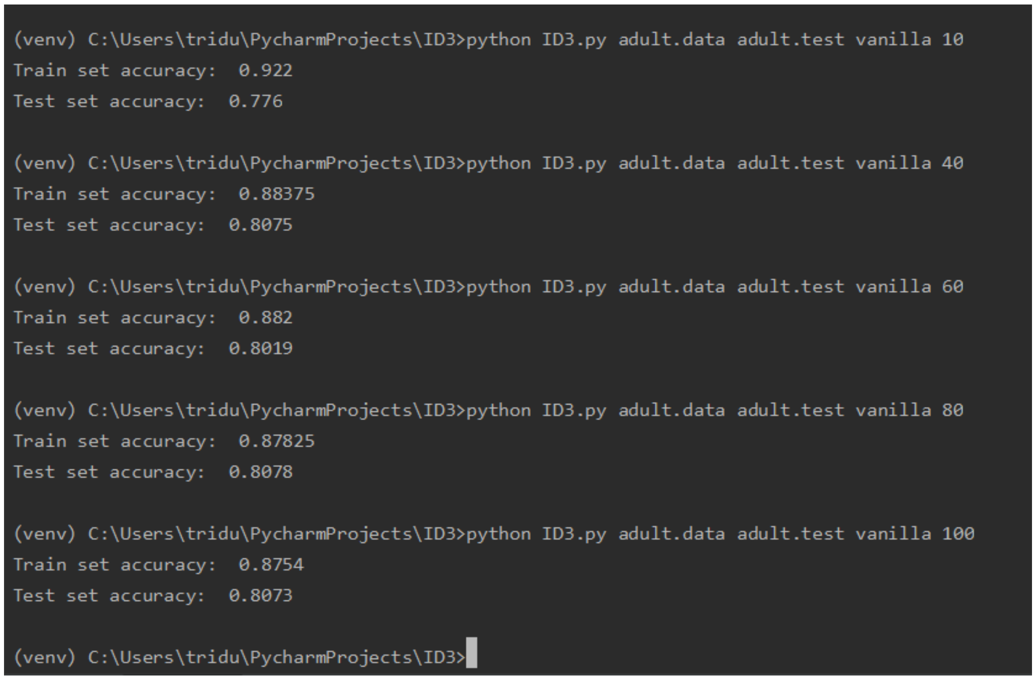
Part 1: Decision Tree

In ID3.py

Part 2: Analysis

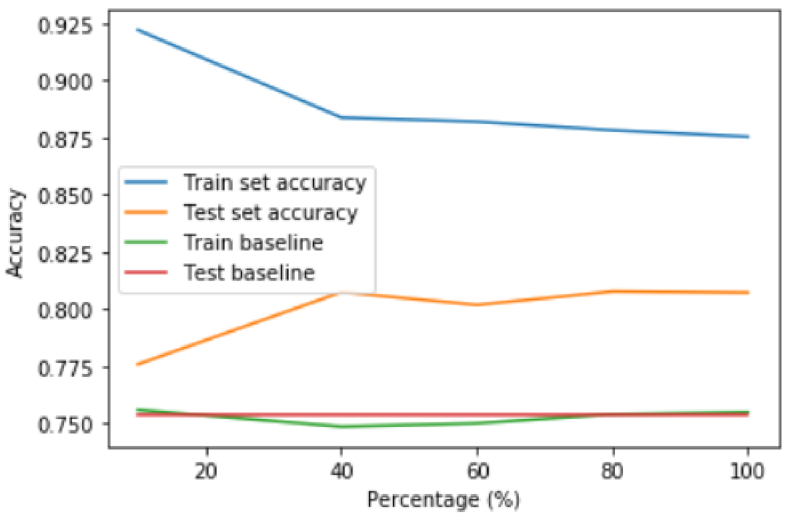
1. Pre-pruning:



