Decision Tree Lab

Drake Foltz

# Task 1: Implement Decision Tree

To start implementing the algorithm I decided to use the homework assignment. The main problem that I ran into with this method was that I didn’t know the shape of the final tree. However, I am confident that I have implemented the algorithm correctly. It seems like it works, because lenses gives me an 67% accuracy, and then when using all-lenses as the predict data I got the expected accuracy of 33%.

# Task 2: Cars and Voting

I could easily see a tree getting 100% accuracy on its training data, but I fail to see the reasoning behind why one would run predict on the data on which they just trained. The only reason that I could see to do that would be to ensure that it really did learn that part of the data. The reason why it would get 100% accuracy is that already did the partitioning on that dataset, and the dataset has not changed. The only exceptions to this would be randomization of the dataset (not a huge effect), the placement of unknown values, and tie-breaking.

The results for Cars and Voting are as follows:

| Cars | Run 0 | Run 1 | Run 2 | Run 3 | Run 4 | Run 5 | Run 6 | Run 7 | Run 8 | Run 9 | Avg. |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test-Fold | 0.8308 | 0.8615 | 0.8154 | 0.8154 | 0.7615 | 0.7692 | 0.7984 | 0.8450 | 0.8295 | 0.8295 | 0.8156 |
| Train | 0.8165 | 0.8130 | 0.8182 | 0.8182 | 0.8242 | 0.8233 | 0.8201 | 0.8149 | 0.8141 | 0.8166 | 0.8179 |

| Voting | Run 0 | Run 1 | Run 2 | Run 3 | Run 4 | Run 5 | Run 6 | Run 7 | Run 8 | Run 9 | Avg. |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test-Fold | 0.9091 | 0.9091 | 0.9394 | 0.8788 | 0.9697 | 0.9697 | 0.9688 | 0.9375 | 1.0000 | 0.9688 | 0.9451 |
| Train | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.9966 | 0.9966 | 0.9966 | 0.9966 | 0.9986 |

At first it was a bit worrying to see Cars getting a lower accuracy than the “sanity check,” but according to the slack discussion, that may be because I am not shuffling the data. Otherwise around 80% is acceptable.

## Task 2a: What did the tree learn

* 1. In looking at the tree generated from my implementation, the first split is along “Physician-fee-freeze” with “Yes”, “No”, and then the unknown value. Afterwards, it seems that for all budget-related features the republicans voted one way and democrats the other. Most of the people who voted along with the democrats on “Phyician-fee-freeze” are democrat, except in the leaf case of “handicapped infants.” Meanwhile the other side is still heavily democrat as well. That would follow with democrat being the most common class overall.

Cars is a smaller tree, with “safety” being the most important feature overall. And then “Persons” or “Buying” are the next important features. Afterwards it is primarily “Lug Boot” or trunk capacity. Once past this point, it gets very scattered.

## Task 2b: Handling the unknown

I had thought at first that I had it taken care of, but I soon learned my mistake. What made it interesting is how NaN does not equal NaN. After some searching on StackOverflow, I came to conclusion to use that as a feature in finding unknown values and then I would use some other default value for other calculations. I selected a string for easy recognition in labeling the output.

# Task 3: Scikit-learn version

It was pretty easy to set this up and use the same type of code. The only complicated part was that it neither wanted NaN nor my special string for the unknown values. My solution to this was to look at the number of features, number of classes, and the different inputs. In the end, I chose the “magic number” of 37.0 to represent unknown values because there were less than that many features and no attributes used it already.

## Task 3a: Cars and Voting

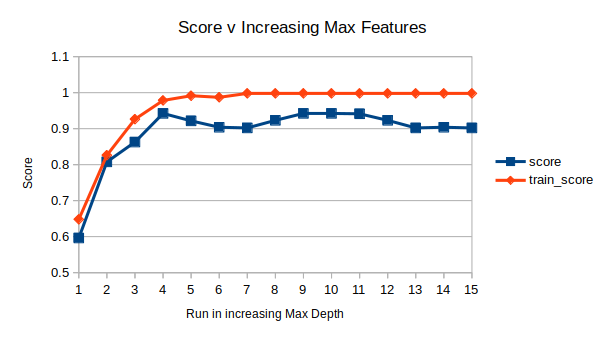
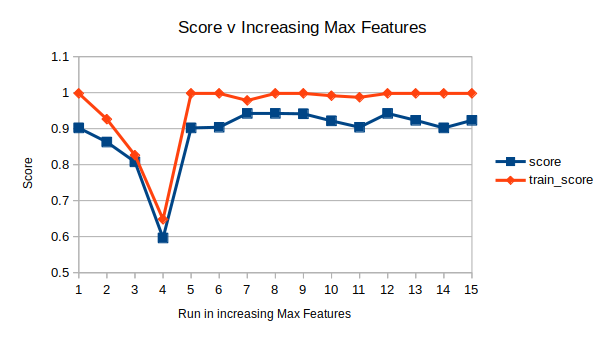
| skCars | Run 0 | Run 1 | Run 2 | Run 3 | Run 4 | Run 5 | Run 6 | Run 7 | Run 8 | Run 9 | Avg. |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test-Fold | 0.9615 | 0.9462 | 0.9538 | 0.9615 | 0.9692 | 0.9846 | 0.9767 | 0.9535 | 0.9767 | 0.9690 | 0.9653 |
| Train | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |

It’s pretty clear here that my implementation is a bit off of theirs, again, it is probably the lack of shuffling. But I can clearly see how it is getting 100% accuracy for the training data, which makes sense.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| skCars | Run 0 | Run 1 | Run 2 | Run 3 | Run 4 | Run 5 | Run 6 | Run 7 | Run 8 | Run 9 | Avg. |
| Test-Fold | 0.9394 | 0.9091 | 0.8788 | 0.9394 | 0.9697 | 1.0000 | 0.9063 | 0.9688 | 0.9375 | 0.9688 | 0.9418 |
| Train | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |

This one matches my implementation more closely. I hit 100% on one of my runs, like this one. Also the averages are similar. The main difference is that I did not get 100% accuracy for all of my Training data.

## Task 3b: Soybeans

* 1. Perhaps my idea of “play with the hyperparameters” is a bit different, but I chose to randomly sample ranges for two of the hyperparameters. I used the range from 1 to the number of features for the “max\_depth” parameter, with the assumption that a “square” tree would be some sort of maximum. And then for “max\_features” I chose a random float between 0.1 and 1. I did this 16 times, running the tree with the same validation set loop as previously. That means I have 160 samples on the “soybeans” dataset that I found.
  2. This first graph is perhaps the easiest to understand. In this, the data is sorted according to increasing max depth. It’s pretty clear to see that as depth increases there is a growth in accuracy up until a certain point. This is around where max\_depth is at 14, or a little over a third of the number of features.
  3.  The next graph is a bit harder to understand, partially as a consequence of the random sampling. However, it appears as though increasing the ratio of features used had little effect on the score, as there is not much visual correlation as max\_features increases.

## Task 4: Tree

I have set this tree to be 5 deep, however, it is in a png, and will not expand right. It looks to be well-organized, and have lots more data than what I normally would have. I don’t really understand how it has the data encoded, but I would assume that the real tree would go much deeper.

## Conclusion:

It seems that my implementation needs to shuffle the data in order to reach the same results as the sk\_learn version, especially for the cars data set. And while the random sampling of hyperparameters was not as helpful as I thought it would be, it looks as if the maximum depth has the most effect on accuracy up until a certain portion of the feature set size (probably relational through a log function).