Decision Tree Lab Report

1

This section of the lab was primarily focused on implementing the decision tree machine learning algorithm described in class. Our particular approach to this technique called for information gain as the discriminator between attributes when deciding how to split. This metric encourages the tree to split when the split does a good job of separating output classes from one another. Of the labs we have done so far, this one has been the easiest for me to implement. For the most part, I was able to just crank out the code and have it work correctly right off the bat.

One area I struggled with was trying to get my drawn tree to match the provided tree for the debug data. Even after I was getting all the right gain amounts, I was still stuck with my splits not being correct. After a long while of working through the problem, setting up print statements, and mulling over my code, I realized that my tree was correct, it was just printing the branches in a different order. Rather than printing the 'astigmatism = no' branch first, mine was printing the 'astigmatism = yes' branch first. I had been so caught up in the splits that I had not looked at the actual values. After I figured this out, I was able to continue forward on applying my new model to the other parts of the problem.

In previous labs, my code has not been particularly robust. When using a new model, I would constantly have to play around with different parameters (like the provided counts) to try and get my model to accept the new training data. This time around, I made a special effort to ensure my model was resilient to more forms of data. Rather than rely on any external parameters, I wanted my code to calculate these itself such that it could work without having to manually pre-process the dataset. While this approach meant coding took a bit longer, I was very grateful for the extra work when I was able to copy/paste the same code with a different file name to get an extra dataset working.

Once all the code was written and I figured out my printing bug, I was able to match the expected results for the debug training set. With the training set all done, I moved on to the evaluation set. My Overall accuracy for this set was 14.7%, which is not particularly good. My Gains at each split (truncated after 2 digits) are shown below.

```
Info gain:  [1.36, 0.72, 0.82, 0.88, 0.69, 0.98, 0.68, 0.86, 0.72]
```

2

Section 2.1 of the lab focused on implementing 10-fold cross validation. The first step was to figure out the exact algorithm for N-fold cross validation. N-fold cross validation is a process used to estimate the generalization accuracy of a model. The first step is to split your training data into N equal size groups. Then, for each of these groups, train a model on each group except the current group. Take the freshly trained model, and test its accuracy on the current group (the group not used for training). Track the accuracies of each group and report the average.

Using this process, you are able to say with reasonable certainty how good at generalization your model is. Each instance in the dataset is used for both training and testing at different times. By using N different test sets and averaging their accuracy, N-fold cross validation avoids the situation where your test set happens to be significantly 'easier' or 'harder' than real novel instances. In models where overfit is a problem, this approach also allows you to estimate the amount of training that should be done to fit the problem well. The algorithm was not especially tricky and essentially boiled down to splitting the data, then joining certain sections as training data.

In section 2.2, I was tasked with applying my cross validation to a real dataset, specifically cars.arff. My first issue with this task was trying to load the file using the arff.loadarff function. There was a comment on this section about potentially using a different arff loader to accomplish the same task, but I was determined to use the loader I was comfortable with.

After a bit of debugging, I was able to diagnose the issue with the cars.arff file and fix it. The issue was the presence of an extra space between the final value of a given attribute and the closing brace. This caused the parser to believe the final values of attributes all had trailing spaces. When these trailing spaces were not found in the data from the grid, an error would be thrown. Once I had fixed this issue locally, I also opened a pull request to the CS472 github repository. I hope this fix gets approved to stop future students from having to learn a whole new dataframe for one single question.

After fixing the file's formatting, I was able to super easily plug the X and Y values into my fresh crossValidation function and get some reasonable accuracy predictions. According to the lab specification, typical accuracies for the car data are about .90-.95. While the cross validation introduces a lot more randomness in the final accuracy result, I found my numbers hover right around that range. This is table of the cars 10-fold cross validation

```
Training Acc:
-  -  -  -  -  -  -  -  -  -
1  1  1  1  1  1  1  1  1  1

Test Acc:
----  ----  ----  ----  ----  ----  ----  ----  ----  ----
0.98  0.93  0.92  0.92  0.93  0.94  0.97  0.92  0.96  0.94

Average Test Accuracy  0.9406976744186046
```