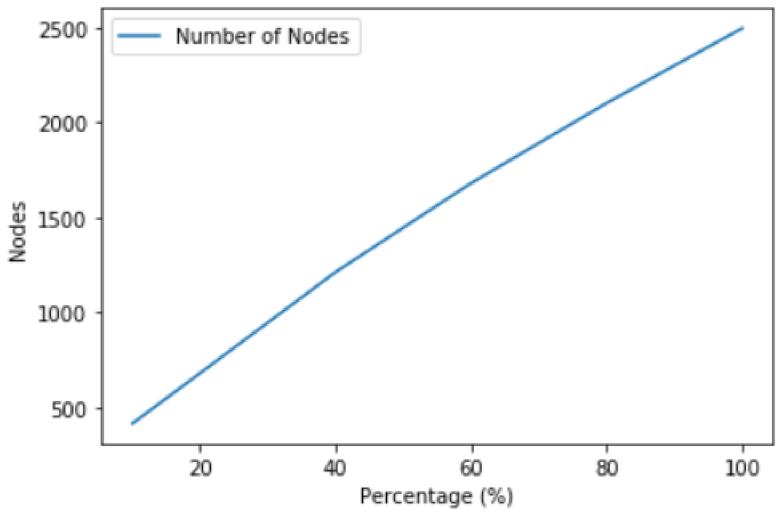
```
(venv) C:\Users\tridu\PycharmProjects\ID3>python ID3.py adult.data adult.test vanilla 10
Train set accuracy:  0.922
Test set accuracy:  0.776

(venv) C:\Users\tridu\PycharmProjects\ID3>python ID3.py adult.data adult.test vanilla 40
Train set accuracy:  0.88375
Test set accuracy:  0.8075

(venv) C:\Users\tridu\PycharmProjects\ID3>python ID3.py adult.data adult.test vanilla 60
Train set accuracy:  0.882
Test set accuracy:  0.8019

(venv) C:\Users\tridu\PycharmProjects\ID3>python ID3.py adult.data adult.test vanilla 80
Train set accuracy:  0.87825
Test set accuracy:  0.8078

(venv) C:\Users\tridu\PycharmProjects\ID3>python ID3.py adult.data adult.test vanilla 100
Train set accuracy:  0.8754
Test set accuracy:  0.8073

(venv) C:\Users\tridu\PycharmProjects\ID3>
```

(code for personal reference)

- 4 plots for 2 accuracies and 2 baseline error in 1 graph:



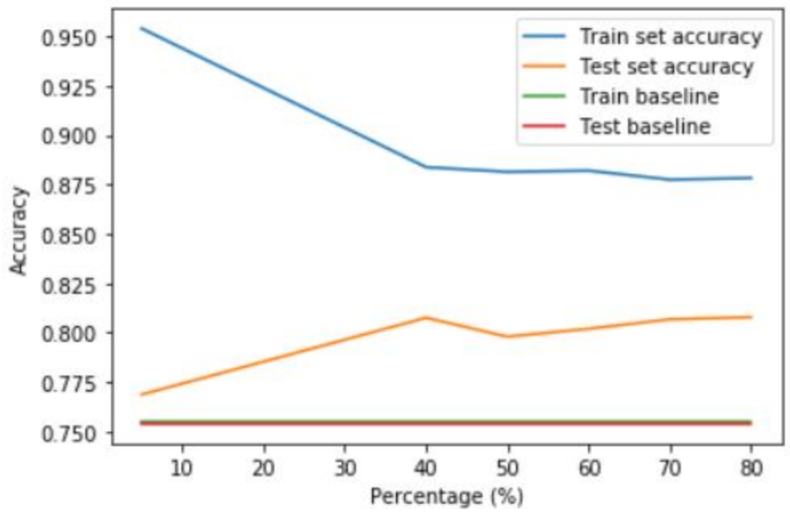
- Plot for number of nodes:



