**Cars/Voting Data Set**

You would get one hundred percent accuracy if you were able to perfectly split classes until they are “pure” or in other words you were able to perfectly separate them all into all of the other attributes before you ran out of categories when splitting the data. If you ran out of attributes to split on and still had non-pure leaf nodes, then it would run into scenarios of not having 100 percent accuracy because you would have non-pure leaf nodes.

These tables are formatted with attempts on the left going down and specific part of partition from right to left.

**Cars Table:**

**Partitions with Respective Accuracy**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **3** | **5** | **6** | **7** | **8** | **9** | **10** | **total** |
| **1 Train** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **0.97** | **1.00** | **1.00** | **0.98** | **0.99** |
| **1 Test** | **0.895** | **0.913** | **0.921** | **0.902** | **0.930** | **0.913** | **0.894** | **0.913** | **0.892** | **0.904** | **0.912** |
| **2 Train** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** |
| **2 Test** | **0.945** | **0.932** | **0.913** | **0.946** | **0.927** | **0.933** | **0.938** | **0.959** | **0.895** | **0.926** | **0.931** |
| **3 Train** | **0.992** | **1.00** | **1.00** | **0.995** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **0.985** | **0.996** |
| **3 Test** | **0.953** | **0.966** | **0.972** | **0.941** | **0.935** | **0.957** | **0.939** | **0.923** | **0.900** | **0.914** | **0.940** |
| **4 Train** | **1.00** | **1.00** | **0.995** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **0.99** |
| **4 Test** | **0.955** | **0.962** | **0.961** | **0.951** | **0.973** | **0.942** | **0.954** | **0.958** | **0.953** | **0.954** | **0.959** |
| **5 Train** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** |
| **5 Test** | **0.940** | **0.953** | **0.966** | **0.934** | **0.954** | **0.965** | **0.906** | **0.965** | **0.941** | **0.922** | **0.943** |

**Discuss these results:**

My cars data set definitely did well on the training data that it received and almost always seemed to get 100 percent accuracy on the training data set accuracy. After viewing the tree that it would make this definitely made sense has most of the time it would be fully expanded into pure leaf nodes. It unfortunately decreased in accuracy however when tested on non-training data and some pruning back should help increase that accuracy a little bit more on that data. However, for the most part it seems pretty accurate. Further below in the “what they learned” section I talk more specifically about each individual characteristic.

**Voting Table:**

**Partitions with Respective Accuracy**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **3** | **5** | **6** | **7** | **8** | **9** | **10** | **AVG** |
| **1 Train** | **0.941** | **1.00** | **0.972** | **1.00** | **0.962** | **1.00** | **0.948** | **0.961** | **1.00** | **0.922** | **0.972** |
| **1 Test** | **0.890** | **0.910** | **0.923** | **0.902** | **0.931** | **0.914** | **0.893** | **0.911** | **0.892** | **0.904** | **0.912** |
| **2 Train** | **0.974** | **1.0** | **0.963** | **1.0** | **0.962** | **1.0** | **1.0** | **0.961** | **1.0** | **0.943** | **0.974** |
| **2 Test** | **0.943** | **0.922** | **0.943** | **0.931** | **0.924** | **0.943** | **0.911** | **0.934** | **0.943** | **0.938** | **0.939** |
| **3 Train** | **1.0** | **1.0** | **1.0** | **0.974** | **1.0** | **0.972** | **1.0** | **0.961** | **0.993** | **0.941** | **0.983** |
| **3 Test** | **0.903** | **0.956** | **0.973** | **0.881** | **0.953** | **0.951** | **0.932** | **0.923** | **0.937** | **0.944** | **0.940** |
| **4 Train** | **0.98** | **1.0** | **0.99** | **0.95** | **1.0** | **0.98** | **1.0** | **0.96** | **1.0** | **0.99** | **0.98** |
| **4 Test** | **0.932** | **0.946** | **0.961** | **0.950** | **0.919** | **0.943** | **0.955** | **0.921** | **0.960** | **0.933** | **0.942** |
| **5 Train** | **0.965** | **0.981** | **0.953** | **0.988** | **0.994** | **0.966** | **0.991** | **1.0** | **1.0** | **1.0** | **0.981** |
| **5 Test** | **0.933** | **0.954** | **0.883** | **0.902** | **0.933** | **0.984** | **0.933** | **0.962** | **0.938** | **0.973** | **0.958** |

**Discuss these results:**

Again, similar to the cars data set, it seemed to do well on the training data, although not quite as well as the cars data set did. When looking at the tree it also seemed to mostly branch out completely when reaching 100% but again it was less frequent at happening compared to the cars data set. It also wasn’t as accurate as the cars data set on testing data. I believe that this was because of the missing data that I had to fill in and might have skewed the tree a little bit and make the trees slightly less accurate then if I was given the correct data. But nevertheless, the tree was mostly accurate and again I believe that pruning should help increase the accuracy later on.

**What They Learned**

**Cars**

It seems that the car data would always split on the “safety” first. This would make sense as safety would be the most important attribute in determining if a car is worth buying as no one would want to buy a dangerous car. Attributes doors and lug boot would usually be the last attributes to split (aka the ones towards the bottom of the tree). These would be of rather low priority to consider when buying a car compared to the other more important attributes. The attributes persons and maintenance would be usually broken up in the middle section of the tree. Those attributes would definitely play a large role on if a car should be bought or not.

