CMPT 459 - Assignment 1
Shan Ying Yang
301305759 (syy7)
March 5th, 2021

Question 1: Pseudocode for Decision Tree with Gini Index

```
class DecisionTree(object):
    def __init__(self,max_depth,min_samples_split,max_features):

      # calling constructors initializes the required values

        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.max_features = max_features

        self.root = root

    def fit(self, X: pd.DataFrame, y_col: str)->float:

        # process features from dataset
        features = self.process_features(X, y_col)
       # calls the generate_tree function and built tree
        self.root = self.generate_tree(X, y_col)

    def generate_tree(self, X: pd.DataFrame, y_col: str,   features: Sequence[Mapping])->Node:

         # generate trees by calling self.split_node function, dataframe X, string y_col, and features
are passed in as parameters
        root = Node(X.shape[0], X[y_col].mode()[0], 0)
        self.split_node(root, X, y_col, features)

        return root

    def split_node(self, node: Node, X: pd.DataFrame, y_col:str, features: Sequence[Mapping]) -> None:

        # If current dataset contains instances of only one class then return
        if (
            (node.depth >= self.max_depth) or
            (len(X) <= self.min_samples_split) or
            (node.single_class == True) or
            len(X[y_col].unique()) <= 1
            ):
          node.is_leaf = True
          node.single_class = True
          return
        else:
          node.is_leaf = False
        # randomly select x% of the possible splitting features in X
        randomly_selected_features = random.choices(features, k=random.randint(2, self.max_features))

        # Select the feature F with the highest gini index
        maxGain = 1
        maxGain_attribute = None

        # split items into attributes list
        attributes = []
        for i in features:
          attributes.append(i['name'])

        # calculate the best gain value using the gini index function
        for col in attributes:
          best_gain = gini(X, X[col], y_col)
          if best_gain < maxGain:
            maxGain = best_gain
            maxGain_attribute = col

        # set node's name to the maxGain attribute
        node.name = maxGain_attribute
        ## if the best gain's feature is categorical value
        if maxGain_attribute not in self.numerical:

          # set node's numerical status
          node.is_numerical = False
          best_feature_all_class = X[maxGain_attribute].unique()

          # iterate in all unique classes of the best attribute's column
          for cur_class in best_feature_all_class:
            cur_dataset = X[X[maxGain_attribute] == cur_class]
            n_current = len(cur_dataset[y_col])

            # create new node with the new dataset's information, and set node class as the mode of
  the target column of the new dataset, increment node.depth
            new_node = Node(n_current, cur_dataset[y_col].mode()[0], node.depth+1)
            new_node.name = maxGain_attribute
            new_node.is_numerical = False

           # saves and update current node → children and class status
            node.single_class = False
            node.children[cur_class] = new_node

           # recursively calls split_node until node is single class/leaf
            self.split_node(new_node, cur_dataset, y_col, features)

        # if column has numerical values
        Else:
        # update node status with the threshold and is_numerical
          node.is_numerical = True
          node.threshold = self.split_value

          cur_dataset_L = X[X[maxGain_attribute] < node.threshold]
          cur_dataset_U = X[X[maxGain_attribute] >= node.threshold]

          # checks if splitted datasets are valid for further splitting
          if len(cur_dataset_L) < 1 or len(cur_dataset_U) < 1:
            node.is_leaf=True
            node.single_class = True
            return
          new_node_l = Node(len(cur_dataset_L), cur_dataset_L[y_col].mode()[0], node.depth+1)
