The alternating decision tree learning algorithm

Yoav Freund
AT&T Labs
180 Park Avenue
Florham Park, NJ, 07932, USA

Llew Mason
Department of Systems Engineering
Australian National University
Canberra, ACT, 0200, Australia

Abstract

The application of boosting procedures to decision tree algorithms has been shown to produce very accurate classifiers. These classifiers are in the form of a majority vote over a number of decision trees. Unfortunately, these classifiers are often large, complex and difficult to interpret. This paper describes a new type of classification rule, the *alternating decision tree*, which is a generalization of decision trees, voted decision trees and voted decision stumps. At the same time classifiers of this type are relatively easy to interpret. We present a learning algorithm for alternating decision trees that is based on boosting. Experimental results show it is competitive with boosted decision tree algorithms such as C5.0, and generates rules that are usually smaller in size and thus easier to interpret. In addition these rules yield a natural measure of classification confidence which can be used to improve the accuracy at the cost of abstaining from predicting examples that are hard to classify.

1 INTRODUCTION

The AdaBoost algorithm [7, 16] has recently proved to be an important component in practical learning algorithms. Two of the most successful combinations have been boosting decision trees and boosting stumps [6, 1, 13, 8] . Stumps are the simplest special case of decision trees which consist of a single decision node and two prediction leaves.

Boosting decision trees learning algorithms, such as CART [2] and C4.5 [14], yields very good classifiers. The software package C5.0[1] is one of the best packages that combines (a variant of) AdaBoost with C4.5. Unfortunately, the classifiers generated by C5.0 using boosting are often large and thus hard to interpret.

In this paper we suggest a new combination of decision trees with boosting that generates classification rules that are often smaller and easier to interpret then the rules that are generated by using C5.0 with boosting. We introduce a new representation for classifiers which we call *alternating decision trees* (ADTrees). This representation generalizes both voted-stumps and decision trees in a natural way. We show how boosting can be used as a method for learning ADTrees from data (Kearns and Mansour in [9] analyze decision tree learning algorithms as boosting algorithms. Their work suggests an algorithm similar to the one presented here).

ADTrees are similar to *option trees* first described by Buntine in [3] and further developed by Kohavi et. al. in [10]. Option trees were shown to provide significant improvements in classification error compared to single decision trees. The results reported in [10] are comparable to bagged decision trees. Our goal here is to learn a structure similar to option trees using boosting and thus achieve better performance levels. Our experiments show that the performance of this learning algorithm is similar to that of C5.0 with boosting. The question of improving the comprehensibility of trained classifiers have been previously studied by Craven [4], Domingos [5], and Margineatu and Dietterich [11].

One of the nice features of alternating decision trees is that they give, in addition to a classification, a measure of confidence which we call the classification *margin*.

[1] C5.0 was written by Ross Quinlan and is available commercially from Rulequest Research: `http://www.rulequest.com/`.

This measure of confidence has been analyzed in prior work [15]. In this work we give some new evidence for the validity and usefulness of interpreting classification margins as measures of confidence.

The paper is organized as follows. In Section 2 we describe alternating decision trees and show how they can be represented as weighted votes of simple prediction rules. In Section 3 we describe how AdaBoost is used to learn ADTrees. In section 5 we describe some experiments we have performed with the new algorithm. We conclude with a summary and a description of current work in Section 6.

2 ALTERNATING DECISION TREES

In this section we give a new semantics for representing decision trees and then show how this semantics can be extended to represent general alternating decision tree classifiers. In addition we show how alternating tree classifiers can be represented as a majority vote over very simple prediction rules. Recall that AdaBoost generates rules that *are* majority votes over simpler, so called "weak" rules. By representing alternating trees as such votes we make it easy to use AdaBoost to learn alternating trees from data.

We give a formal description of alternating trees in Figure 2. For our intuitive explanation we use a sequence of simple examples given in Figure 1. Consider first the decision tree in Figure 1(a). This is a simple decision tree with two decision nodes and three prediction leaves. The tree defines a binary classification rule which maps instances of the form $(a, b) \in R^2$ into one of two classes denoted by -1 and $+1$ (in this paper we restrict ourselves to binary classification problems.) In Figure 1(b) we give a different representation of the same classification rule. In this representation each decision node is replaced by two nodes: a *prediction node* (represented by an ellipse), and a *splitter node* (represented by a rectangle). The decision node is identical to what we had before, while the prediction node is associated with a *real valued* number. As in decision trees, an instance is mapped into a *path* along the tree from the root to one of the leaves. However, *unlike* decision trees, the classification that is associated with the path is *not* the label of the leaf, instead, it is the sign of the sum of the predictions along the path. For example, the classification of the instance $a = b = 0.5$ is $\text{sign}(0.5 - 0.7 - 0.2) = \text{sign}(-0.4) = -1$. It is easy to check that the two trees define the same classification rule. It is also clear that many different trees of the second type can represent the same tree of the first type.

