Answers for HW 3

Happy Ta
COMP-5710
Professor Richard Freedman

Question 1

Part A

i : How can we keep track of the node's purpose in the code with this boolean/binary variable? Because decision nodes are always the leaves of the tree, how could we check the type of node without a specific variable?

- For my implementation, I used a binary variable called `node_type` to keep track of the node's type. 0 represent a query node, 1 represent a decision node.
- If we did not have `node_type` available, we can check if the node have any children node, if not then it is probably a decision node.

ii: Why is it sufficient to only have one variable store either value rather than two variables for feature and label? How does this help save memory?

- It is sufficient to only have one variable for both feature and label because both value are of the same type and only 1 is needed at one time, so when a node is a decision node, we do not care about the feature, and when we are a query node, we do not care about the label.
- This can also save memory by requesting less memory, instead of need 2 extra int size memory chunk when the object is created, we only need 1 extra int size memory chunk.

iii: What do the key and value for this hash map/dictionary represent with respect to the decision tree node's children? How does the hash map/dictionary data structure save runtime during decision tree prediction when we have the response from the input feature (passed in as an argument)?

- The key in this hash map/dictionary will be the position, either left or right, the value will be a node object, representing the children nodes.
- Using a hash map/dictionary is great for runtime because any operation using this data type can result in O(1), greatly improve performance compare to other data structure.

Part B

Train function *ii*: How does changing the control flow only when the choice of a hyper-parameter reduce the amount of code we write if the rest of the control flow is identical regardless of hyper-parameter choice?

- By changing the control flow based on the hyper-parameter help reduce code duplication and from that reduce the amount of code we write.

iii: How does a recursive function build a tree when it only return a pointer to one decision tree node?

- The node that is return is the root of the tree and as the function recursively run, it will add children node to this root and slowly build out a tree.

Predict Function

iii: If we had access to a GPU could we perform each prediction in parallel; why or why not?

- We can definately perform the prediction in parallel, this is because each prediction is independent from each other.

Question 2

i: Would the random forest class still compile/interpret and run as expect if we changed the type the list stores from decision tree pointers to ML algorithm pointers; why or why not? If we added a hyper-parameter selecting the specific ML algorithm, then what would change in the train function to get the desired classification models? Why would the test function not have to change despite the change in algorithm?

- The random forest class would still likely compile if we change the type of list as long as it contain the same properties expected by the random forest class. Furthermore, since most programming languages support polymorphism, as long as these claass share the same interface, they should be able to compile.
- If we add a hyper-parameter that choose a ML algorithm, we need to add some form of switch-case clause in the train function so it can dynamically instantiate to fit the selected algorithm
- The test function does not need any changes because it usually does not care about the algorithm used to train the models, it only use the model for prediction. As long as all models implement some kind of `predict()` function, it should work fine.

Question 3

i: How likely is a decision node within the decision tree to correctly predict a label for a random input in our dataset if its training data subset has only one class label? If all the features are queries before making a decision (regardless of the order in which the decision tree queried them), is it possible to gain any more information from a future query; why or why not? How do these two questions' responses contribute to over-fitting on the training dataset?

- A decision node trained on a subset of data with only one class label, will always predict that class for any input received.
- If all features has been queried, making any more query will not give us any more information. This is because the decision tree would have considered all available features values to partition.
- These two questions display the possibility of over-fitting for decision trees, both with single class training and all feature queries.

ii: If each decision tree in the random forest trains on q% of the training dataset samples, then what is the greatest and least amount of training dataset overlap that two decision trees in the random forest could have? If two over-fit decision trees each share r% of their training datasets, then what is the probability that exactly one, both, or neither will correctly classify some random input found in the union of their training datasets (combining both datasets into one larger dataset) when sampled uniformly (all rows in the dataset are equally likely to be chosen)? Which of the probabilities in these previous question's response will increase or decrease as the number of decision trees considered increases; why?

- Greatest overlap: Two decision trees can have, potentially, **100%** overlap if they are training on the same subset.
- Least overlap: This happen when two decision trees is trained on completely disjointed subset of training dataset.
- If two over-fit decision trees each share r% of their training datasets, then what is the probability that exactly one, both, or neither will correctly classify some random input found in the union of their training datasets (combining both datasets into one larger dataset) when sampled uniformly (all rows in the dataset are equally likely to be chosen)?
  - Exactly one tree correct:
    - This can only happen if the training dataset appear in 1 tree but not the other.
  - Both tree correct:
    - For this to happen, the random input must be part of the overlap portion of their dataset.
  - Neither Tree correct:

    - For this to happen, the random input must be outside of the union of their training dataset.

- As the number of decision trees increase, then the chances of both trees predicting since the overlapping would increase with more decision trees. Because of this, the chance of only 1 tree predict correctly would be lower since there's lower chance of the random input being in 1 tree dataset. Same can be said for the probability of Neither Tree being correct, this is be cause the chance of the random input being outside of the Trees union dataset would also reduce as more decision trees is used.