Multiclass Alternating Decision Trees

Geoffrey Holmes, Bernhard Pfahringer, Richard Kirkby,
Eibe Frank and Mark Hall

Department of Computer Science
University of Waikato
Hamilton, New Zealand
{geoff, bernhard, rkirkby, eibe, mhall}@cs.waikato.ac.nz

Abstract. The alternating decision tree (ADTree) is a successful classification technique that combines decision trees with the predictive accuracy of boosting into a set of interpretable classification rules. The original formulation of the tree induction algorithm restricted attention to binary classification problems. This paper empirically evaluates several wrapper methods for extending the algorithm to the multiclass case by splitting the problem into several two-class problems. Seeking a more natural solution we then adapt the multiclass LogitBoost and AdaBoost.MH procedures to induce alternating decision trees directly. Experimental results confirm that these procedures are comparable with wrapper methods that are based on the original ADTree formulation in accuracy, while inducing much smaller trees.

1 Introduction

Boosting is now a well established procedure for improving the performance of classification algorithms. AdaBoost [8] is the most commonly used boosting procedure, but others have gained prominence [3, 10]. Like many classification algorithms, most boosting procedures are formulated for the binary classification setting. Schapire and Singer generalize AdaBoost to the multiclass setting producing several alternative procedures of which the best (empirically) is AdaBoost.MH [14]. This version of AdaBoost covers the multilabel setting where an instance can have more than one class label as well as the multiclass setting where an instance can have a single class label taken from a set of (more than two) labels.

Alternating decision trees are induced using a real-valued formulation of AdaBoost [14]. At each boosting iteration three nodes are added to the tree. A splitter node that attempts to split sets of instances into pure subsets and two prediction nodes, one for each of the splitter node's subsets. The position of this new splitter node is determined by examining all predictor nodes choosing the position resulting in the globally best improvement of the purity score.

Essentially, an ADTree is an AND/OR graph. Knowledge contained in the tree is distributed as multiple paths must be traversed to form predictions. Instances that satisfy multiple splitter nodes have the values of prediction nodes that they reach summed to form an overall prediction value. A positive sum represents one class and a negative sum the other in the two-class setting. The result is a single interpretable tree with predictive capabilities that rival a committee of boosted C5.0 trees [7].

An additional attractive feature of ADTrees, one that is not possible with conventional boosting procedures, is their ability to be merged together. This is a particularly useful attribute in the context of multiclass problems as they are often re-formulated in the two-class setting using one or more classes against the others. In such a setting ADTrees can be combined into a single classifier.

In their original exposition on ADTrees, Freund and Mason [7] note that because alternating trees can be defined as a sum of simple base rules it is a simple matter to apply any boosting algorithm to the problem of inducing ADTrees. For the multiclass setting one possible candidate is AdaBoost.MH. In this paper we also explore and compare two other solutions. The first is to adapt the original two-class ADTree algorithm to the multiclass setting using a variety of wrapper methods. The second is to use the multiclass LogitBoost [10] procedure as the underlying boosting algorithm. This algorithm is a natural choice as it is directly applicable to multiclass problems.

The paper is organized as follows. In Section 2 we review ADTrees and the LogitBoost procedure. Section 3 describes our attempts to cast ADTrees to the multiclass setting. Section 4 describes the new algorithm that induces ADTrees using LogitBoost. Section 5 contains experimental results that compare both the LogitBoost and AdaBoost.MH methods with the best of the adaptations of the original algorithm on some benchmark datasets. Section 6 summarizes the contributions made in this paper.

2 Background

In this section we first summarize the original algorithm for inducing ADTrees. As Freund and Mason [7] argue that any boosting method is applicable to ADTree induction, it is natural to suppose that AdaBoost.MH would provide a good setting for the multiclass extension (given that AdaBoost works so well in the two-class setting). A similar argument can be made for an alternative framework based on LogitBoost, and this is discussed in the final part of this section.