We call the second representation an "alternating tree" representation for the obvious reason that it consists of alternating layers of prediction nodes and splitter nodes. So far we have described how standard decision trees can be represented as alternating trees. Before we describe general decision trees we show how alternating trees can be represented as a vote over simple prediction rules.

We think of the tree in Figure 1(b), as consisting of a root prediction node and two units of three nodes each: a decision node and the two prediction nodes that are its children (this structure is very similar to decision stumps, but the numbers associated with the leaves are real number, rather than just $+1$ or -1). It is now a simple transition to rewrite the classification rule described in Figure 1(b) as a weighted majority vote. We associate with each of the decision nodes a simple rule of the following form:

```
if(precondition) then
     if(condition) then output p1
     else output p2
else output 0
```

Specifically, with the decision nodes in Figure 1(b) we associate the following two rules:

```
r1(a,b)=                  |    r2(a,b)=
if(always) then           |    if(a<4.5) then
  if(a<4.5) then          |      if(b>1) then
       output -0.7        |           output +0.4
  else output +0.2        |      else output -0.2
else output 0             |    else output 0
```

By combining these two rules with the constant prediction associated with the root node we can rewrite the classification rule represented by the decision tree as: `sign(0.5 + r1(a,b) +r2(a,b))`. We refer to these rules as *base rules*.

It is clear that through this transformation we can represent any standard decision tree as a sum of base rules each of which corresponds to one of the decision nodes in the tree. In general `precondition` is the conjunction of conditions that lead to a given decision node, `condition` is the decision associated with the node and `p1, p2` are the predictions associated with the two children of the decision node.

We proceed now to show how we generalize standard decision trees to general alternating decision trees that give a much more flexible semantics for representing

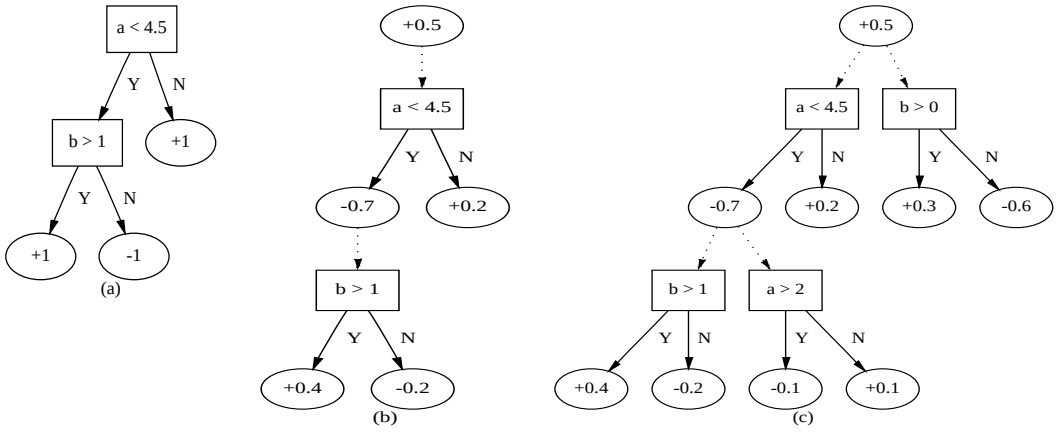

Figure 1: Tree-based classifiers: (a) a decision tree, (b) the same decision tree represented as an alternating tree. (c) a general alternating tree.

1. A *base condition* is a boolean predicate over instances. We use $\wedge$ to denote conjunction (AND), $\neg$ to denote negation (NOT) and $\mathbf{T}$ to denote the constant predicate that is always true. We use $\mathcal{C}$ to denote a set of base conditions.
2. A *precondition* is a conjunction of base conditions and negations of base conditions.
3. A *base rule* r is a mapping from instances to real numbers which is defined in terms of a precondition c_1, a base condition c_2, and two real numbers a and b. The base rule maps each instance to a *prediction* that is defined to be a if $c_1 \wedge c_2$, b if $c_1 \wedge \neg c_2$ and 0 if $\neg c_1$.
4. An *alternating decision tree* is a mapping from instances to real numbers which is defined in terms of a set of base rules. The set of base rules must obey the two following two conditions:
   (a) The set must include a base rule for which both the condition and the pre-condition are $\mathbf{T}$. The a value of this rule is the prediction associated with the root of the tree.
   (b) A base rule r with precondition d can be in the set only if the set includes a rule r' with precondition c_1 and condition c_2 such that $d = c_1 \wedge c_2$ or $d = c_1 \wedge \neg c_2$. d corresponds to the prediction node that is the direct parent of r.

   The alternating tree maps each instance to a real valued *prediction* which is the sum of the predictions of the base rules in its set. The *classification* of an instance is the sign of the prediction.

