Assignment-4

Surendra Parla

4 March 2024

1 1

Instead of binary normalization I have used quantiles which are customizable through the parameter "num_of_splits_per_feature". For example if we choose this hyper parameter as 3 we would validate 25 percentile, 50 percentile (median) and 75 percentile and take the best of these (which gives maximum mutual information).

2 2

Mutual Information $I(x_j, y)$ is calculted by the function "mutual_information" (code in the appendix).

3 3

Generalized Decision Tree is implemented as the code is provided in the appendix. The stopping criteria used in this code is either small mutual information of about 0.05 is reached or when the subsample size is less than or equal to 5 percent of entire dataset. These, fields can be adjusted through parameters and can be tuned as hyper parameters as well to avoid overfitting.

4 4

Decision Tree obtained from the dataset is:

5 5

Code for cross validation is attached in the appendix and the accuracy is 79.54

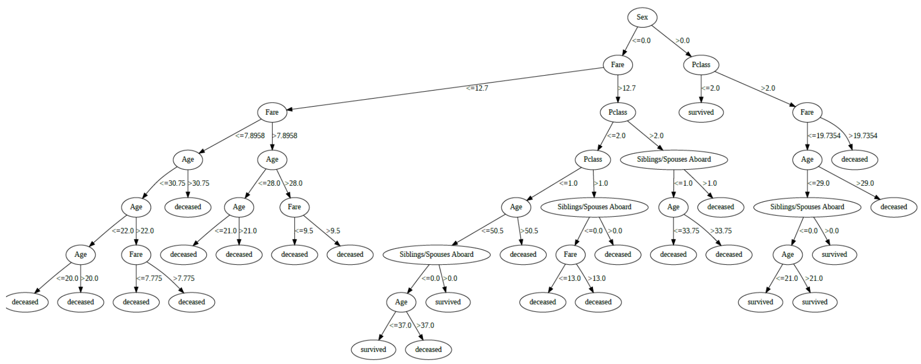

Figure 1: Decision Tree for Titanic Dataset

6 6

My feature vector is

$$X = \begin{pmatrix} 2 \\ 0 \\ 25 \\ 0 \\ 0 \\ 20 \end{pmatrix}$$

The out from decision tree is 0 which represents I would be deceased which is inline with what we observed in logistic regression.

7 7

7.1 a

Random forest for 80% random data sampling Figure 2 to 6

7.2 b

Accuracy obtained by 10 fold cross validation on random forest is 79.1%

7.3 c

According to this random forest i would have not survived as all 5 trees predicted 0.