2.1 ADTrees

Alternating decision trees provide a mechanism for combining the weak hypotheses generated during boosting into a single interpretable representation. Keeping faith with the original implementation, we use inequality conditions that compare a single feature with a constant as the weak hypotheses generated during each boosting iteration. In [7] some typographical errors and omissions make the algorithm difficult to implement so we include below a more complete description of our implementation. At each boosting iteration t the algorithm maintains two sets, a set of preconditions and a set of rules, denoted $\mathcal{P}_t$ and $\mathcal{R}_t$, respectively. A further set $\mathcal{C}$ of weak hypotheses is generated at each boosting iteration.

1. Initialize Set the weights $w_{i,t}$ associated with each training instance to 1. Set the first rule $\mathcal{R}_1$ to have a precondition and condition which are both true. Calculate the prediction value for this rule as $a = \frac{1}{2} \ln \frac{W_+(c)}{W_-(c)}$ where $W_+(c)$, $W_-(c)$ are the total weights of the positive and negative instances that satisfy condition c in the training data. The initial value of c is simply True.

2. Pre-adjustment Reweight the training instances using the formula $w_{i,1} = w_{i,0}e^{-ay_t}$ (for two-class problems, the value of y_t is either +1 or -1).

3. Repeat for t = 1, 2, ..., $\mathcal{T}$

(a). Generate the set $\mathcal{C}$ of weak hypotheses using the weights associated with each training instance $w_{i,t}$

(b). For each base precondition $c_1 \in \mathcal{P}_t$ and each condition $c_2 \in \mathcal{C}$ calculate

$$Z_t(c_1, c_2) = 2\left(\sqrt{W_+(c_1 \wedge c_2)W_-(c_1 \wedge c_2)} + \sqrt{W_+(c_1 \wedge \neg c_2)W_-(c_1 \wedge \neg c_2)}\right) + W(\neg c_1)$$

(c). Select c_1, c_2 which minimize $Z_t(c_1, c_2)$ and set $\mathcal{R}_{t+1}$ to be $\mathcal{R}_t$ with the addition of the rule r_t whose precondition is c_1, condition is c_2 and two prediction values are:

$$a = \frac{1}{2} \ln \frac{W_+(c_1 \wedge c_2) + \epsilon}{W_-(c_1 \wedge c_2) + \epsilon}, \qquad b = \frac{1}{2} \ln \frac{W_+(c_1 \wedge \neg c_2) + \epsilon}{W_-(c_1 \wedge \neg c_2) + \epsilon}$$

(d). Set $\mathcal{P}_{t+1}$ to be $\mathcal{P}_t$ with the addition of $c_1 \wedge c_2$ and $c_1 \wedge \neg c_2$.

(e). Update the weights of each training example according to the equation

$$w_{i,t+1} = w_{i,t}e^{-r_t(x_i)y_t}$$

4. Output the classification rule that is the sign of the sum of all the base rules in $R_{\mathcal{T}+1}$:

$$class(x) = sign\left(\sum_{t=1}^{\mathcal{T}} r_t(x)\right)$$

In terms of parameter settings for implementations described in this paper, we set the value of ϵ to 1, and vary the value of $\mathcal{T}$ for stopping the induction in fixed increments (namely, 10, 20, 50 and 100). Determining an optimal setting for $\mathcal{T}$ is still an open research question.

2.2 LogitBoost

As mentioned above, the underlying learning algorithm for ADTrees is AdaBoost. Friedman et al. [10] analyze AdaBoost from a statistical perspective and find that it can be viewed as a stage-wise estimation procedure for fitting an additive logistic regression model according to an exponential loss function. This finding enables them to derive a stage-wise boosting procedure, implementing an adaptive Newton algorithm, that optimizes the (more standard) binomial likelihood instead of the exponential loss function used in AdaBoost. They call this algorithm *LogitBoost*. They also describe a generalized version of LogitBoost that optimizes the multinomial likelihood. This algorithm is directly applicable to multiclass problems. Compared to AdaBoost.MH (see Section 3.2), the general form of LogitBoost (which we call *LT1PC* later) has the advantage that it can be wrapped around any numeric predictor without any modifications. AdaBoost.MH, on the other hand, requires serious modification to the weak learner so that it can produce a separate prediction for each class value and also deal with class specific weights.