Figure 2: A formal definition of alternating trees as weighted votes of simple rules

classifiers.[2]

Standard decision trees define a partition of the instance space into disjoint regions. Most algorithms for learning decision trees work by iteratively splitting one of the parts in two. Each part can be split at most once. In other words, only leaf nodes can be split. In *general* alternating decision trees we allow each part to be split multiple times. Returning to our example, we observe that in the alternating tree described in Figure 1(b) each predictor node has at most one splitter node attached to it. In Figure 1(c) we add two splitter nodes to this decision tree and get an example of a general alternating tree.

A general alternating tree defines a classification rule as follows. An instance defines a *set of* paths in the alternating tree. As in standard decision trees, when a path reaches a decision node it continues with the child which corresponds to the outcome of the decision associated with the node. However, when reaching a prediction node, the path continues with *all* of the children of the node. More precisely, the path splits into a *set* of paths, each of which corresponds to one of the children of the prediction node. We call the union of all

[2] Strictly speaking, any ADTree classifier can be represented by a standard decision tree. However, the decision tree can be exponentially larger.

the paths reached in this way for a given instance the *"multi-path"* associated with that instance. The sign of the sum of all the prediction nodes that are included in a multi-path is the classification which the tree associates with the instance. As examples consider the following two instances: if $a = 1$ and $b = 0.5$ then the classification is $\text{sign}(0.5 + 0.3 - 0.7 - 0.2 + 0.1) = \text{sign}(0.1) = +1$, if $a = 5$ and $b = 1$ then the classification is $\text{sign}(0.5 + 0.2 + 0.3) = \text{sign}(1.0) = +1$. In both cases the classification is $+1$, however, we can think of the second prediction as more confident then the second. We'll return to this point later.

From our description of the alternating decision trees it is clear that they are generalization of decision trees. It is not hard to see that they also generalize voted stumps. These correspond to alternating trees with exactly three layers: a root, a set of decision nodes, and the predictor nodes associated with these decision nodes. Finally, voted decision trees are also a special case of alternating decision trees, they correspond to alternating trees in which only the root prediction node has more than one child.

The representation of alternating decision trees as a majority vote over base rules is a direct extension of the representation that was described above for simple decision trees. The formal definition of alternating trees in this form is given in Figure 2.

3 BOOSTING ALTERNATING DECISION TREES

As shown in the previous section, alternating trees can be defined as a sum of simple base rules. As a result, it is a simple matter to apply any boosting algorithm to the problem of learning alternating decision trees from examples. The only special thing here is that the set of base rules (sometimes called "weak hypotheses") that are considered at each stage is not constant but increases as the tree is grown.

As our base rules generate predictions that can be *any* real valued number we use the version of AdaBoost suggested by Schapire and Singer [16] which extends the original AdaBoost algorithm of Freund and Schapire [7] that allows the base rules to make real valued prediction. In addition, Singer and Schapire give formulas for calculating the best predictions for a given partition of the input space. We use these formulas to choose the value of a and b for the base rules.

We fix a set of base conditions. In the experiments described here we use only inequality conditions that compare a single feature with a constant. This set of base rules is sufficiently restricted that it is feasible to enumerate all possible base rules that can be added to a given tree for a given training set.

Initialize Set $\mathcal{R}_1$ to consist of the single base rule whose precondition and condition are both $\mathbf{T}$, and whose first prediction value is

$$a = \frac{1}{2} \ln \frac{W_+(\mathbf{T})}{W_-(\mathbf{T})} .$$

Do for $t = 1, 2, \ldots, T$

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

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

2. 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:

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

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

4. Update the weights of each training example according to the equation

$$w_{i,t+1} = w_{i,t} e^{r_t(\mathbf{x}_i) y_i}$$

(Note that if $r(\mathbf{x}_i) = 0$ the weight is not changed.)

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

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

Figure 3: An application of AdaBoost to learning alternating decision trees.

The learning algorithm is described in Figure 3. We introduce some notation used in the figure.

- The training set is denoted $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ where $\mathbf{x}_i \in R^d$ and $y_i \in \{-1, +1\}$.
- The set of base conditions (inequalities comparing a single feature and a constant) is denoted by $\mathcal{C}$.

