroid	1.04	.89	1.00	1.05
Coding	1.00	.68	.91	1.02
Lymphography	.97	.72	.70	1.05
Glass	.94	.73	.60	1.10
Waveform	.82	.68	.63	1.08
Soybean	1.20	.91	1.75	1.17
Annealing	1.05	1.08	.89	1.11
Vowel	1.03	.82	.45	1.23
Splice	1.12	.66	.70	1.50
Abalone	.87	.62	.44	1.03
Nettalk(s)	1.14	1.81	.89	1.21
Satellite	.87	.70	.49	1.09
Mean	.99	.86	.86	1.07

due to a simpler C5 representation for nominal attribute tests next to the leaves of the tree. In C4.5, when a decision node next to the leaves is a test using a nominal attribute, the number of leaves is the same as the number of possible values for this attribute. This usually results in many leaves with no covered instance, especially when the nominal attribute has many values. C5 simplifies the representation by collecting all zero-instance leaves into a single leaf with a subset branch. When we simplify the C4.5CS representation, as in C5, the tree sizes reduce from 76.7 to 55.5 for the Annealing data set and from 1650.4 to 566.2 for the Nettalk(s) data set , which are both smaller than the trees produced by C5. Taking these new figures into account, the mean ratio for tree size is reduced from .86 to .80.

Although trees produced by C4.5CS have more misclassification errors (i.e., 7 percent on average) than those by C5, C4.5CS is less likely to make high cost errors. This feature is particularly important when the aim is to minimize the number of high cost errors. C4.5CS makes 14 percent fewer high cost errors than C5 on average. This effect is apparent in 16 out of the 22 data sets.

Unfortunately, no description is given of the cost-sensitive induction method employed in C5 [11] and we have no access to source code, except that used for classifying from an induced tree which is in the public domain. The latter reveals that C5 predicts a class which has the minimum expected costs, whereas C4.5CS (like C4.5) predicts a class which has the maximum weights. The

TABLE 7
Average Misclassification Costs for Five Classifers

Dataset	C4.5CS	C5	CART	Discrim	NaiveBayes
Heart(Statlog)	0.404	0.430	†0.452(8)	†0.393(2)	†0.374(1)
GermanCredit	0.303	0.304	0.613(8)	0.535(1)	0.703(12)

†: the 9-fold cross-validation is used, and others use 10-fold.
(n): ranking in the Statlog project; n=1 is the best result.

significantly smaller trees produced by C4.5CS are also another indication that the cost-sensitive induction method used in C5 is different from that in C4.5CS.

3.5 Comparison with the Results from the Statlog Project

The Statlog project [5] studied a number of classifiers for cost-sensitive classification in the Heart and GermanCredit data sets. We restate their results for three classifiers for comparison. The classifiers are CART [3], linear discriminant (Discrim), and NaiveBayes. The Statlog project uses average misclassification costs (the ratio of the total misclassification costs and the total number of test instances) for evaluation. We convert our results for C4.5CS and C5 to this measure and the results for the five classifiers are tabulated in Table 7.

The results show that C4.5CS performs better than CART in both data sets. It performs marginally worse than Discrim and NaiveBayes in the Heart data set , but significantly better than both classifiers in the GermanCredit data set .

4 Relation to Altered Priors

Breiman et al. [3] discuss a method of incorporating variable misclassification costs via altered priors for cost-sensitive tree induction. Let priors $\pi(j) = N_j/N$ and $C(j)$ as defined in (4); then, the altered priors are given by [3]

$$\pi'(j) = \frac{C(j)\pi(j)}{\sum_i C(i)\pi(i)} = \frac{C(j)N_j}{\sum_i C(i)N_i}.$$

In the instance-weighting method, every instance is weighted proportional to $C(j)$. The weight of a class j instance is computed as

$$w(j) = \frac{C(j)N}{\sum_i C(i)N_i} = \pi'(j)N/N_j = \frac{\pi'(j)}{\pi(j)}.$$

Thus, the instance weight is a ratio of the altered prior and the original prior. Both methods share the same idea of changing the class distribution according to the given misclassification costs, but one implementation is simpler and more effective than the other. Implementation using altered priors or by merely modifying instance weights will produce the same tree at the end of the tree growing process, but the former would require an equivalent modification in the pruning process; otherwise, it will perform poorly. This is demonstrated by modifying C4.5 accordingly to yield $\mathrm{C4.5}(\pi')$. Because instance weights are not altered, the tree induced by $\mathrm{C4.5}(\pi')$ will be pruned according to unit instance weights.

