Decision Tree

1. I correctly implemented the ID3 Algorithm and including the option to handle unknown attributes. I used information gain as the decider for splitting and I got 67% accuracy on the lenses data set and 100% on the example tennis dataset. See decision_tree.py for the code.

2. On the cars and voting data sets, I used the 10-fold CV. I purposely didn't use a stopping criteria to allow the tree to grow as far as possible. I've included a table below showing the accuracies at each fold of the cross validation and the mean accuracy. I also made sure to account for missing data by taking the most frequent value in the column and using it as the value for any missing data. I discuss this more in section 4. I got a 94% mean accuracy on the voting dataset and a 91% mean accuracy on the cars data set.

ID3 ALGORITHM		
Fold	Voting	Cars
1	0.9534883720930233	0.9069767441860465
2	0.9772727272727273	0.9364161849710982
3	0.9534883720930233	0.8901734104046243
4	0.9318181818181818	0.884393063583815
5	0.9302325581395349	0.9364161849710982
6	0.9318181818181818	0.9244186046511628
7	0.9302325581395349	0.9190751445086706
8	0.9772727272727273	0.9364161849710982
9	0.9069767441860465	0.9248554913294798
10	0.9090909090909091	0.9248554913294798
Mean:	0.9401691331923889	0.9183996504906572

This table shows the accuracies at each iterations through the cross-validations. Notice the mean value takes the accuracies at each iterations and takes the mean.

With a full tree the accuracy on the training set is often 100% or 99.4% in my case. The are a few reasons this happens, one reason is that the last node reached is pure and therefore will always get 100% accuracy, the other reason is that if you run out of features/attributes to split on, there are very low chances that the there are more than one instance where both have the exact same features/attributes, but different outputs. In the case that we have a very large dataset or our features/attributes have little variance, there may be a higher number of nodes that have subsets where the instances have the exact same attributes, but different outputs, producing bad results.

3. Cars:

   Based on the induced car set tree, I drew just the first few decisions the tree makes and found that the first split is on **Safety** which we can conclude is therefore the largest decider of a car's evaluation. Which makes sense, because it's the most important thing in deciding if a car is in acceptable condition. Then, within the safety_low branch, the tree splits next on 'persons' then 'buying' or 'maintenance'. Because these are the splits, we can assume that safety and number of persons is the most frequent decider for a car evaluation.

   Voting:

   Based on the induced voting tree, I drew just the first few decisions the tree makes and found that the first split was on the **Physician Free Freeze** which I read about online and found that it stops all physicians from charging 30 million elderly people. Basically this decision tree is learning the most controversial topics. Both the yes and no of the 'physician free freeze' split on immigration which means that whether or not they care about 'physician free freeze', the tree still needs to know their stance on immigration. The tree learned quite well because in our country immigration is actually a very controversial topic and pushes people to take a specific political standing. The tree then went on to split on education and religion in schools as the most likely deciders of republican vs democrat. Most people I know are republicans and it's interesting to see the similarities between what the tree is learning and the the first questions you should ask someone to decide their political standing, like religion, education, use of funds and immigration.

4. In the voting problem for the missing attributes, I computed the most common class and chose that to be the missing attribute. These substitutions were necessary both before training as well as during the predictions. The predictions, I simply used the same value used in the training. I chose this based on the fact that there is a higher probability that the missing value is the same as the most common class than any other. Although that could be false if our dataset is too small or a bad representation of the actual population.

5. To avoid overfitting, I created a method called prune() which is called recursively and works from the top down testing the accuracy at each stem of the tree. See decision_tree_prun.py for the pruning example.

	Accuracy	Depth	# of Nodes
Voting-Mean	0.9401691331923889	12	69
Cars-Mean	0.9183996504906572	7	408
Voting-Pruned	1.0	3	8
Cars-Pruned	1.0	4	7

This table shows the similarities and differences between the depths and sizes of voting and car decision trees, both pruned and unpruned.

6. For my extra experiment, I decided to treat the nodes as I would in perceptron and trained them by sending the features through the tree and labeling stopping node with +1 weight for a specific output, and an addition incrementation of +1 for each layer of tree, such that a tree with five layers would have a +5 weight added at the bottom layer node. As a result, I had built a tree that at each of the leaf nodes there's an array of weights for each output and during testing, I selected the output with the greatest weight instead of selecting the most common output.

   This experiment wasn't very successful. I found that I couldn't get a better accuracy than 70% and even that may have been luck. I included a graph similar to the one above that shows the accuracy for this experiment.

   See decision_tree_exp.py for the code.

EXTRA EXPERIMENT		
Fold	Voting	Cars
1	0.7209302325581395	0.6802325581395349
2	0.5227272727272727	0.7341040462427746
3	0.4883720930232558	0.7109826589595376
4	0.5909090909090909	0.6994219653179191
5	0.5348837209302325	0.6416184971098265
6	0.6363636363636364	0.6627906976744186
7	0.6511627906976745	0.7398843930635838
8	0.6590909090909091	0.7225433526011561
9	0.6046511627906976	0.6589595375722543
10	0.7272727272727273	0.7514450867052023
Mean:	0.6136363636363636	0.7001982793386208

This table shows the accuracies at each iterations through the cross-validations.

These numbers may be low due to the difference in weights at each level. Or, they may be this way because of my base cases that I kept the same. The base cases often decided the prediction. Yet, the actual predict() function is very different because I'm selecting based on weight and not on most common output, that is probably the reason for these low accuracies.