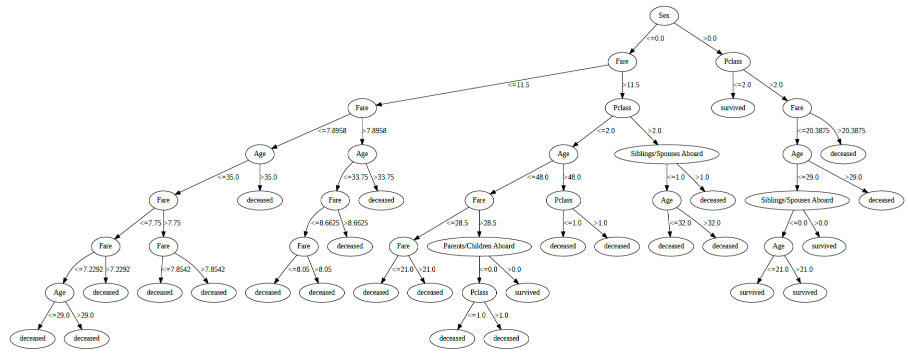

Figure 2: Random Forest 1

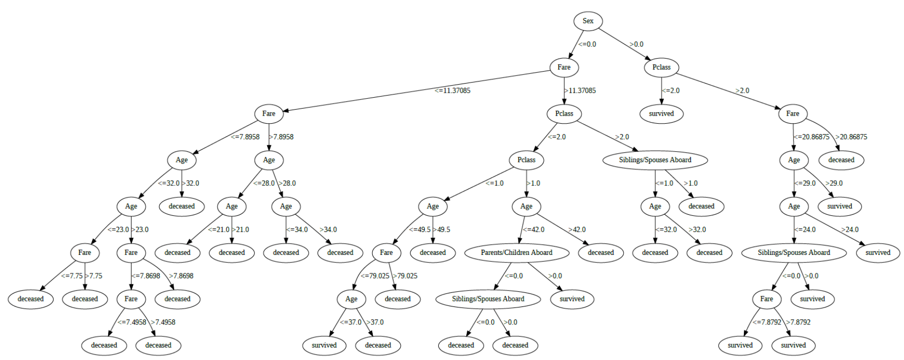

Figure 3: Random Forest 2

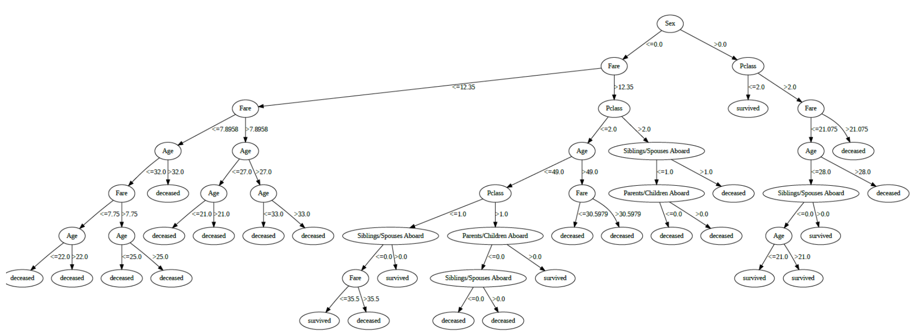

Figure 4: Random Forest 3

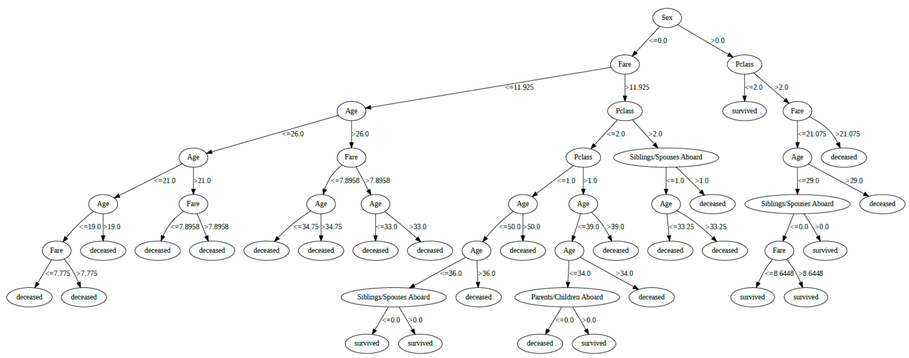

Figure 5: Random Forest 4

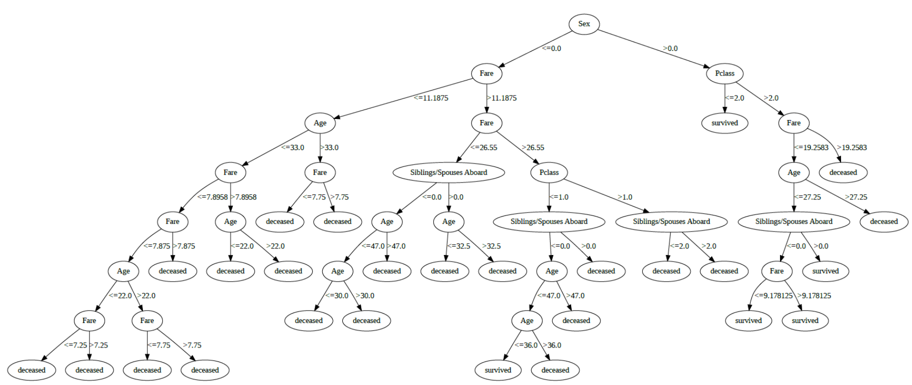

Figure 6: Random Forest 5

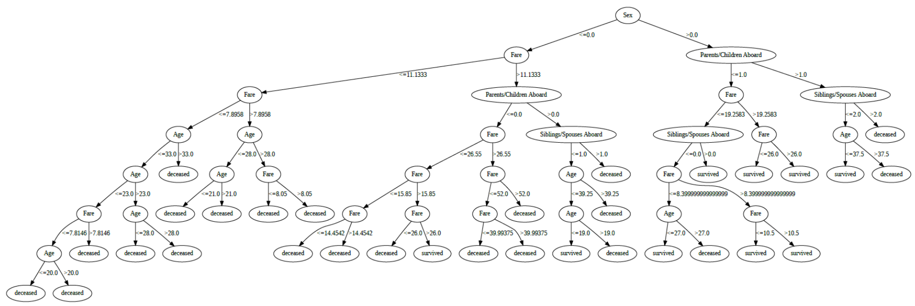