The mean ratios $(\mathrm{C4.5CS}/\mathrm{C4.5}(\pi'))$ for misclassification cost and the number of high cost errors are .70 and .43, respectively, averaged over the 12 two-class data sets. $\mathrm{C4.5}(\pi')$ is significantly worse than C4.5CS for the two important measures in cost-sensitive classifications. The poor result of $\mathrm{C4.5}(\pi')$ is due to the inconsistent use of instance weights from the tree growing process to the tree pruning process.

5 Discussion

The results presented in Table 2 clearly show that the greedy divide-and-conquer algorithm can effectively induce a cost-sensitive tree directly from the training set, given a cost matrix. A previous reported method—cost-sensitive specialization [17], which also applied to C4.5, only achieves a cost reduction of 3 percent on average over 20 data sets.[5] This method maintains the same size of the tree produced by C4.5. In comparison, the instance-weighting method achieves a 38 percent reduction in costs and produces trees which are 34 percent smaller in the two-class data sets.

The idea of using fractional instance weights in decision tree induction has been applied in C4.5 [9]. In the case of missing value, C4.5 partitions the training set using fractional weights in evaluating a test during tree induction and classification. Quinlan [10] further develops the idea to boosting decision tree algorithms. In both cases, *equal* initial weights are used. We extend this idea to *different* initial weights, which we demonstrate to be a simple and viable approach for extending the tree induction algorithm to include cost-sensitive classifications.

Indeed, the instance-weighting method only requires the weights to be proportional to the misclassification cost. For example, instead of (2), one can use $w(j) = C(j)$ to weight the instances and induce the same cost-sensitive tree as that produced by C4.5CS. This is fine if the task requires only one tree to be produced. However, in the case where multiple trees are needed, as in the case of boosting ([12], [14]) in which it is important to maintain identical total initial weight for each tree induction, then (2) is essential to the process.

Pazzani et al. [8] show that trees induced using the Gini criterion (the test selection criterion as used in CART) with altered priors perform worse than those induced using only the Gini criterion in terms of misclassification costs in both two-class and multiclass data sets. Pazzani et al. [8] attribute the negative result to the conversion of cost matrix $cost(i, j)$ to cost vector $C(j)$. We disagree with this explanation, suspecting that their negative result is due to the use of two different priors in the tree growing and pruning processes, as shown in our experiment in Section 4. Our results indicate that a tree induced with altered priors, together with the corresponding modification in the pruning method, should perform better than those without. Also, we think the difference is not due to the test selection criterion employed (C4.5 uses the entropy criterion rather than the Gini criterion).

However, the conversion does result in a weakness in the instance-weighting method. This method will induce the same tree for different cost matrices whenever these matrices convert to the same cost vector $C(j)$. This is undesirable for multiclass problems. While other forms of conversion are possible (for example, $C(j) = max_i\ cost(i, j)$), they suffer from the same problem. One possible remedy is to conduct tree pruning using $cost(i, j)$ directly. However, so far we have not found an effective method for this kind of pruning. Bradford et al. [2] and Knoll et al. [4] studied several cost-sensitive pruning methods and reported that no pruning method dominated over all the data sets they studied. One way to mitigate this problem is to generate multiple trees and then combine them to produce a final prediction. We have suggested a cost-sensitive version of boosting [12] to address both this weakness and the problem with skew class distribution reported in [15] and [14].

5. There are 10 same data sets which are used in both Webb's and our experiments. We use nine data sets of size more than 900, whereas Webb uses only two.

There are several tree induction algorithms that consider the costs of tests, such as EG2 [7], CS-ID3 [13], and IDX [6]. Turney [16] investigates both the costs of tests and misclassifications using a genetic algorithmic search in tree induction. We do not consider costs of tests in this paper to avoid complicating the issue under investigation.