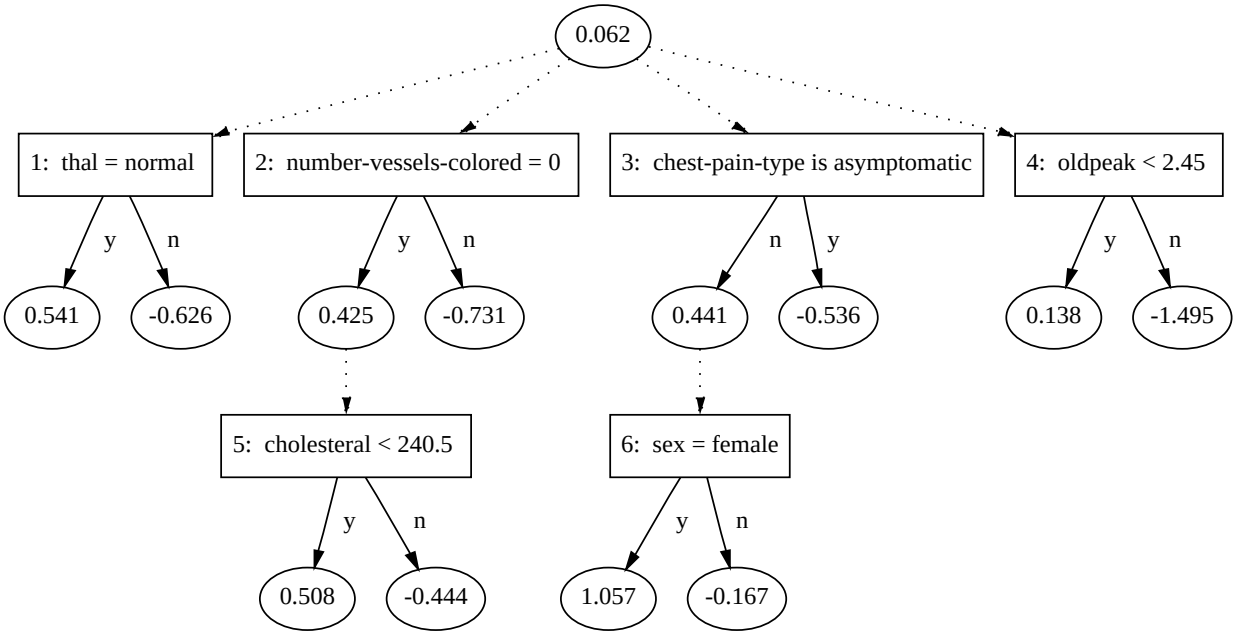

Figure 4: An alternating decision tree learned for the cleve data set (negative numbers correspond to predicted sickness).

A base rule is denoted by r and we use $r(\mathbf{x})$ to denote the real value that the rule associates with the instance $\mathbf{x}$.

- The algorithm maintains two sets, a set of preconditions and a set of rules. The symbols $\mathcal{P}_t$, $\mathcal{R}_t$ correspond to these two sets on boosting iteration t. The initial precondition set is $\mathcal{P}_1 = \{\mathbf{T}\}$.
- The algorithm associates a positive weight with each training example, we denote by $w_{i,t}$ the weight of example number i on boosting iteration t the initial weights are $w_{i,0} = 1$ for all examples $1 \leq i \leq m$.
- The notation $W(c)$ represents the total weight of the training examples which satisfy the predicate c. Similarly, we use $W_+(c), W_-(c)$ to denote the total weight of those examples that satisfy c and are labeled $+1$ or -1 respectively (i.e. $W(c) = W_+(c) + W_-(c)$).

The algorithm starts by finding the best constant prediction for the entire data set. This prediction is placed at the root of the tree. It then grows the tree iteratively, adding one base rule at a time. The added base rule corresponds to a subtree with a decision node as its root and two prediction nodes as the leaves. This subtree is added as a child of a predictor node which might or might not be a leaf node.

In this description we have not specified how to choose when to *stop* the boosting process. In other words, how to choose T. In fact, it is still unclear what is the best way for choosing T. In the experiments described in this paper we used cross-validation to make this choice.

The algorithm described here has some similarities to well studied learning algorithms such as C4.5 and CART. Similarly to those algorithms it proceeds in a greedy way, adding one decision rule at a time by finding the best split to expand the current tree. However, there are several important differences, among them:

- Decision nodes can be added at any location in the tree, not just the leaves.
- The splitting criterion is different, it is the weighted error of the added rule, rather than the GINI index (CART) or the information gain (C4.5).

4 INTERPRETING ALTERNATING DECISION TREES

In this section we argue for the interpretability and robustness of the alternating tree representation.

To demonstrate our interpretation, we consider the alternating tree presented in Figure 4. This tree is the result of running our learning algorithm for six iterations on the `cleve` data set from Irvine. This is a data set of heart-disease diagnostics for which the goal is to discriminate between sick and healthy people[3] In our mapping positive classification correspond to healthy and negative to sick.

This alternating tree, which consists of six decision nodes, achieves an average test error of 17%. Compare this to the classification rule that is generated by boosting C5.0 (see Table 1 at the end of the paper) which achieves a average test error of 20%. This last rule consists, on the average, of 446 decision nodes! We claim that the fact that the description of the alternating tree classifier is so much smaller makes it easier to read and to interpret. Of course, the structure of the alternating tree is more complex than that of a decision tree. However, as we shall now argue, interpreting an alternating tree is not harder, and may be even easier, than interpreting a decision tree with the same number of decision nodes.