3 Multiclass ADTrees

When extending any algorithm from binary to multiclass classification there are two options. The simplest approach is to transform the multiclass problem into several binary classification problems. This general approach can be applied to any classification algorithm, resulting in a set of voting models. Typically, this approach leads to a large number of models. Alternatively, we can attempt to induce a single tree capable of predicting each of the class labels directly.

3.1 Multiclass as multiple two-class problems

Transforming ADTrees to map multiple class labels to two classes can be approached in several ways. As ADTrees can be merged, the resulting multiclass model can be a single tree derived from the set of two-class voting trees.

A standard method [6] is to treat a subset of class labels as class A, and the set of remaining labels as class B, thus reducing the problem to two classes from which a model can be built. This is then repeated for different subsets and the models vote towards the class labels they represent. Provided there is sufficient class representation and separation between the subsets, the vote tallies for individual class labels can be collected to form a reasonable prediction.

We experimented with a number of subset generation schemes:

1-against-1 [9, 1]: generate a tree for every pair of classes, where subset A contains only the first class and subset B contains only the second. An advantage of this approach is that each tree need only be trained with a subset of the data, resulting in faster learning [11].

1-against-rest: one tree per class, where subset A contains the class, and subset B contains the remaining classes.

random: randomly generate a unique subset, creating twice as many trees as there are classes. Random codes have good error-correcting properties [13].
exhaustive: every unique subset possible.

Note that the exhaustive method is not computationally practical as class numbers increase (in our experiments this occurs when there are more than 16 class labels).

3.2 Direct Induction

The AdaBoost.MH algorithm is almost identical to AdaBoost. The major difference is that instead of generating weak hypotheses h_t that map the input space $\mathcal{X}$ to either a discrete set $[-1, +1]$ or by extension $\mathbb{R}$, the weak hypotheses map $\mathcal{X} \times \mathcal{Y}$ to $\mathbb{R}$, where $\mathcal{Y}$ is a finite set of class labels.

It would appear that the correct interpretation of AdaBoost.MH is not immediately obvious, for example, Friedmann et al [10] interpret the method as a variant of *1-against-rest* and build a distinct classifier per class. Many of the criticisms of AdaBoost.MH in [10] are based on this mis-interpretation. Our results suggest that AdaBoost.MH and LogitBoost actually share much in common in terms of both predictive performance and computational complexity.

In fact, AdaBoost.MH constructs a single tree per iteration. To construct an ADTree using AdaBoost.MH we need to change predictor nodes to handle a vector of predictions (one per class) and splitter nodes to compute a Z value per class label. At each iteration the test that minimises the sum of Z scores over all class labels is added to the tree.

To perform prediction using this tree we sum all contributions at each predictor node that is satisfied by the example, to form a prediction vector containing a single prediction per class. We choose the maximum value from this vector as the single output class.

4 LADTree Algorithm

We follow Friedmann et al [10] in defining the multiclass context. Namely, that for an instance i and a J class problem, there are J responses y^*_{ij} each taking values in $\{-1, 1\}$. The predicted values, or indicator responses, are represented by the vector $F_j(x)$ which is the sum of the responses of all the ensemble classifiers on instance x over the J classes. The class probability estimate is computed from a generalization of the two-class symmetric logistic transformation to be:

$$p_j(x) = \frac{e^{F_j(x)}}{\sum_{k=1}^{J} e^{F_k(x)}}, \sum_{k=1}^{J} F_k(x) = 0 \tag{1}$$

The LogitBoost algorithm can be fused with the induction of ADTrees in two ways, which will be explained in the following subsections. In the first, more conservative approach called *LT1PC* we grow separate trees for each class in parallel. In the second approach called *LT*, only one tree is grown predicting all class probabilities simultaneously.

