Free-response Questions (4 points)

To answer some of these questions, you will have to write extra code (that is not covered by the test cases). The extra code should import your implementation and run experiments on the various datasets (e.g., choosing `ivy-league.csv` for a dataset and doing `experiment.run` with a 80/20 train/test split, averaged over a few trials). **You do not need to submit this extra code.**

1. Assume you have a deterministic function that takes a fixed, finite number of Boolean inputs and returns a Boolean output. Can a Decision Tree be built to represent any such function? Give a simple proof or explanation for your answer. If you choose to give a proof, don't worry about coming up with a very formal or mathematical proof. It is up to you what you want to present as a proof.

   The answer is yes. I think it basically doesn't matter what the function is. We can represent it as a pair of features & targets arrays. E.g. In the 2-parameter case, f(a, b) => c can be represented as features = [[T, T],[T, F],[F, T],[F, F]] and whatever targets the function produces e.g. targets = [T,T,F,F]. We can fit the tree on these features and targets (being careful of how we make predictions when a leaf node has the same number of each class). Whatever tree we get after that will represent the function. The way we know its representing the function is by passing in the training data and getting the same result. The question is basically equivalent to "Can a binary decision tree achieve 100% accuracy on the training data?" The answer is yes. In fact, trees are pretty good at overfitting.

   The key is that the function is deterministic, the same way a binary tree is deterministic i.e. when we give a binary tree the same input, it always produces the same output.

2. Let $\square$ be a voter in the set of voters $\square$. Let each $\square$ have a value, assigned to either -1 or 1. Let's define Majority-rule as the sign of the sum all voters. If the sum is negative, it returns -, else it returns +. Assume an odd number of voters (so, there are no ties). Can the Majority-rule algorithm be represented by a Decision Tree that considers a single voter at each decision node? Why or why not? Note that the question is not asking if a Decision Tree can learn Majority-rule in general. It is asking if this particular algorithm, as described, can be learned.

   - The key factor here is that we have an ODD number of voters. This makes the outcome of any input of voters deterministic i.e. we will either get a "-" or a "+" as a result. We can make a table of features where each column represents the possible vote (-1 or 1) for a given voter. Overall, we would have 2^n number of rows for n number of voters. The table will look very similar to the data we have in the majority-rule.csv except that 0's will be -1's and we will have an odd nubmer of feature columns (the .csv file has 6 feature columns).
   - The csv file actually represents an example of why a decision tree cannot represent the Majority Rule algorithm for an EVEN number of voters. We see in the csv that whenever there are an equal number of 1's and 0's, the class appears to be 0. But for the Majority-rule algorithm, this is not defined. In other words, a decision tree can only represent a deterministic function, where as the majority rule algorithm is not deterministic for an EVEN number of features.

3. In the coding section of this assignment, you trained a Decision Tree with the ID3 algorithm on several datasets (`candy-data.csv` , `majority-rule.csv` , `ivy-league.csv` , and `xor.csv`). For each dataset, report the accuracy on the test data, the number of nodes in the tree, and the maximum depth (number of levels) of the tree.

data	acc (20% test set)	# nodes	# levels
candy	70.5%	18	8
majority rule	61%	24	6
ivy-league	92.3%	17	5
xor	100%	7	3
PlayTennis	66%	11	4

4. What is the Inductive Bias of the ID3 algorithm?

   - making the shortest tree possible

5. Explain what overfitting is, and describe how one can tell it has occurred.

   Overfitting is when our ML model fits the data "too well" i.e. it is not capturing some real relationship between features and targets, but rather recreates whatever noise might be in the data.

   We can tell that we are overfitting when the accuracy of our model on our training dataset is much higher than the accuracy of our model on a test (holdout) set. This process can be repeated with different train/test splits or even better, via cross-validation.

   A good way of avoiding overfitting is via regularization i.e. introducing some "penalty" for the model complexity. The right amount of penalty can be calculated via cross-validation. This process of finding the right penalty can also be called hyper-parameter tuning.

6. Explain how pre- and post-pruning are done and the reason for doing them.

   Pre-pruning and post-pruning are both methods for avoiding overfitting for tree-based models.

   pre-pruning establishes certain stopping criteria for growing a tree e.g. maximum depth of a branch, minimum number of observations in a leaf node, etc.

   post-pruning occurs after a tree has been fully trained. At that stage, certain non-leaf nodes are picked and converted into leaf nodes (effectively discarding all nodes below). If there isn't a significant difference in the tree's performance (on a validation set) then the pruning is kept. If the change in performance is large, then the tree is restored and a new pruning is tried. This process continues, for example, until a certain number of iterations.

7. One can modify the simple ID3 algorithm to handle attributes that are real-valued (e.g., height, weight, age). To do this, one must pick a split point for each attribute (e.g., height > 3) and then determine Information Gain given the split point. How would you pick a split point automatically? Why would you do it that way?

   I would use the median of the continuous feature. This will ensure that I have roughly equal number of observations in each branch of the node. In this way, the tree won't be skewed by outliers.

8. Ensemble methods are learning methods that combine the output of multiple learners. The hope is that an ensemble will do better than any one learner, because it reduces the overfitting that a single learner may be susceptible to. One of the most popular ensemble methods is the Random Forest. The high level idea is to build multiple Decision Trees from the training data. One way to build different Decision Trees from the same data is to train several Decision

Trees on different random subsets of the features. If, for example, there are 20 measurable features, you randomly pick 10 of the features and build a tree on those 10, then you randomly pick another 10 features and build a tree using those 10 features. If you were building an ensemble of trees this way, how would you combine their decisions in the end? Explain why you would choose this method. Feel free to provide a citation for your choice (if you cite something, please ALSO provide a hyperlink), but also explain the reason this is your choice.

In order to answer this question I read the Wikipedia article on Random Forests. It says: "bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets."

I initially thought about calculating a weighted average to give a final prediction from a Random Forest, but I think that would be a bad idea. My idea was to use weights proportional to how well the specific tree fits the data. However, this would bias the prediction towards trees, which tend to overfit.

The simplest, and probably better way, is as suggested by the wikipedia article. Namely, just taking a majority vote i.e. the most common prediction from all the trees. E.g. if the ensemble has 3 trees, and 2 say True, while the other one says False, then the prediction would be True (assuming binary T/F classes).