Our main argument for the interpretability of alternating trees rests on the fact that the contribution of each decision node can be understood in isolation. Summing these contributions generates the prediction and the classification. Consider the example in Figure 4. Considering in isolation each of the decision nodes we observe that a cholesterol level higher than 240.5 and an asymptomatic chest pain are both predictors of heart problems. This is indicated by the fact that they both generate negative contributions to the prediction sum. We can easily interpret the meaning of all the decision nodes in the tree in a similar way.

After gleaning the meaning of each decision node in isolation we can analyze the interactions of the nodes. Parallel decision nodes, such as the four nodes in the first level, represent little or no interaction. In other words, the fact that the `thal` test is normal increases the likelihood that the person is healthy irrespective of the `number of vessels colored` or the type of chest pain. This is the type of association of indicators that is represented by voted decision stumps or by naive Bayes classifiers. Note that in order to represent the same information in a standard decision tree we would need a decision tree with 4 levels, each level to test for one of the indicators. Such a tree would have $2^4 - 1 = 15$ decision nodes instead of four. Indeed, the decision tree that C5.0 generates for this data set is large and has a high error rate.

In contrast to the independence of the decision nodes in the first level, the significance of the two decision nodes in the second level depends on the evaluation of their ancestral decision nodes. Specifically, regarding the node `sex = female` the fact that a patient is a male appears to be more predictive of a heart problem when their chest pain is symptomatic then in the population in general. This implies that only when the chest pain is asymptomatic it is worthwhile to check the patients gender. The decision-tree structure can represent dependences between indicators, which cannot be represented by voted stumps.

The root of the tree is associated with the fixed (unconditional) contribution of 0.062. This small positive number indicates that (according to this training set) there are slightly more healthy people then sick. This means that before testing any feature value we should predict "healthy" (but with low confidence).

All the contributions are summed in order to give the final prediction, and this prediction is thresholded to give the classification. This means that if we test the conditions given in the tree serially, we accumulate evidence for or against the health of the person as we proceed. If at some intermediate point during this process we have a sum whose absolute value is large, and the total contribution of all of the (untested) prediction nodes is small, [4] then we don't need to continue our computation - the current sign of the sum cannot change. Even if it is theoretically possible for the sign to change, it might require that most of the future tests will give contribution in the direction opposite to the sign of the current sum, which is unlikely. We thus think about the absolute value of the sum as a measure of confidence of the classification, and call it the classification *margin*. This interpretation is further justified by the theoretical analysis of margins in Schapire et. al. [15]. In Section 5 we show some empirical evidence that strengthens this interpretation and suggests how it can be quantified and used.

[3] provided, through David Aha, by the V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

[4] This situation is less likely to occur in our small example in which all of the contributions, other than that of the root, are large. However, it is a common occurrence in larger trees.

The fact that each decision node has a limited influence on the prediction adds to the robustness of this representation. In some situations we cannot evaluate some of the decision nodes. This might be because some feature values are unknown (a common occurrence in the UCI dataset) or because we are pressed for time. In such situations we can do well by considering only the reachable decision nodes whose associated predictions are large. This will degrade the accuracy only slightly because the influence of the ignored nodes is usually not sufficient to change the classification. The indices printed in the left side of the decision nodes in Figure 4 indicate the boosting iteration on which the nodes were added. In general, lower indices correspond to more influential nodes that were added earlier in the boosting process.

5 EXPERIMENTS AND RESULTS

In order to examine the practical performance of the alternating decision tree algorithm described in the previous section we ran experiments on a collection of nineteen data sets taken from the UCI machine learning repository [12]. To simplify matters, we restricted ourselves to binary classification problems only.

The three learning algorithms that we compared were:

- C5.0 (with and without boosting),
- StumpBoost: our boosting algorithm restricted to generate trees with a single layer of decision nodes.
- ADTree: our boosting algorithm run without restrictions.

We tested C5.0 by averaging over 10 runs of 10 fold cross-validation.

On the smaller data sets ADTree quickly overfits the data, as can be seen in the `cleve data` set in Figure 5. For this reason we had to carefully choose when to stop the boosting process. To choose this number of iterations we did 10-fold cross validation within the training set in each of our experiments. We chose as the stopping time the iteration in which the validation error was minimized. We then reran the algorithm on the whole training set and stopped it at the chosen iteration to generate the classifier that we tested on the held-out data set. Because of this time-consuming method for choosing the stopping iteration we tested our algorithm using 3 runs of 10-fold cross validation.

Figure 5 shows two typical learning curves for ADTree and StumpBoost. For the data set `cleve` both algorithms reach their best performance after less then ten boosting iterations. After more iterations the training error continues to decrease but the test error increases. To overcome this overfitting behavior, early stopping of the boosting algorithm is critical.The `kr-vs-kp` data set is much larger, which is probably the reason that there is almost no difference between the training and test errors. Here we see that ADTree reaches a very small error after about 50 iterations while the error of StumpBoost remains large even after 200 iterations. This is a case in which the larger capacity of ADTree gives it an advantage.