4.1 LT1PC: inducing one tree per class

The LADTree learning algorithm applies the logistic boosting algorithm in order to induce an alternating decision tree. As with the original algorithm, a single attribute test is chosen as the splitter node for the tree at each iteration. Stored with each training instance is a working response and weights on a per-class basis. The aim is to fit the working response to the mean value of the instances, in a particular subset, by minimising the least-squares value between them. When choosing tests to add to the tree we look for the maximum gain, that is, the greatest drop in the least squares calculation. Note, in the algorithm below the $f_{mj}(x)$ vector is equivalent to the single prediction weight of a predictor node in the original ADTree algorithm. The algorithm is as follows:

1. Initialize Create a root node with $F_j(x) = 0$ and $P_j(x) = \frac{1}{J} \forall j$

2. Repeat for $m = 1, 2, \ldots, \mathcal{T}$:
(a) Repeat for $j = 1, \ldots, J$:
(i) Compute working responses and weights in the jth class
$z_{ij} = \frac{y^*_{ij} - p_{ij}}{p_{ij}(1-p_{ij})}$ $\qquad w_{ij} = \frac{y^*_{ij} - p_{ij}}{z_{ij}}$
(ii) Add the single test to the tree that best fits $f_{mj}(x)$ by a weighted least-squares fit of z_{ij} to x_i with weights w_{ij}
(b) Add prediction nodes to the tree by setting
$f_{mj}(x) \leftarrow \frac{j-1}{J}(f_{mj}(x) - \frac{1}{J}\sum_{k=1}^{J} f_{mk}(x))$, and
$F_j(x) \leftarrow F_{ij}(x) + f_{mj}(x)$
(c) Update $p_j(x)$ via Equation 1 above

3. Output Output the classifier $argmax_j F_j(x)$

With this algorithm, trees for the different classes are grown in parallel. Once all of the trees have been built, it is then possible to merge them into a final model. If the structure of the trees is such that few tests are common, the merged tree will mostly contain subtrees affecting only one class. The size of the tree cannot outgrow the combined size of the individual trees. The merging operation involves searching for identical tests on the same level of the tree. If such tests exist then the test and its subtrees can be merged into one. The additive nature of the trees means that the prediction values for the same class can be added together when merged.

4.2 LT: directly inducing a single tree

We can make a simple adjustment to this algorithm within Step 2 by moving Step (a)(ii) out to become Step (b). We then obtain a single directly induced tree, as follows:

2. Repeat for $m = 1, 2, \ldots, \mathcal{T}$:

(a) Repeat for $j = 1, \ldots, J$:

(i) Compute working responses and weights in the jth class

$$z_{ij} = \frac{y^*_{ij} - p_{ij}}{p_{ij}(1-p_{ij})} \qquad w_{ij} = \frac{y^*_{ij} - p_{ij}}{z_{ij}}$$

(b) Add the single test to the tree that best fits $f_{mj}(x)$ by a weighted least-squares fit of z_{ij} to x_i with weights w_{ij}

(c) Add prediction nodes to the tree by setting

$f_{mj}(x) \leftarrow \frac{j-1}{J}(f_{mj}(x) - \frac{1}{J}\sum_{k=1}^{J} f_{mk}(x))$, and

$F_j(x) \leftarrow F_{ij}(x) + f_{mj}(x)$

(d) Update $p_j(x)$ via Equation 1 above

The major difference to LT1PC is that in LT we attempt to simultaneously minimise the weighted mean squared error across all classes when finding the best weak hypothesis for the model.

5 Experimental Results

The datasets and their properties are listed in Table 1. The first set of ten datasets are used to compare ADTrees with LT as an algorithm for solving two-class problems. The remainder are used in multiclass experiments, ordered incrementally from the smallest number of classes (3) to the largest (26). Most of the datasets are from the UCI repository [2], with the exception of *half-letter*. *Half-letter* is a modified version of *letter*, where only half of the class labels (A-M) are present.

In the case of the multiclass datasets, on the first nine having less than eight classes, accuracy estimates were obtained by averaging the results from 10 separate runs of stratified 10-fold cross-validation. In other words, each scheme was applied 100 times to generate an estimate for a particular dataset. For these datasets, we speak of two results as being "significantly different" if the difference is statistically significant at the 5% level according to a paired two-sided *t*-test, each pair of data points consisting of the estimates obtained in one ten-fold cross-validation run for the two learning schemes being compared.