In section 2.3, I did the exact same thing as I did in section 2.2, but the voting data instead of cars. Here, my code's robustness paid great dividends. I was able to simply copy/paste my code from the previous cell and change the file name. With this one change made, everything worked flawlessly right away. My ranges for test accuracy seemed to be at or above the typical accuracy. This is my table for 10-fold cross validation:

```
Training Acc:
-  -  -  -  -  -  -  -  -  -
1  1  1  1  1  1  1  1  1  1

Test Acc:
----  ----  ----  ----  ----  ----  ----  ----  ----  ----
0.95  0.91  0.98  0.93  1.00  0.93  0.95  0.95  0.91  0.91
```

I am very satisfied with the accuracy I observed in all the test runs. For both the datasets, my accuracy was in the expected range. One thing I consistently saw across all runs was the 1 for all of the training accuracies, though this is not uncommon for decision tree models. This happens because a decision tree is able to split on every attribute value. Therefore, as long as no two instances share all the same inputs and have different outputs, a fully expanded decision tree can reach 100% training set accuracy. The only other way a decision tree can fail to achieve full accuracy is when overfit avoidance mechanisms like pruning or minimum gain threshold are introduced. Without these measures in place, a decision tree will just keep splitting until it has either a) reached a pure node or b) split on every attribute.

3

For section 3, I am attempting to explain what exactly my models have learned from the training data. One advantage of a decision tree over something like an MLP is interpretability. Interpretability means a human can inspect a trained decision tree and start to understand which features the tree thought were important. This is done by examining which features were split on early and looking at what predictions are more frequent in each of the features sub-trees. One issue is that interpretability becomes harder and harder as a tree keeps getting larger and more complex. To improve my ability to interpret the tree, I used a validation set and reduced-error pruning. As you will see in section 6, my approach was able to decrease the number of nodes by at least a factor of three, but sometimes by a factor of ten. This means there is a lot less noise to look through when trying to understand a tree.

The first split in the cars dataset was consistently done on the safety attribute. The model discovered that all cars with low safety received an 'unacceptable' rating. Cars which received a 'high' safety rating would commonly fall in the 'good' or 'vgood' categories while cars with a 'med' safety rating were more frequently in the 'acc' or 'good' classes. This insight meant that safety was a consistent discriminator for the different output classes. The next attribute split on was persons. From what I could see in the tree, it looks like all 2 person cars are rated as unacceptable, which makes this a good attribute to split on. Other values of 'persons' did not seem to be particularly different in terms of the output class. I believe the reason this attribute was not split on first was because of the relatively small number of 2 person cars compared to the larger number of low safety cars. The third split was typically on 'buying' which I was surprised to see was generally inversely correlated with acceptability. I would have thought more expensive cars would have been viewed as more acceptable, but it seems the tree discovered that to not be the case.

For the voting dataset, my pruning made a huge difference in the interpretability. Without pruning, my tree would hover around 55 nodes which is very hard to sift through. After pruning, my tree was able to get that number down to just 4 nodes. It seems that splitting on just physician-fee-freeze was enough to get really good accuracy. My tree discovered that '?' and 'N' inputs were most likely democrats, and 'Y ' inputs were most likely republicans.

Part of dealing with real data is having values which are unknown. There are a few different good ways of handling these missing values. The approach I took was to allow 'missing' to be just another potential attribute. This allows the model to find potential relationships among the non-voters which can be helpful for some problems. Additionally, this approach does not force you to interpolate new data which can often be inaccurate or misleading for a model. If I had instead replaced each unknown with the mode of that attribute, I may be further pushing my model into the 'baseline accuracy' method where you simply output the most frequent class. Finally, this approach was incredibly simple to implement because it let me just feed in the raw data which used '?' as an unknown.