Table 1 summarizes the results from all the data sets. For each data set the number of examples and features is listed along with the average size of the generated classifier and average test error (with standard error) for each classifier tested. Note that *C5.0* refers to C5.0 without boosting (i.e. a single decision tree) and *C5.0 + Boost* refers to C5.0 with 10 rounds of boosting. For the C5.0 results, the size represents the average of the total number of decision nodes, summed over the ten trees. For StumpBoost and ADTree, the size is the average number of boosting rounds which is equal to the number of decision nodes.

For the sake of comparison, we summarize our results in two scatter plots (Figure 6.) We see that the performance of C5.0+boost is very close to the performance of our algorithm. With a slight advantage to C5.0+boost. We see that in some of the smaller data sets StumpBoost outperforms both C5.0 and ADTree, the reason for that probably being that it is less prone to overfitting.

Comparing the size of the classifiers we find that, in all but three cases, the classifiers generated by alternating tree algorithm are much smaller then those generated by C5.0 with boosting.

One interesting feature of these experimental results is that, in general, when C5.0 performed significantly better than StumpBoost, so did ADTree and when StumpBoost performed significantly better than C5.0, so did ADTree. This demonstrates the ability of alternating decision trees to represent both voted sums and decision trees.

Surprisingly, in some of the data sets the best method is C5.0 without boosting. This method has the lowest test error in `house-votes-84`,`hypo1`, `hypothyroid`, `sick-euthyroid` and `dis` and the tree size is small. We have no explanation for this behavior.

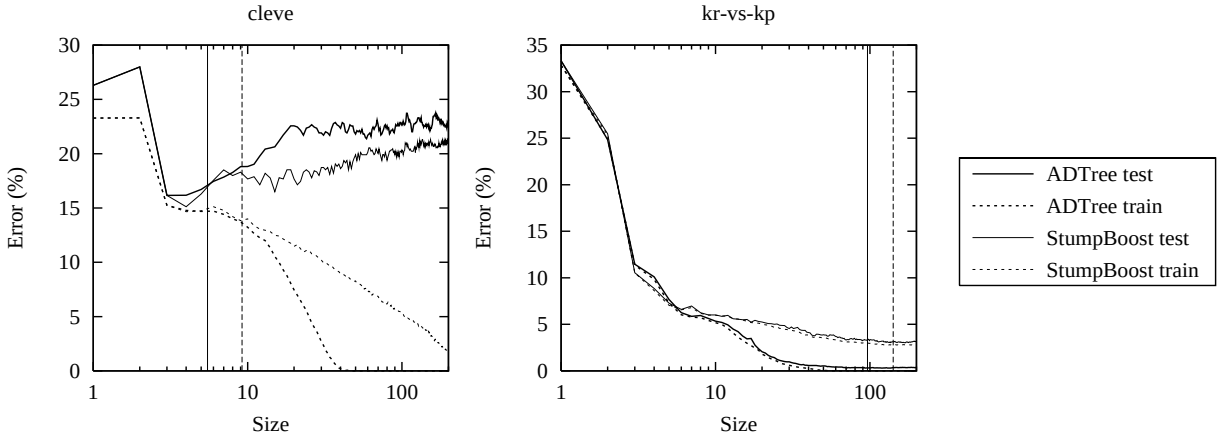

Figure 5: Averaged train and test error curves for ADTree and StumpBoost on the `cleve` and `kr-vs-kp` data sets. The average stopping points for ADTree and StumpBoost are marked with solid vertical lines and dashed vertical lines respectively.