On the datasets with more than eight classes, a single train and test split was used. Statistical significance was measured by the McNemar [5] test.

NA (for not available) in the results table signifies that the learning scheme did not finish training. If learning could not complete within the time period of a week then it was terminated and marked NA. It is not surprising that the exhaustive method did not finish above 16 classes when one considers the number of permutations required. Due to the presence of these unfinished experiments the averages for **all** methods listed in this table exclude the last four datasets. Thus a fair comparison is possible.

Table 2 shows that LT is comparable to ADTree over ten boosting iterations on two-class datasets. There is little change to this result when raising the number of boosting iterations to 100.

Table 1. Datasets and their characteristics

Dataset	Classes	Instances (train/test)	Attributes	Numeric	Nominal
breast-wisc	2	699	9	9	0
cleveland	2	303	13	6	7
credit	2	690	15	6	9
hepatitis	2	155	19	6	13
ionosphere	2	351	34	34	0
labor	2	57	16	8	8
promoters	2	106	57	0	57
sick-euthyroid	2	3163	25	7	18
sonar	2	208	60	60	0
vote	2	435	16	0	16
iris	3	150	4	4	0
balance-scale	3	625	4	4	0
hypothyroid	4	3772	29	7	22
anneal	6	898	38	6	32
zoo	7	101	17	1	16
autos	7	205	25	15	10
glass	7	214	9	9	0
segment	7	2310	19	19	0
ecoli	8	336	7	7	0
led7	10	1000/500	7	0	7
optdigits	10	3823/1797	64	64	0
pendigits	10	7494/3498	16	16	0
vowel	11	582/462	12	10	2
half-letter	13	8000/1940	16	16	0
arrhythmia	16	302/150	279	206	73
soybean	19	307/176	35	0	35
primary-tumor	22	226/113	17	0	17
audiology	24	200/26	69	0	69
letter	26	16000/4000	16	16	0

Table 2. Two-class problems: ADTree vs. LT

dataset	ADTree(10)	LT(10)
breast-wisc	95.61	95.65
cleveland	81.72	80.36 −
credit	84.86	85.04
hepatitis	79.78	77.65
ionosphere	90.49	89.72
labor	84.67	87.5 +
promoters	86.8	87.3
sick-euthyroid	97.71	97.85 +
sonar	76.65	74.12 −
vote	96.5	96.18 −

+, − statistically significant difference

Table 3. Wrapping two-class ADTree results

dataset	1vs1	1vsRest	Random	Exhaustive
iris	95.13	95.33	95.33	95.33
balance-scale	83.94	85.06 +	85.06 +	85.06 +
hypothyroid	99.61	99.63	99.64	99.64
anneal	99.01	98.96	99.05	99.19 +
zoo	90.38	93.45 +	95.05 +	95.94 +
autos	78.48	77.51	77.98	79.99 +
glass	75.90	74.33 −	73.79 −	76.76
segment	96.74	95.94 −	95.91 −	96.62
ecoli	83.31	83.96	84.69 +	85.95 +
led7	75.40	74.40	76.40	75.60
optdigits	92.49	90.26 −	92.21 −	93.82
pendigits	94.11	91.48 −	86.16 −	89.54 −
vowel	47.40	41.13 −	48.48 +	50.65
half-letter	88.71	80.77 −	76.13 −	80.98 −
arrhythmia	68.00	66.00 −	66.00	68.00
soybean	89.36	89.10	89.36	NA
primary-tumor	46.90	43.36 −	46.90	NA
audiology	76.92	80.77	84.62	NA
letter	85.98	70.63 −	65.20 −	NA
average	84.57	83.21	83.46	84.87

+, − statistically significant difference to 1vs1

Given the large number of options for solving multiclass problems using ADTrees we designed the following experiments to provide useful comparisons. First, we determine the best multiclass ADTree method by treating the induction as a two-class problem (Table 3). Second, we compare this method with the AdaBoost.MH and the two LADTree methods described in the last section.