          new_node_l.name = maxGain_attribute
          new_node_l.is_numerical = False

          # save node to children list (less than nodes)
          node.children['l'] = new_node_l
          node.single_class=False
          self.split_node(new_node_l, cur_dataset_L, y_col, features)
          new_node_ge = Node(len(cur_dataset_U), cur_dataset_U[y_col].mode()[0], node.depth+1)
          new_node_ge.name = maxGain_attribute
          new_node_ge.is_numerical = False

          # save node to children list (greater than nodes)
          node.children['ge'] = new_node_ge
          node.single_class=False

          # recursively calls split_node until node is single class/leaf
          self.split_node(new_node_ge, cur_dataset_U, y_col, features)
          return
        return

    def gini_calc(self, lower, upper):
    # helper function for computing the gini index with splitted values
      split_gain = 0
      for targets in [lower, upper]:
        gini = 1
        for i in range(len(targets.unique())):
          prob = targets.value_counts()[i] * 1.0/len(targets)
          gini -= prob ** 2
        split_gain += len(targets)*1.0/(len(lower)+len(upper))*gini
      return split_gain

    def gini(self, X: pd.DataFrame, feature: Mapping, y_col: str) -> float:
        unique_f = None
        #  checks whether passed in feature is numerical or not
        if feature.name in self.numerical:

          cur_dataset = X[[feature.name,y_col]].sort_values(feature.name)
          unique_f = list(cur_dataset[feature.name].unique())
          best_gain = 1

        # user percentile to find the split value for the gini index calculation
          unique_f = np.percentile(unique_f, [85,75,50,25,15])
          for i in unique_f:
            lower = cur_dataset[cur_dataset[feature.name] < i][y_col]
            upper = cur_dataset[cur_dataset[feature.name] >= i][y_col]
          # calls helper gini_calc function to compute the best_gain value
            best_gain = self.gini_calc(lower, upper)
          # update split value
            self.split_value = i
        else:
        # if feature values are categorical
          cur_dataset = X[[feature.name,y_col]].sort_values(feature.name)
          unique_f = cur_dataset[feature.name].value_counts(sort=True)
          total_size = unique_f.sum()
          best_gain = 1
          prob = 0
          # computing split value
          for i in unique_f:
            prob += (i/total_size)
            best_gain -= prob ** 2
        return best_gain

# Node class follows structure provided in RandomForest.py
class Node(object):
    def __init__(self, node_size: int, node_class: str, depth: int, single_class:bool = False):
        self.is_leaf = True
        self.name = None
        self.children = {}
        self.is_numerical = None
        self.threshold = None
        self.node_class = node_class
        self.size = node_size
        self.depth = depth
        self.single_class = single_class
        self.mode_val = None
    def set_children(self, children):
        self.is_leaf = False
        self.children = children
    def get_child_node(self, feature_value)-> 'Node':
        if not self.is_numerical:
            return self.children[feature_value]
        else:
            if feature_value >= self.threshold:
                return self.children['ge'] # ge stands for greater equal
            else:
                return self.children['l'] # l stands for less than

   def calc_predict_val(self, X, tree:Node):
      # returns value when tree’s node is a leaf, else recursively iterates until value is obtained
      if (tree.is_leaf == True):
        return tree.node_class
      else:
        value = X[tree.name]

        if tree.is_numerical == False:
          if value in tree.children:
            return self.calc_predict_val(X, tree.get_child_node(value))
          else:
            return tree.node_class
        else:
          return self.calc_predict_val(X, tree.get_child_node(value))
```