6 Conclusions

We have introduced an instance-weighting method to induce cost-sensitive trees and demonstrated that it is a viable approach, which is simple to implement or adapt to an existing learning algorithm. It is a generalization of the standard tree induction process to include both minimum error trees and minimum high cost error trees. The instance-weighting method is simpler and more effective in implementation than an existing method based on altered priors.

Our empirical results show convincingly that the greedy divide-and-conquer procedure can effectively induce a truly cost-sensitive tree directly from the training data. This work refutes an earlier negative result [8] with regard to cost-sensitive tree induction employing the greedy divide-and-conquer procedure in two-class data sets.

The algorithm incorporating the instance-weighting method is found to be better than the original algorithm in two-class data sets in terms of the number of high cost errors, total misclassification costs, and tree size. The instance weighting which changes the class distribution directly contributes to this improved performance.

Acknowledgments

The author would like to thank the anonymous reviewers, Zijian Zheng, Geoff Webb, and Albert Goodman for providing helpful suggestions to improve this paper. The author also would like to thank Ross Quinlan for providing C4.5.

References

[1] C. Blake, E. Keogh, and C.J. Merz, "UCI Repository of Machine Learning Databases," Univ. of California, Dept. Information and Computer Science, http://www.ics.uci.edu/~mlearn/MLRepository.html, 1998.

[2] J.P. Bradford, C. Kunz, R. Kohavi, C. Brunk, and C.E. Brodley, "Pruning Decision Trees with Misclassification Costs," *Proc. 10th European Conf. Machine Learning,* pp. 131-136, 1998.

[3] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, *Classification and Regression Trees.* Belmont, Cailf.: Wadsworth, 1984.

[4] U. Knoll, G. Nakhaeizadeh, and B. Tausend, "Cost-Sensitive Pruning of Decision Trees," *Proc. Eighth European Conf. Machine Learning,* pp. 383-386, 1994.

[5] D. Michie, D.J. Spiegelhalter, and C.C. Taylor, *Machine Learning, Neural and Statistical Classification.* Ellis Horwood Limited, 1994.

[6] S.W. Norton, "Generating Better Decision Trees," *Proc. 11th Int'l Joint Conf. Artificial Intelligence,* pp. 800-805, 1989.

[7] M. Núñez, "The Use of Background Knowledge in Decision Tree Induction," *Machine Learning,* vol. 6, pp. 231-250, 1991.

[8] M. Pazzani, C. Merz, P. Murphy, K. Ali, T. Hume, and C. Brunk, "Reducing Misclassification Costs," *Proc. 11th Int'l Conf. Machine Learning,* pp. 217-225, 1994.

[9] J.R. Quinlan, *C4.5: Program for Machine Learning.* Morgan Kaufmann, 1993.

[10] J.R. Quinlan, "Boosting, Bagging, and C4.5," *Proc. 13th Nat'l Conf. Artificial Intelligence,* pp. 725-730, 1996.

[11] J.R. Quinlan, "C5," http://rulequest.com, 1997.

[12] R.E. Schapire, Y. Freund, P. Bartlett, and W.S. Lee, "Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods," *Proc. 14th Int'l Conf. Machine Learning,* pp. 322-330, 1997.

[13] M. Tan, "Cost-Sensitive Learning of Classification Knowledge and Its Applications in Robotics," *Machine Learning,* vol. 13, pp. 7-33, 1993.

[14] K.M. Ting, "A Comparative Study of Cost-Sensitive Boosting Algorithms," *Proc. 17th Int'l Conf. Machine Learning,* pp. 983-990, 2000.

[15] K.M. Ting and Z. Zheng, "Boosting Cost-Sensitive Trees," *Proc. First Int'l Conf. Discovery Science,* pp. 244-255, 1998.

[16] P.D. Turney, "Cost-Sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm," *J. Artificial Intelligence Research,* vol. 2, pp. 369-409, 1995.

[17] G.I. Webb, "Cost-Sensitive Specialization," *Proc. 1996 Pacific Rim Int'l Conf. Artificial Intelligence,* pp. 23-34, 1996.