Decision Tree Analysis using Weka

Machine Learning – Project II

Sam Drazin and Matt Montag

University of Miami

***Abstract* - This paper discusses applications of the Weka interface, which can be used for testing data sets using a variety of open source Machine Learning algorithms. The data sets were tested using the J48 decision tree-inducing algorithm (Weka implementation of C4.5), which was published by Ross Quinlan in 1993.**

I. INTRODUCTION

THERE exist a number of prominent Machine Learning algorithms used in modern computing applications. These algorithms have been engineered over the last several decades, and many are open-source and available for public usage and modification. A common application for these algorithms often involves decision-based classification and adaptive learning over a training set. The decision tree is a popular utility for implementing such tactics. A *decision tree* is a decision-modeling tool that graphically displays the classification process of a given input for given output class labels. This paper will discuss the algorithmic induction of decision trees, and how varying methods for optimizing the tree, or *pruning* tactics, affect the classification accuracy of a testing set of data.

Weka is an open-source Java application produced by the University of Waikato in New Zealand. This software bundle features an interface through which many of the aforementioned algorithms (including decision trees) can be utilized on preformatted data sets. Using this interface, several test-domains were experimented with to gain insight on the effectiveness of different methods of pruning an algorithmically induced decision tree.

Pruning decision trees is a fundamental step in optimizing the computational efficiency as well as classification accuracy of such a model. Applying pruning methods to a tree usually results in reducing the size of the tree (or the number of nodes) to avoid unnecessary complexity, and to avoid over-fitting of the data set when classifying new data. For example, when classifying an example in a decision tree and reaching a certain node, there are two possible outcomes: positive and negative. Imagine that running a testing set through this node shows that 95% of the examples are evaluated as positive. The assumption is made that this *test* node can be replaced entirely with a positive classifier. Depending on the percentage of total error caused by pruning, however, the noisy data incurred may not be deemed negligible depending on the tolerance from the data set (and/or a sufficiently dominant consistency of classification by the test node). Such a replacement is illustrated in Figure 1.

A smaller sub-tree can be inserted instead of replacing a test node with a class label to prune a decision tree as well. In reference to Figure 1, this would involve replacing a node such as 't2' with a smaller test node, like 't5'. In either case, the resulting tree is smaller and ideally more efficient than the pre-pruned tree. The process of pruning traditionally begins from the bottom of the tree (at the child leaves), and propagates upwards.

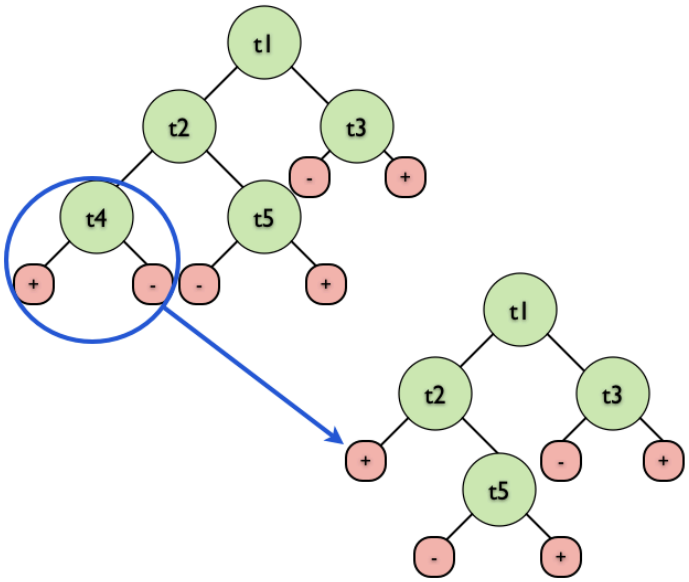

Figure 1: A decision tree's test node (t4) is replaced in entirety with a positive classification (assuming a binary class domain).

II. PRUNING METHODS

Two pruning methods were examined in this study: the *post-pruning* method, and the *online pruning* method. Each method has strengths and weaknesses, yet their largest differences concern the timing of implementation, and the parameters they consider for node removal. In either case, a decision tree must be in the process of being induced, or completely created for pruning to occur (depending on the method). The overlying principle of pruning is to compare the amount of error that a decision tree would suffer before and after each possible prune, and to then decide accordingly to maximally avoid error. The metric used to describe possible error, denoted *error estimate* (or E), is calculated with the

formula shown in Figure 2.

$$E = \frac{e+1}{N+m}$$

Figure 2: Error estimate formula, where *'e':* misclassified examples at the given node, *'N':* examples that reach the given node, and *'m':* all training examples.