Question 2:

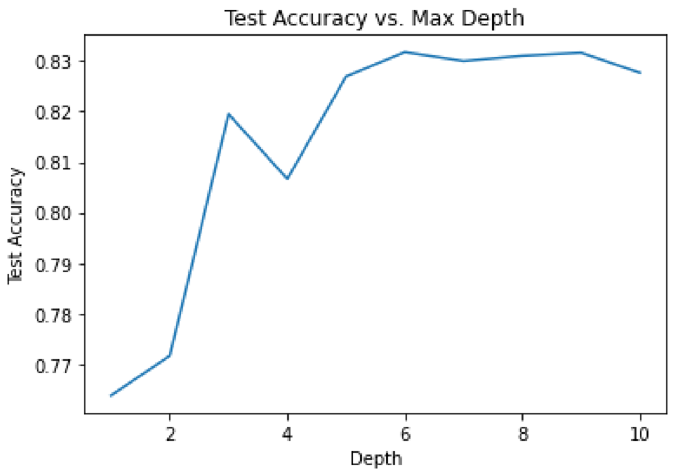
Random Forest Classifier Accuracy		
	Train Data	Test Data
Q1-gini	0.8444458094038881	0.82943308150605
Q2-information gain	0.8471177175148183	0.8364351084085744
Q4_1-gini	0.7591904425539756	0.7637737239727289
Q4_2-gini	0.7688031694358282	0.7716356489159143
Q4_3-gini	0.8227020054666626	0.8194828327498311
Q4_4-gini	0.8161911489204877	0.8066457834285363
Q4_5-gini	0.835232333159301	0.826914808672686
Q4_6-gini	0.8431559227296459	0.8317670904735581
Q4_7-gini	0.8443843862289242	0.8299858731036177
Q4_8-gini	0.8467491784650348	0.8310300350101345
Q4_9-gini	0.8500046067381223	0.8316442478963209
Q4_10-gini	0.8483154694266146	0.8276518641361096

1. n_classifiers = 10, maxdepth = 10, min_sample_split = 20, max_features = 11, criterion = gini:
   a. training data accuracy is 0. 8444458094038881with Gini Index as the criterion, and testing data accuracy is 0.82943308150605.
2. n_classifiers = 10, maxdepth = 10, min_sample_split = 20, max_features = 11, criterion = entropy:
   a. accuracy is 0. 8471177175148183 with Information Gain as the criterion, and testing data accuracy is 0. 8364351084085744.

Yes, changing the criterion does affect the accuracy of training and test datasets. It seems like that the model gives a better accuracy with information gain than using Gini index. Gini index usually is better at predicting datasets with large partitions, while the information gain favors smaller partition with more diversified values. When dealing with categorical values, Gini Index returns an index based on binary splitting. But with information gain, it calculates the entropy differences based on splitting. Since the max_features for both models were initialized as 11 (total of 14), the model performs slightly better with information gain due to potential increase of sampled features and partitions. The training data accuracy for both models are better than the testing accuracy.

3. The accuracy of my training data is not equal to 100% accuracy because my random forest classifier uses bagging and feature randomness when classifying the data. The dataset for each tree is randomly sampled (randomized rows), and the feature columns are also randomly chosen at each node split of the tree.
   a. Yes, there are ways to train a tree that yields 100% accuracy on the training dataset, and that is to not randomly sample the data/feature columns. The 100% accuracy can be achieved by trying to cover all of the data in the training dataset, and thus model will cache inaccurate values (from noises) and result in overfitting of the model.
   b. The dataset passed in for each tree generation would have to stay the same → so no sampling on the dataset. The features at split node would also need to be normalized instead of randomized, so that the columns are consistent. But random forest classifier uses the randomness feature and bagging, so the other ways to increase the accuracy by increasing the max_depth parameter. Such a classification model has a high variance and low bias. The model has low bias because the made decisions are not careful enough for each tree. The model has high variance when it attempts to learn everything.

4. The graph below is a plot of the accuracy of the 10 models on the test dataset:



The test accuracy curve has a positive increasing trend, which denotes that the prediction accuracy of the random forest model increases as the depth increases. It makes sense as the depth variable increases, each tree could potentially be splitting against more nodes. Deeper decision trees tend to have higher accuracy, but at the same time it also tends to overfit the dataset. We can see that the curve surges at around depth equals to 3, and then maintains a more consistent trend at depth equals to 5. The phenomenon thus concludes that the max_depth parameter does not need to be as extremely high to obtain high accuracy, and reasonable tree depth yields similar accuracy. Moreover, from examining the graph I can see that there are spikes at some points, where the test accuracy fluctuates more (depth 3 to depth 5). This could be from overfitting the data, resulting more noises added to the predicted values.