Generally, it is difficult to compare all of these methods fairly in terms of the number of trees produced. For example, the 1-against-1 method produces $\frac{J(J-1)}{2}$ trees, 1-against-rest J, random $2 * J$, and exhaustive 2^{J-1} trees. LT1PC produces J trees while AdaBoost.MH and LT induce a single tree. Thus, it can be the case that the number of trees is greater than the number of boosting iterations, for example, the average number of trees produced by 1-against-1 over all nineteen multiclass datasets is 79. Unless otherwise stated, in all tables we compare methods against a fixed number of boosting iterations (10).

Table 3 allows us to compare the results of each method on an overall basis through the average and on a pair-wise basis through the significance tests. Note that the significance tests are all performed with respect to the first column in the table. On both scales the exhaustive method is the best. As the exhaustive method is not practical for large class datasets we chose the 1-against-1 method to compare against LADTrees, as this method is very similar in overall performance.

Table 4 compares the "winner" of Table 3 (*1-against-1*) to AdaBoost.MH and both versions of LADTrees of various sizes. It demonstrates the improvements that can be made by increasing the number of boosting iterations for the single tree methods LT and AdaBoost.MH as they generate tree sizes closer to the number generated by 1-against-1.

The 1-against-1 method defeats each of the small tree methods at 10 boosting iterations. But when the number of iterations is increased to 100 tests each, we notice a dramatically different picture: all methods are outperforming the 1-against-1 method.

Consider the 100 iteration case: 1-against-1 is boosted 10 times but produces $\frac{J(J-1)}{2}$ trees, which represents an average tree size of 790 (tests). LT and AdaBoost.MH outperform this method on average after 100 iterations (i.e. using trees with 100 tests). Table 4 shows that LT(100) outperforms most of the early datasets (class sizes 3-13) but struggles against two of the later datasets. For *soybean* 1-against-1 uses a tree of size 1710, and for *primary-tumor* it uses a tree of size 2310. Perhaps the most remarkable result is for *half-letter* where 1-against-1 using 780 tests has an accuracy of 88.71% whereas LT(100) achieves 92.16% using only 100 tests.

Clearly, both on an overall average and on a per dataset basis, AdaBoost.MH and LT are comparable methods. There are no obvious performance differences between these methods at 10 and 100 iterations.

Table 4 also compares the two logistic methods. Due to the number of trees used by LT1PC it outperforms LT both on average and on pairwise tests. But these differences seem to disappear as the number of iterations increases: at 10 boosting iterations LT1PC wins on 11 datasets and has 4 losses; at 100 boosting iterations LT1PC has only 4 significant wins and 3 losses.

6 Conclusions

This paper has presented new algorithms for inducing alternating decision trees in the multiclass setting. Treating the multiclass problem as a number of binary classification problems and using the two-class ADTree method produces accurate results from large numbers of trees. Although ADTrees can be merged, the size of the combined tree prohibits its use as a practical method, especially if interpretable models are a requirement.

Using AdaBoost.MH for multiclass problems was thought to be problematic. The theoretical objections to this method presented in [10] appear to be based on a mis-interpretation of AdaBoost.MH. Our experimental results demonstrate that this method is competitive with LogitBoost in the multiclass setting, at least for ADTrees.

Two new algorithms, LT1PC and LT, for inducing ADTrees using LogitBoost are presented. One method induces a single tree per class, the other a single tree, optimised across all classes. In experimental results comparing these methods to *1-against-1*, the best of the wrapper methods, both LADTree methods LT1PC and LT and AdaBoost.MH show significant promise, especially when we consider the relative sizes of the induced trees.