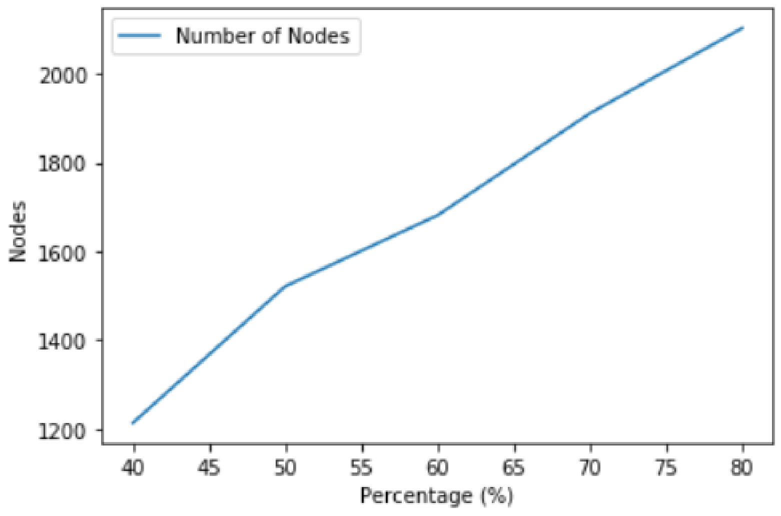
- As we can see from the graph, increasing the training set size decreases the train set accuracy, but it also increases the test set accuracy, making the 2 plots slowly converge. Therefore, increasing training set size decreases the difference between the 2 accuracies, but this increase does not really affect the baseline errors much (as the two baseline stays flat within a range of value). When the training set size is below 40%, the decision tree is overfitting because the different between train set and test set accuracy is much higher, meaning there is high variance but low bias in the training set. Therefore, increasing the training set size here somewhat fixes the problem of overfitting. Also, increasing training set size increases the number of nodes linearly, which means a large training set size learns more about the dataset.

2. Post-pruning:

- 4 plots for 2 accuracies and 2 baseline error in 1 graph:



- Plot for number of nodes:



- As we can see from the graph, increasing the training set size decreases the train set accuracy, but it also increases the test set accuracy, making the 2 plots slowly converge. Therefore, increasing training set size decreases the difference between the 2 accuracies, but this increase does not really affect the baseline errors much (as the two baseline stays flat within a range of value). Here, pruning did not affect much because not many nodes were pruned. When the training set size is below 40%, the decision tree is overfitting because the different between train set and test set accuracy is much higher, meaning there is high variance but low bias in the training set. Therefore, increasing the training set size here somewhat fixes the problem of overfitting. Also, increasing training set size increases the number of nodes linearly, which means a large training set size learns more about the dataset.

Part 3: Theoretical questions

1. **Will ID3 always include all the attributes in the tree?**
   No, because there will be time theoretically when all 8 inputs have rows with the same value, but the tree has not reached every attribute yet. Another case is that when all outputs are the same, but we have not reached every attribute yet.
2. **Why do we not prune directly on the test set and use use a separate validation set instead?**
   We prune on a validation set, then test it on test set to see whether or not the pruned tree is overfitting after pruning. If we just prune on the test set, we cannot test the variance of the data.
3. **How would you handle missing values in some attributes? Answer for both during training and testing.**
   Having a preprocessing step right after getting both datasets will effectively eliminate this problem. During preprocess for the train and test datasets, divide each multi-valued column into binary columns using get_dummies() in Pandas. Inside the columns of the dummies, a missing value will just yield a 0 in all of the attribute's dummies.
4. **How would you convert your decision tree from a classification model to a ranking model (i.e., how would you output a ranking over the possible class labels instead of a single class label)?**
   I can consider each attribute and measure the distance between each attribute to the other ones. In other words, I would try to see which one is the most closely related, choose it as the root, then repeat. That way, it becomes a ranking model.
5. **Consider the case where instead of binary values, the class label has continuous values. How would you adapt your decision tree to predict the class value of a test instance?**

I would create thresholds among the range of values of these continuous variables, so these columns would have 2 or more ranges of continuous values, and these are back to discrete values. Separate them into binary columns and build the decision tree just as before and we can predict these continuous values within a reasonable range of a class instance. We can either divide these continuous values based on interquartile ranges, or similar methods of dividing a continuous distribution.