Post-pruning is implemented on a fully induced decision tree, and sifts through to remove statistically insignificant nodes. Working from the bottom up, the probability (or relative frequency) of sibling leaf nodes will be compared, and any overwhelming dominance of a certain leaf node will result a pruning of that node in one of several ways. The error estimate of each child node is calculated and used to derive the total error of the parent node. The parent node is then pruned according to the relative frequencies of the child nodes, and this replacement node's error is compared with that of the old parent node (which was influenced by the child nodes' error). This comparison will dictate whether or not pruning is advantageous at a given node.

Online-pruning differs from post-pruning in that it operates on the decision tree while is being induced. To create the decision tree, contemporary algorithms divide the data set on the attributes that provide the most information gain about the class label; this dividing factor serves as the *deciding* attribute for each test node. Whenever a split is made which yields a child leaf that represents less than a minimum number of examples from the data set, the parent node and its children are compressed into a single node. This process continues throughout the creation of the entire tree. Figure 3 exemplifies how at a given split (marked at 't4'), the minimum number of data points needed to represent a class label is not met. Weakness of the positive class (in this example) leads to the negative classifier replacing the parent node 't4'; this takes place as the tree is populated algorithmically.

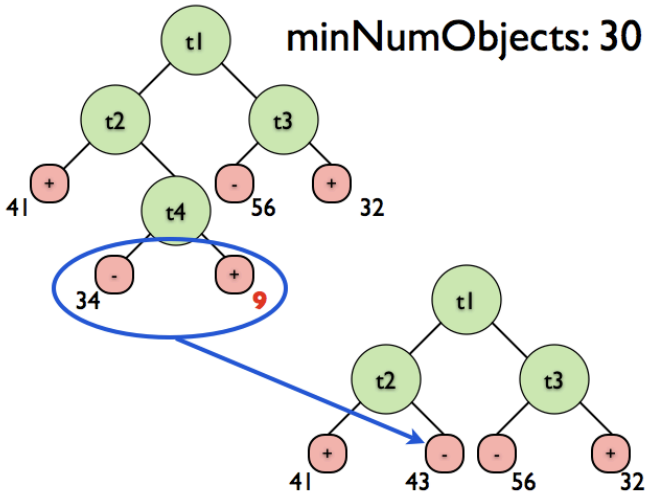

Figure 3: Replacing a test node that meets the requirements for online-pruning due to a leaf with less than the *minNumObjects* classified items.

III. Analysis/Results

Based on predefined thresholds and limits specified by various parameters, the tolerance for test node replacement proved to be a worthy candidate for experimentation. Weka provides a number of sample domains, each of which varying in the number of instances, classes, and attributes. We used the *soybean.arff* data set in our experiments. This data set contains 36 attributes over 683 instances having 19 possible class labels.

To assess the performance of both pruning methods on the sample domains, a series of trials with both training and testing sets were sent through a decision tree created by the J48 algorithm, and the classification accuracy recorded.

A. Post-pruning

The parameter altered to test the effectiveness of post-pruning was labeled by Weka as the *confidence factor*. In the Weka J48 classifier, lowering the confidence factor decreases the amount of post-pruning. In order to understand how this works, some discussion of the confidence factor is necessary.

Post-pruning in the C4.5 algorithm is the process of evaluating the decision error (estimated percent misclassifications) at each decision junction and propagating this error up the tree. At each junction, the algorithm compares (1) the weighted error of each child node versus (2) the misclassification error if the child nodes were deleted and the decision node were assigned the class label of the majority class. The training data misclassifications at each node would not provide a sufficient error estimate - the tree was created from this data so it would not lead to any pruning. Instead, the misclassification error must be understood as an approximation of the actual error based on incomplete data. This is where the statistical notion of confidence comes into play. We take a pessimistic guess of the actual classification error based on the training data. If a given node is labeled "positive" and contains 1 positive and 1 negative (incorrectly classified) instance, its error is 50%. The actual error, however, for classifying future instances at this node could be anywhere between 28% and 72% (with a confidence factor of 0.25). Since we are pessimistic about our classifier, we assign the high error of 72%. If we have less confidence in our training data (**which corresponds to a lower "confidence factor"**), the error estimate for each node goes up, increasing the likelihood that it will be pruned away in favor of a more stable node upstream. Nodes reached by very few instances from the training data are penalized, since we cannot make confident assumptions about their actual classification error.

We tested the J48 classifier with confidence factor ranging from 0.1 to 1.0 by an increment of 0.1, as well as auxiliary values approaching zero. The number of minimum instances per node (minNumObj) was held at 2, and cross validation folds for the Testing Set (crossValidationFolds) was held at 10 during confidence factor testing.