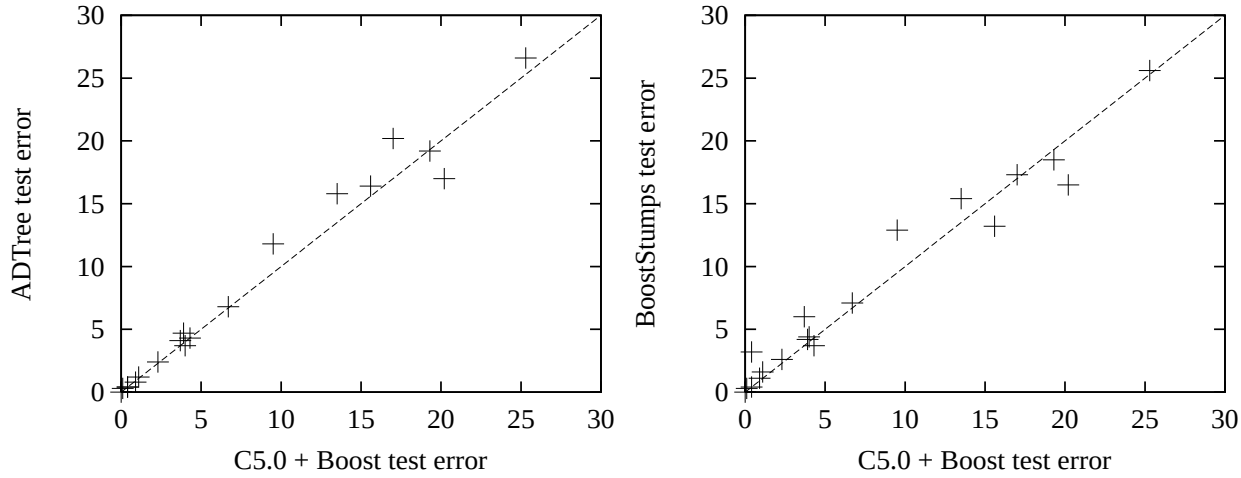

Figure 6: Average test error of C5.0 + Boost versus ADTree (left) and C5.0 + Boost versus StumpBoost (right) on all of the examined data sets.

We explored several statistics in an attempt to quantify the relation between the absolute value of the prediction and the probability that the classification is correct. From our preliminary studies, we found a graphical representation for measuring and possibly using this relationship. Two Examples are given in Figure 7. Each graph represents the conditional probability that the label is positive (which we denote by p) given the value of the prediction (denoted s). As expected, p is close to 1 when $s > 0$ and is close to 0 when $s < 0$ (otherwise, by definition, the error would be large). However, note that when s is far from zero, the conditional probability is even closer to zero and to 1 respectively. This means that when s is far from zero (i.e. the classification margin is large) we can classify the instance much more reliably then when s is close to zero (small classification margin). This gives experimental justification to interpreting the classification margin as a measure of confidence.

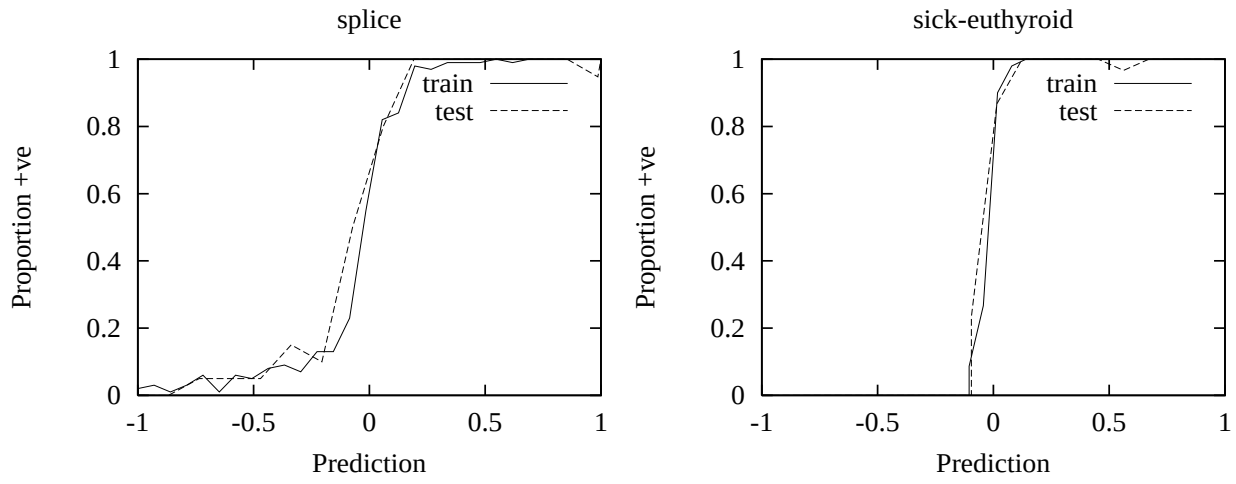

Figure 7: Calibration graphs for the `splice` and `sick-euthyroid` data sets for train and test after 100 rounds of ADTree.

In addition to justifying our interpretation, the calibration graphs can potentially be used to improve our performance situation where our classification rule is allowed to abstain from predicting on some fraction of the test examples. For the `splice` data set after 100 rounds of boosting we can choose to abstain when $-0.2 \leq s \leq 0.2$, which would reduce the test error from 6.3% to 1% at the cost of abstaining on 22.5% of the data (for one particular train/test split). Similarly for the `sick-euthyroid` data set abstaining when $-0.063 \leq s \leq 0.063$ reduces the test error from 3.2% to 1.7% at the cost of abstaining on 10% of the data.

6 CONCLUSIONS

We have described a new representation for classification rules that is interpretable and robust. We have shown how boosting can be used to learn this type of rules. The new learning algorithm combines, in a simple fashion, decision trees and boosting. In our experiments we show that the error performance of the new algorithm is close to that of C5.0 with boosting. Lastly, we gave some preliminary experimental justification to the interpretation of the absolute real-valued prediction that is generated by the alternating decision tree as a measure of classification confidence.

In the near future we plan to further research the use of calibration graphs to decide when to abstain from prediction. We plan to integrate a new boosting algorithm which is more robust against noise than AdaBoost (to be described in a separate paper). Lastly, we plan to explore ways for speeding up the algorithm, especially for large data sets.

