Janara Christensen
AI mini-project

Implementation Summary

For my mini-project I decided to implement a decision tree learner and then compare it to the decision trees provided in Weka. For my test sets and training sets I used the spam data on the UCI repository http://archive.ics.uci.edu/ml/datasets/Spambase. I think this is the same dataset suggested on the ‘Assignment 3’ website referred to by the mini-project website. It has 57 continuous attributes and a class label (1 or 0).

My decision tree learner hill-climbs through the space of decision trees using the information gain formula discussed in Russell and Norvig on page 659 and returns the resulting decision tree. I also provided methods for classifying a specified test set on the decision tree, performing *n* fold cross validation, and finding the accuracy of the decision tree given a specified test set.

My decision tree learner is built to handle specific types of data. All data must be formatted just like the test sets I have included in this file. Furthermore, all features must be numeric and the classes must be ‘1’ and ‘0’. I have not implemented any support for multi-class or non-numeric learning. Because I wanted to deal with numeric data, it was much trickier to find appropriate splits. I decided to follow the advice in Russell and Norvig on page 664 and make the feature splits binary by finding the best split point for each feature. This method proved to be extremely slow so I then implemented the suggestion on the ‘Assignment 3’ website for speeding it up. I did make one change to the method suggested in my implementation. Instead of checking only points where the classification changed, I check all points. The reason for this decision was that there appeared to be a lot of repeated values of attributes in the spam dataset. The problem with repeated values is that instead of just keeping track of the last classification, you have to make sure the last point examined wasn’t a repeated value, and, if it was, you have to check all of the classifications for that same value instead of just the last classification. I was concerned that this would be more expensive than just doing an entropy calculation for each individual attribute value. Most likely, the advantage of either method just depends on the data. If almost all the points are repeat points, my method is faster, but if they’re all different, then the way suggested on the website is faster. In any case, this new method was about six times faster than my original method.

My decision tree learner performs cross validation but does not do any sort of ensemble learning. My learner also does not split a node if the split will not decrease the entropy at all. Essentially this is a much simpler and less extensive version of chi-squared pruning. I decided not to implement any additional pruning for my decision tree learner. Rather, I thought it would be interesting to see how my relatively unpruned decision tree learner compared to the decision tree learners in Weka.

To run my program enter: `python decisiontree.py filename #folds`
so for dataset2.txt and 5 folds you would type: `python decisiontree.py dataset2.txt 5`
(I have included two datasets; dataset1.txt is much larger than dataset2.txt. The .txt files are for the python program, and the .arff files are for Weka.)

Experiments Summary

First I ran my classifier on dataset1.txt (which is the original dataset from the UCI repository). I trained it on the entire dataset and then tested it on the entire dataset. Next I did the same thing for a few of the decision trees on Weka. These are the results from that test:

Algorithm	Accuracy	Algorithm	Accuracy
Janara	99%	J48	97%
REPTree	94%	RandomForest	99%
ADTree	92%	RandomTree	99%

Next I tried five fold cross validation on each of the above algorithms on the same dataset. I did five fold cross validation because I was concerned that, given the size of the dataset, it would simply take too long. Here are the results from that test:

Algorithm	Accuracy	Algorithm	Accuracy
Janara	92%	J48	92%
REPTree	91%	RandomForest	94%
ADTree	92%	RandomTree	88%

Clearly from these results, when trained and tested on the same set, my learner does very well and the other algorithms (for the most part) do about the same or only slightly worse. When we apply cross validation, my algorithm does about the same as the others, which perform about as well as they did before.

I was very surprised that my learner performed about as well as REPTree, ADTree, and J48 on the cross validation test. REPTree and J48 both apply pruning and ADTree is careful not to overfit by adding too many nodes to the tree. These trees, however, were smaller (especially ADTree), making them much easier to understand. And because decision trees are often noted for their understandability, it’s clearly better to produce the simplest trees possible that perform well.

I was not as surprised that RandomForest and RandomTree appeared to be overfitting the data. RandomForest produces many trees and then uses an ensemble method to combine the trees but does not do any pruning, and RandomTree considers a certain number of randomly chosen attributes at each node but also does not do any pruning.

I was very impressed by the speed that some of the algorithms demonstrated. In particular, REPTree, which is described as a fast decision tree learner, was *very* fast. In the description provided by Weka, it notes that it only sorts the data once. In my implementation, I was sorting the data for each attribute every time it considered splitting the data.