**Voting**

It seemed that the voting data set really like to first split on the attribute “adoption-of-the-budget-resolution”. This out of all the attributes I could see easily being one of the most partisan in nature and would definitely polarize into democrat and republican respectively depending on which party is in charge of the budget. “Immigration” and “water-project-cost-sharing” also seemed to be issues that would be higher up in the tree and, while I’m not exactly shore what water-project-cost-sharing is, I can see it being a polarizing issue if it’s a government social program, along with immigration being a hot button issue. Towards the bottom of the tree were attributes such as “Duty free exports” and “handicapped infants”. These would make sense to be at the bottom of the tree as these are issues that pretty much everyone agrees on so wouldn’t be or at least SHOULDN’T be divisive issues and thus wouldn’t distinguish a republican from a democrat.

**Unknown Attributes**

I handled attributes by comparing the unknown piece of data to the rest of the data set that was identical in every aspect with the exception of the missing data. The complete data in that subset with the highest occurrence of that specific missing data in that set would be the data that would be assigned to that missing data. If there was a tie then I would assign it randomly one of attributes in the tie. I chose this approach because in my mind the majority rules and if we have other complete data to compare with, that ought to be a good indicator of what type of data the “missing” data should be.

**Reduced Error Pruning**

**Cars**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Pruned Tree Test Set Accuracy | Pruned Tree Nodes | Depth Pruned | Regular Tree Test Set Accuracy | Regular Tree Nodes | Depth  Regular |
| Attempt 1 | 0.973 | 40 | 5 | 0.912 | 210 | 6 |
| Attempt 2 | 0.952 | 22 | 4 | 0.933 | 197 | 6 |
| Attempt 3 | 0.969 | 48 | 5 | 0.945 | 225 | 7 |
| Attempt 4 | 0.933 | 24 | 4 | 0.956 | 185 | 6 |
| Attempt 5 | 0.960 | 50 | 5 | 0.921 | 240 | 8 |
| Average | 0.957 | 39.8 | 5 | 0.933 | 150 | 6 |

**Voting**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Pruned Tree Test Set Accuracy | Pruned Tree Nodes | Pruned  Depth | Regular Tree Test Set Accuracy | Regular Tree Nodes | Depth  regular |
| Attempt 1 | 0.934 | 24 | 4 | 0.912 | 52 | 6 |
| Attempt 2 | 0.970 | 34 | 5 | 0.905 | 58 | 5 |
| Attempt 3 | 0.961 | 29 | 5 | 0.946 | 46 | 5 |
| Attempt 4 | 0.978 | 19 | 4 | 0.932 | 48 | 5 |
| Attempt 5 | 0.974 | 12 | 4 | 0.929 | 38 | 5 |
| Average | 0.961 | 23.6 | 4 | 0.925 | 49.4 | 6 |

As you can see above it does seem that the pruning does help increase the accuracy a little in both the voting and the cars data sets. This thus helps account for the overfitting problem that our decision trees can have happen to them where they learn the training data too well. This also makes the trees smaller as well which in larger trees with many nodes would help improve run performance as well. Obviously in this lab the trees were small enough that it didn’t matter that much but in general it would. The depth obviously was smaller on the pruned trees most of the time.

**Creative Part**

For the creative part I decided to experiment with the methods of deciding to fill in the missing data in the voting data set. I made special effort to not use this changed data in the testing data set as I did this experiment. I decided to test (along with my average accuracy method that I used for the project), what would happen if I just filled it with yes, just with no, A random yes or no, the average, treat it as a spate class type to learn, the mode and the mean.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Just Yes Accuracy | Just No Accuracy | Random Accuracy | Average Accuracy | Separate Class Accuracy | Mode Accuracy | Median Accuracy |
| Attempt 1 | 0.876 | 0.893 | 0.831 | 0.934 | 0.946 | 0.917 | 0.931 |
| Attempt 2 | 0.891 | 0.893 | 0.947 | 0.959 | 0.961 | 0.936 | 0.945 |
| Attempt 3 | 0.904 | 0.923 | 0.884 | 0.967 | 0.935 | 0.912 | 0.945 |
| Attempt 4 | 0.910 | 0.894 | 0.963 | 0.973 | 0.954 | 0.919 | 0.956 |
| Attempt 5 | 0.851 | 0.922 | 0.859 | 0.923 | 0.938 | 0.934 | 0.932 |

As you can see it seems to be not as effective with the first three columns(just yes just no or random). I would expect these three to be less effective as their effects don’t really attempt to solve or fill in the missing data with any degree of accuracy. Interestingly it seems like the average, mode, median and separate class seem to get around the same results for this data set. It seems that for this data set they would generally most of the time fill in the same values. It would be interesting in the future to really test these different methods on different data sets with varying degrees of missing data. The voting data set seemed to be mostly filled in and didn’t have too much missing data from what I could tell. It would be interesting to see if these methods varied considerably if there was less data given or more data given respectively.