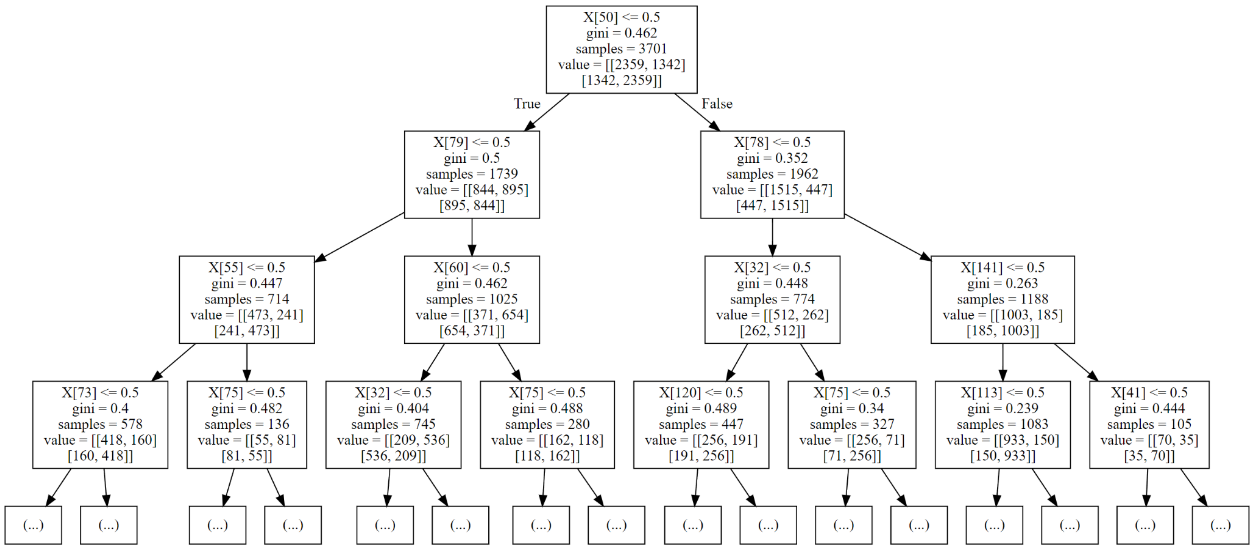
4

For section 4, I used sklearn's decision tree to solve the same voting task from earlier. The most difficult part of this was to figure out the encoding that sklearn would accept. By default, data in the voting set comes in as a 2D array of strings, which is not usable by sklearn. To get sklearn to accept the data, I had to build a custom 'oneHottify' function which would take in the Xs and Ys then convert them to their appropriate one-hot encoding. Once I had written and verified my one hot function, I was able to simply feed the training data into the model and test. Overall, the accuracy of the sklearn model was indistinguishable to my accuracy from earlier in the lab.

One hyperparameter I tried was max_depth. This parameter prevents the tree from growing to be more than a specific number of nodes deep. As my pruning approach found earlier, the voting problem can be solved very well with a single split. When I set max_depth to 1, the accuracy was not significantly impacted for the test set. I also tried playing around with different 'criterion' and 'splitter' values, but I was unable to determine if either of these values significantly affected accuracy.

In section 4.2, I decided to use the supermarket dataset. This is an entirely nominal dataset which seeks to predict how much a customer will spend based on their shopping history. This dataset has a large number of binary input parameters which makes it very difficult for a human to understand the relationships. However, the decision tree model does not care about high dimensional data so was able to consistently get around 70% accuracy. For comparison, I tried using the sklearn DummyClassifier to find the baseline accuracy which hovered around 65%. Across runs, my model consistently beat the baseline accuracy by around 5-10%. While this is not a huge leap, it does show that some learning was done to improve accuracy to at least some extent. When I set the decision tree's max_depth to be 1, I noticed a shop drop in improvement over the baseline. With this change, my accuracy was only around ~1% better than the baseline. I once again played around with various different other parameters such as splitter and criterion, but neither of these other parameters made a significant difference.