Figure 7: Random Forest excluding feature Pclass

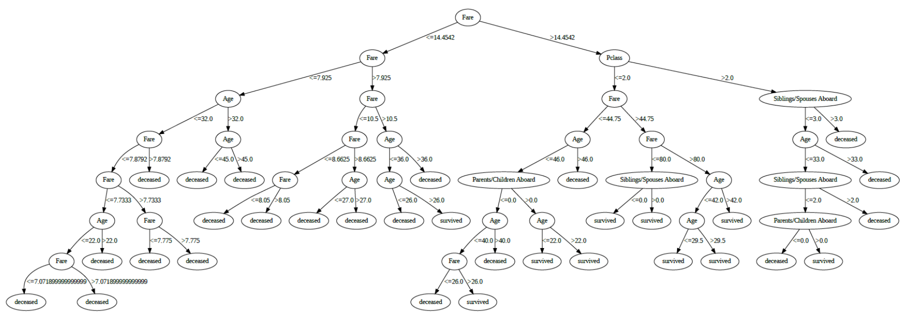

Figure 8: Random Forest excluding feature Sex

8 8

8.1 a

Random forest trees: Figure 7 to 12

8.2 b

Accuracy obtained by leave out method is 80%

8.3 c

Applying this model to the own feature we get [[1], [0], [0], [0], [0], [0]]. So, it shows we wont survive which is inline with decision tree.