Table 4. LADTree and AdaBoost.MH results

dataset	1PC(10)	LT(10)	MH(10)	1PC(100)	LT(100)	MH(100)
iris	95.07	94.20 −	94.93	95.13	95.13	95.13
balance-scale	88.80 +	84.50	84.21	86.53 +	90.40 +	90.82 +
hypothyroid	99.49 −	99.59	99.57 −	99.55 −	99.62	99.63
anneal	99.44 +	98.50 −	97.41 −	99.62 +	99.66 +	99.72 +
zoo	92.95 +	94.34 +	94.55 +	92.35 +	94.53 +	94.34 +
autos	81.12 +	64.57 −	69.92 −	82.71 +	82.43 +	82.69 +
glass	71.81 −	67.95 −	66.65 −	77.05	75.51	73.97 −
segment	96.68	92.27 −	93.14 −	97.99 +	97.84 +	97.72 +
ecoli	82.44	84.64 +	84.40	84.27 +	83.54	83.99
led7	75.20	77.60	72.80	75.00	73.60	74.00
optdigits	91.32	78.63 −	77.69 −	95.77 +	94.94 +	94.49 +
pendigits	91.65 −	78.53 −	78.24 −	96.74 +	96.51 +	96.00 +
vowel	39.61 −	34.85 −	34.85	48.05	46.54	46.54
half-letter	83.92 −	66.80 −	65.36 −	95.00 +	92.16 +	91.65 +
arrhythmia	70.00	64.67	64.67	68.67	66.67	67.33
soybean	90.43	81.38 −	79.79 −	85.90	83.51 −	92.82 +
primary-tumor	34.51 −	43.36	42.48	33.63 −	42.48 −	45.13
audiology	80.77	80.77	88.46	76.92	76.92	80.77
letter	76.78 −	50.53 −	44.25 −	93.25 +	86.78	84.80 −
average	81.16	75.67	75.44	83.38	83.09	83.77

+, − statistically significant difference to 1vs1

From a different point of view one can also argue that the LADTree and AdaBoost.MH methods are the first direct induction methods for multiclass option trees, a hitherto unsolved problem. Previous attempts [4, 12] were plagued by the need to specify multiple parameters, and also seemed to contradict each other in their conclusion of why and where in a tree options (i.e. alternatives) were beneficial. Contrary to these attempts, the LADTree and AdaBoost.MH methods have only a single parameter, the final tree size, and automatically add options where they seem most beneficial.

A research problem that deserves attention is the determination of the stopping condition $\mathcal{T}$ for boosting methods. Freund and Mason [7] use cross-validation with some success but this method is impractical for large datasets. One possible solution is to use out-of-bag samples to determine if adding new tests will continue to increase performance. This will be a topic of future work.

Acknowledgements

We would like to thank the anonymous referees for making us re-address the results we had earlier achieved with our first implementation of AdaBoost.MH. This uncovered what appears to be a common misunderstanding of how to implement this method.

References

1. Erin Allwein, Robert Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.
2. C. Blake, E. Keogh, and C.J. Merz. UCI repository of machine learning databases. Technical report, University of California, Department of Information and Computer Science, Irvine, CA, 1998. [www.ics.uci.edu/˜mlearn/MLRepository.html].
3. Leo Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.
4. Wray Buntine. Learning classification trees. *Statistics and Computing*, 2:63–73, 1992.
5. Thomas G. Dietterich. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.
6. Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.
7. Yoav Freund and Llew Mason. The alternating decision tree learning algorithm. In *Proc. 16th Int. Conf. on Machine Learning*, pages 124–133. Morgan Kaufmann, 1999.
8. Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th Int. Conf. on Machine Learning*, pages 148–156. Morgan Kaufmann, 1996.
9. Jerome Friedman. Another approach to polychotomous classification. Technical report, Stanford University, Department of Statistics, 1996.
10. Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistic*, 28(2):337–374, 2000.
11. Johannes Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2:721–747, 2002.
12. Ron Kohavi and Clayton Kunz. Option decision trees with majority votes. In *Proc. 14th Int. Conf. on Machine Learning*, pages 161–169. Morgan Kaufmann, 1997.
13. Robert E. Schapire. Using output codes to boost multiclass learning problems. In *Proc. 14th Int. Conf. on Machine Learning*, pages 313–321. Morgan Kaufmann, 1997.
14. Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Proc. 11th Conf. on Computational Learing Theory*, pages 80–91. ACM Press, 1998.