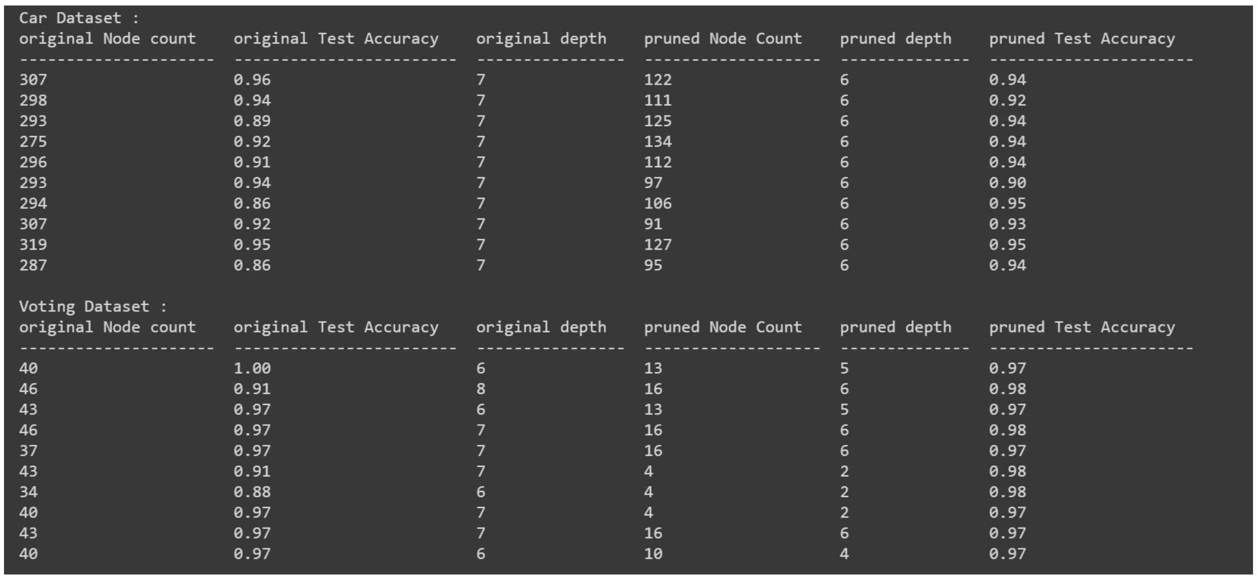
5



This section was very easy to do, I just copy/pasted a code snippet from the sklearn documentation. It looks like the x[50] or 'cleaners-polishers' was the first split, but I cannot say with high confidence what this tree is actually doing. Each subtree is quite large and the width of the tree grows exponentially. This dataset has over 200 attributes which was doubled when used with one-hot encoding, meaning this tree is exceptionally unintelligible.

6

For this section, I was tasked with implementing a reduced error pruning algorithm. My algorithm works as follows. 1) Visit each node in a post-order fashion. 2) if a given node is a leaf, stop. 3)Test the accuracy of the validation set. 4) Mark the node as a leaf for the majority class of its instances. 5) Re-test the accuracy of the validation set. 6) if the new validation set accuracy is worse, unmark the node as a leaf. With this approach, I got no significant change in accuracy, but I was able to significantly reduce the node count and therefore potential overfit of the car and voting models. These are my raw results:



```
Car Dataset :
original Node count    original Test Accuracy    original depth    pruned Node Count    pruned depth    pruned Test Accuracy
--------------------   -----------------------   ---------------   ------------------   -------------   ---------------------
307                    0.96                      7                 122                  6               0.94
298                    0.94                      7                 111                  6               0.92
293                    0.89                      7                 125                  6               0.94
275                    0.92                      7                 134                  6               0.94
296                    0.91                      7                 112                  6               0.94
293                    0.94                      7                 97                   6               0.90
294                    0.86                      7                 106                  6               0.95
307                    0.92                      7                 91                   6               0.93
319                    0.95                      7                 127                  6               0.95
287                    0.86                      7                 95                   6               0.94

Voting Dataset :
original Node count    original Test Accuracy    original depth    pruned Node Count    pruned depth    pruned Test Accuracy
--------------------   -----------------------   ---------------   ------------------   -------------   ---------------------
40                     1.00                      6                 13                   5               0.97
46                     0.91                      8                 16                   6               0.98
43                     0.97                      6                 13                   5               0.97
46                     0.97                      7                 16                   6               0.98
37                     0.97                      7                 16                   6               0.97
43                     0.91                      7                 4                    2               0.98
34                     0.88                      6                 4                    2               0.98
40                     0.97                      7                 4                    2               0.97
43                     0.97                      7                 16                   6               0.97
40                     0.97                      6                 10                   4               0.97
```