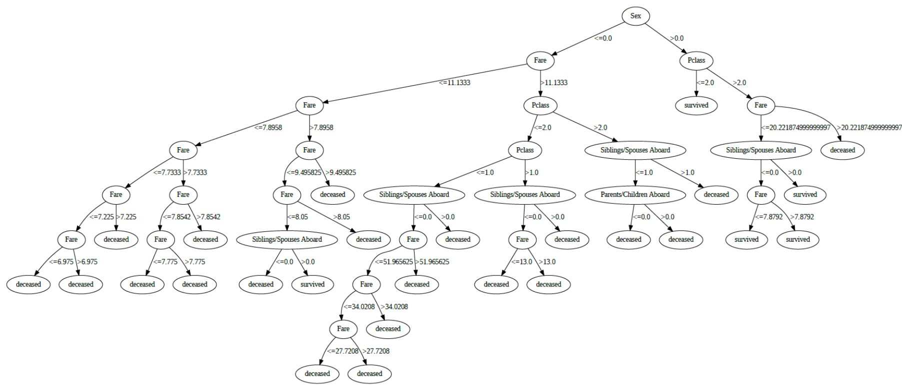

Figure 9: Random Forest excluding feature Age

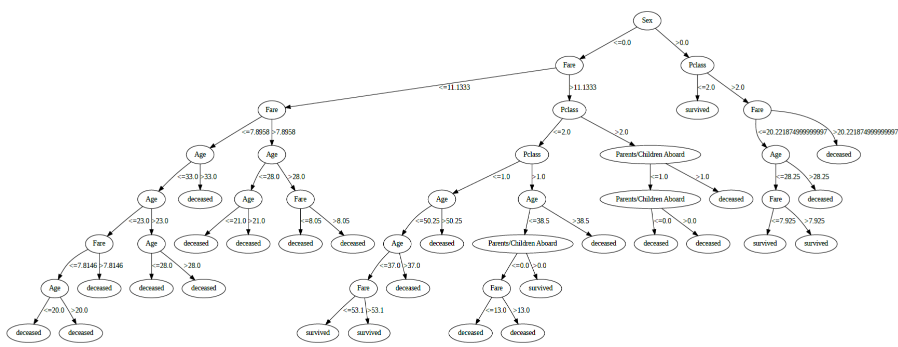

Figure 10: Random Forest excluding feature Siblings/Spouses Aboard

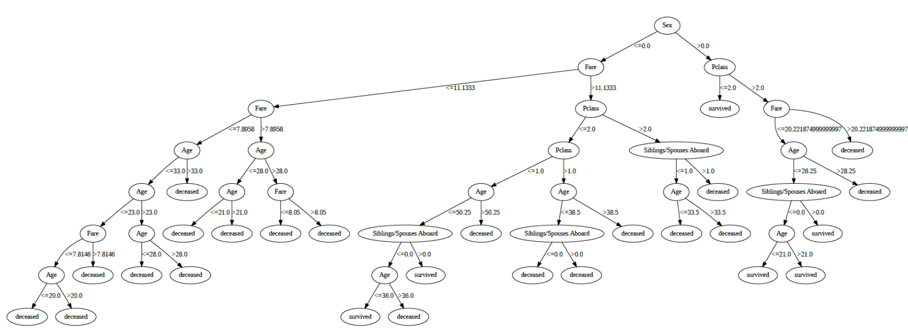

Figure 11: Random Forest excluding feature Parents/Children Aboard

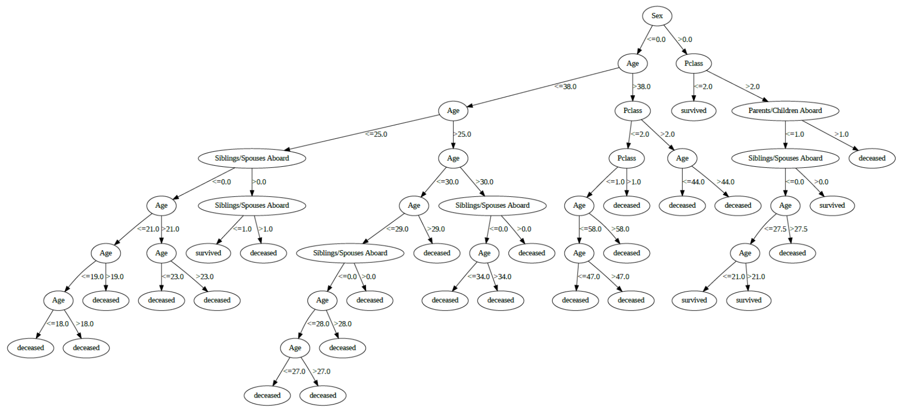

Figure 12: Random Forest excluding feature Fare

9 9

Yes, all the models random forest with leave out, random sampling, decision Tree and logistic regression predicted 0 for my feature vector. Logistic regression is the simplest model and easy to use and it is not effected by as we can stop after few iterations and its easy to tune. We can also get the confidence interval for the predictions. Its easy to store and export the model as we have just an array of parameters. On the other hand decision trees and random forest provide intuitive graph representation of decision taken at each step. So, we get the advantage of explainability. Decision tress might be effected by the overfitting and bias where cross validation helps in minimizing this as we test across different batches and random sampling. So, I would like to use random forest with cross validation. The observed accuracies are:

Model	Accuracy(%)
Logistic Regression	80
Decision Tree with cross-validation	79.54
Random Forest with sampling	79.1
Random Forest with feature dropout	80

10 10

$$
\begin{aligned}
I(X;Y) &= \sum_{x\in\mathcal{X},y\in\mathcal{Y}} P(x,y)\log\frac{P(x,y)}{P(x)P(y)} \\
&= \sum_{x\in\mathcal{X},y\in\mathcal{Y}} P(x,y)\log\frac{P(x,y)}{P(x)} - \sum_{x\in\mathcal{X},y\in\mathcal{Y}} P(x,y)\log P(y) \\
&= \sum_{x\in\mathcal{X},y\in\mathcal{Y}} P(x)P(y|x)\log P(y|x) - \sum_{x\in\mathcal{X},y\in\mathcal{Y}} P(x,y)\log P(y) \\
&= \sum_{x\in\mathcal{X}} P(x)\left(\sum_{y\in\mathcal{Y}} P(y|x)\log P(y|x)\right) - \sum_{y\in\mathcal{Y}} P(y)\log P(y) \\
&= -\sum_{x\in\mathcal{X}} P(x)H(Y|X=x) - \sum_{y\in\mathcal{Y}} P(y)\log P(y) \\
&= -H(Y|X) + H(Y) \\
&= H(Y) - H(Y|X) \\
&= I(Y;X)
\end{aligned}
$$