As shown in Figure 4, performance of the classifier on the testing set increased as the confidence factor increased up to about 0.5 at a peak of 92.8% accuracy, after which the

classifier exhibited effects of over-training. These effects are seen by a decrease in classification accuracy with a confidence factor above 0.5. Figure 6 shows an example of Weka's visualization of a decision tree regarding the seed-size attribute from the *soybean.arff* file.

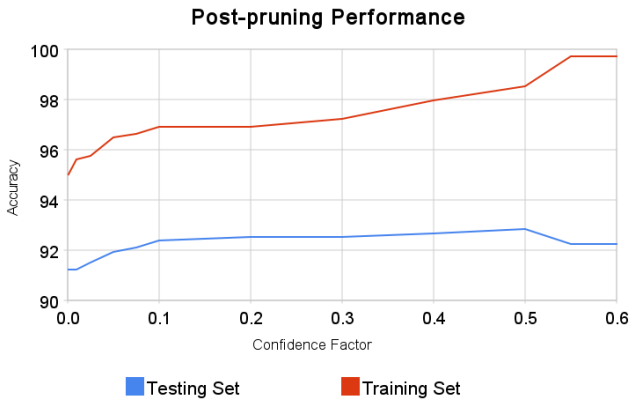

Figure 4: Classification accuracy after post-pruning on a given test domain was processed. Training and testing sets are shown.

B. *On-line pruning*

For online pruning tests, we tested minimum instance (minNumObj) values ranging from 1 to 40. The confidence factor was held constant at 1.0 to minimize post-pruning (it is important to note this did not completely disable post-pruning). Cross validation folds for the testing set (crossValidationFolds) was held at 10. We found that increasing the minimum instance requirement reduced the accuracy of the classifier. There was no over-training error at low values. We suspect this was because baseline post-pruning reduced noise in the training set and eliminated this effect.

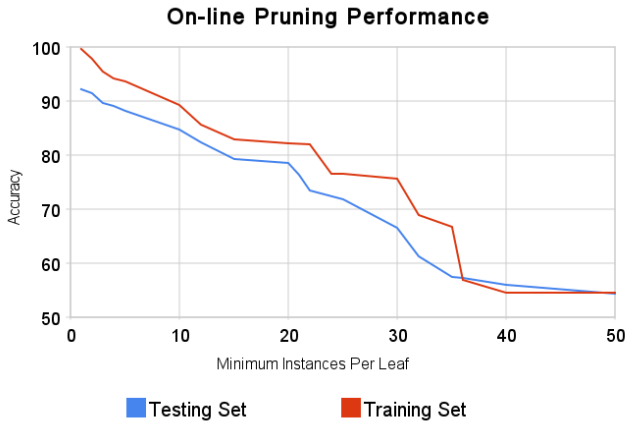

Figure 5: Classification accuracy after online-pruning on a given test domain was processed. Training and testing sets are shown.

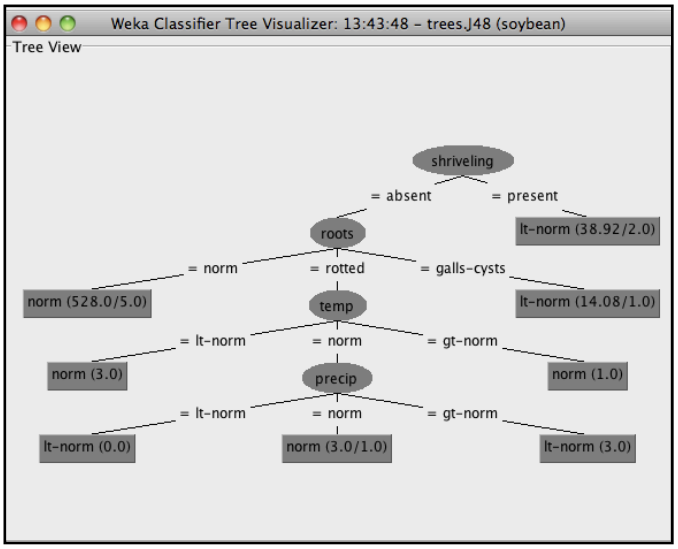

Figure 6: A tree visualization from one of the sample domains in Weka.

IV. Conclusion

Proper utilization of pruning methods and techniques has shown to increase classification accuracy given an induced decision tree.

In our testing, we found that online pruning is useful for reducing the size of the decision tree, but always imparts a penalty on accuracy. On the other hand, lowering the confidence in the training data (confidenceFactor) can not only reduce the tree size, but also helps in filtering out statistically irrelevant nodes that would otherwise lead to classification errors. We conclude that several values for the confidence factor should be tested when generating decision trees to find the most appropriate value for the particular training set under examination.