References

[1] Leo Breiman. Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California at Berkeley, 1996.

[2] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth & Brooks, 1984.

[3] Wray Buntine. Learning classification trees. *Statistics and Computing*, 2:63–73, 1992.

[4] Mark W. Craven. *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, University of Wisconsin-Madison, 1996. Also appears as UW Technical Report CS-TR-96-1326.

[5] Pedro Domingos. Knowledge acquisition from examples via multiple models. In *ml97*, pages 98–106, 1997.

[6] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

[7] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[8] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. Technical Report, 1998.

		C5.0	C5.0 + Boost	Stump Boost	ADTree	C5.0	C5.0 + Boost	Stump Boost	ADTree
Data Set	Examples & Features	Size	Size	Size	Size	Test Error	Test Error	Test Error	Test Error
labor	57 : 16	4.3	61.0	27.6	43.3	19.9 ± 0.8	15.6 ± 0.8	13.2 ± 1.0	16.4 ± 3.4
promoters	106 : 57	16.6	134.6	133.8	309.5	22.4 ± 1.1	9.5 ± 0.8	12.9 ± 1.8	11.8 ± 2.1
hepatitis	155 : 20	9.8	197.8	47.9	42.5	21.0 ± 0.4	17.0 ± 0.4	17.3 ± 0.3	20.2 ± 0.6
sonar	208 : 60	14.3	197.7	463.1	338.1	25.6 ± 0.8	19.3 ± 0.7	18.5 ± 2.2	19.2 ± 0.6
cleve	303 : 13	27.2	446.8	9.2	5.5	27.2 ± 0.5	20.2 ± 0.5	16.5 ± 0.8	17.0 ± 0.6
ionosphere	351 : 34	13.8	216.8	51.3	299.3	10.3 ± 0.3	6.7 ± 0.2	7.1 ± 0.6	6.8 ± 0.3
house-votes-84	435 : 16	5.8	292.2	9.1	10.2	3.3 ± 0.1	4.0 ± 0.1	4.4 ± 0.3	3.7 ± 0.2
vote1	435 : 16	10.9	360.8	19.6	9.1	4.9 ± 0.2	4.3 ± 0.1	3.7 ± 0.3	4.3 ± 0.2
credit	690 : 15	18.3	770.2	31.1	160.3	14.3 ± 0.3	13.5 ± 0.2	15.4 ± 0.5	15.8 ± 0.0
breast-cancer	699 : 9	12.4	346.3	48.3	135.7	5.4 ± 0.1	3.9 ± 0.1	4.2 ± 0.7	4.7 ± 0.5
pima-indians	768 : 8	22.4	821.8	56.8	14.5	25.5 ± 0.2	25.3 ± 0.3	25.6 ± 0.2	26.6 ± 0.2
kkn	1000 : 16	7.0	50.1	10.1	5.3	0.1 ± 0.0	0.1 ± 0.0	0.3 ± 0.1	0.3 ± 0.2
hypo1	2514 : 29	6.9	235.4	33.0	23.0	0.2 ± 0.0	0.4 ± 0.0	0.4 ± 0.0	0.4 ± 0.0
hypothyroid	3163 : 25	6.6	567.9	58.7	6.5	0.7 ± 0.0	0.9 ± 0.0	1.1 ± 0.0	0.8 ± 0.0
sick-euthyroid	3163 : 25	13.3	1153.0	34.2	49.7	2.0 ± 0.0	2.3 ± 0.0	2.6 ± 0.1	2.4 ± 0.1
splice	3190 : 60	109.5	2637.0	94.8	318.8	4.2 ± 0.0	3.7 ± 0.0	6.0 ± 0.1	4.1 ± 0.1
kr-vs-kp	3198 : 36	29.2	849.4	141.4	96.3	0.6 ± 0.0	0.4 ± 0.0	3.2 ± 0.0	0.4 ± 0.1
dis	3772 : 29	14.2	629.4	129.0	118.4	1.0 ± 0.0	1.1 ± 0.0	1.6 ± 0.0	1.2 ± 0.1
agaricus-lepiota	8124 : 22	22.0	51.7	42.4	14.8	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

Table 1: Summary of results for a collection of binary classification problems from the UCI database.

[9] Michael Kearns and Yishay Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, 1996.

[10] Ron Kohavi and Clayton Kunz. Option decision trees with majority votes. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 161–169, 1997.

[11] Dragos D. Margineantu and Thomas G. Dietterich. Pruning adaptive boosting. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 211–218, 1997.

[12] C. J. Merz and P. M. Murphy. UCI repository of machine learning databases, 1998. www.ics.uci.edu/∼mlearn/MLRepository.html.

[13] J. R. Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730, 1996.

[14] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[15] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, October 1998.

[16] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 80–91, 1998. To appear, *Machine Learning*.