11 Appendix

Accuracy by 90:10 split is around 80%.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math

df = pd.read_csv('titanic_data.csv')
features  = df.iloc[:,1:].columns.values

split_ratio = 0.9
num_rows = int(split_ratio*len(df))
X_train, X_test = df.iloc[:num_rows,1:].to_numpy(), df.iloc[num_rows:,1:].to_num
Y_train, Y_test = df.iloc[:num_rows,:1].to_numpy(), df.iloc[num_rows:,:1].to_num

percentiles = np.linspace(0, 100, 3 + 2)[1:-1]
X = np.array([1,2,3,54,22,2,1,34,54,26])
quantiles = np.percentile(X, percentiles)

#section 4.2, 4.3
class DecisionTree:
```

```
def __init__(self, num_of_splits_per_feature):
    self.num_of_splits_per_feature = num_of_splits_per_feature
    self.num_of_samples = 0
    self.root = None

def fit_training_data(self, X_train, Y_train):
  self.num_of_samples = len(Y_train)
  # self.Y_train = Y_train
  self.root = self.build_tree(X_train, Y_train)

def build_tree(self, X_data, Y_data):
  subsample_size = len(Y_data)
  #verify if the sample is less than 5% of whole data
  #if yes exit as leaf node
  if subsample_size <= int(0.05*self.num_of_samples):
    # print('current path ends as the sample space is less than 5% of total sa
    node = TreeNode(None, None)
    node.isLeaf = True
    if np.sum(Y_data) >= subsample_size//2:
      node.Y_pred = 1
    return node

  #find the feature with max mutual information
  max_gain = 0
  feature_column, threshold  = 0 , 0
  for i in range(X_data.shape[1]):
    #if the feature is continous compare quartiles
    percentiles = np.linspace(0, 100, self.num_of_splits_per_feature + 2)[1:-1
    quantiles = set(np.percentile(X_data[:, i], percentiles))
    for quantile in quantiles:
      curr_gain = self.mutual_information(self.normalize_data(X_data[:, i], qu
      if curr_gain > max_gain:
        feature_column, threshold, max_gain = i, quantile, curr_gain

  #if the resulting gain is small we can end as leaf node
  if max_gain <= 0.05:
    node = TreeNode(None, None)
    node.isLeaf = True
    if np.sum(Y_data) >= subsample_size//2:
      node.Y_pred = 1
    return node

  #split the data accourding to max gain feature
  # Create a boolean mask based on the feature value
  zipped_data = np.hstack((X_data, Y_data.reshape([-1,1])))
  mask = X_data[:, feature_column] <= threshold
```

```
    # Split the data into two parts
    left_data = zipped_data[mask]
    right_data = zipped_data[~mask]

    node = TreeNode(feature_column=feature_column, threshold=threshold)
    node.left = self.build_tree(left_data[:, :-1], left_data[:, -1])
    node.right = self.build_tree(right_data[:, :-1], right_data[:, -1])
    return node

  def normalize_data(self, X, threshold):
    return [1 if X[i] <= threshold else 0 for i in range(len(X))]

  #4.2
  def mutual_information(self, X_data, Y_data):
    subsample_size = len(Y_data)
    p_x_1 = np.sum(X_data)/subsample_size
    entropy_x = self.entropy_helper(p_x_1) + self.entropy_helper(1-p_x_1)

    #conditional entropy
    count_x_1_y_1, count_x_0_y_1, count_x_0_y_0, count_x_1_y_0 = 0,0,0,0
    for i in range(subsample_size):
      if X_data[i] == 1 and Y_data[i] == 1:
        count_x_1_y_1 += 1
      elif X_data[i] == 0 and Y_data[i] == 1:
        count_x_0_y_1 += 1
      elif  X_data[i] == 0 and Y_data[i] == 0:
        count_x_0_y_0 == 1
      else:
        count_x_1_y_0 += 1

    count_y_1 = np.sum(Y_data)
    count_y_0 = len(Y_data) - count_y_1
    entropy_x_given_y = ((count_y_1*self.entropy_helper(count_x_1_y_1/count_y_1)
                      + count_y_1*self.entropy_helper(count_x_0_y_1/count_y_1))
                      + (count_y_0*self.entropy_helper(count_x_0_y_0/count_y_0)
                      + count_y_0*self.entropy_helper(count_x_1_y_0/count_y_0))

    entropy_x_given_y /= subsample_size
    #info gain
    return entropy_x - entropy_x_given_y

  def entropy_helper(self,prob):
    if prob == 0:
      return 0
    return prob*math.log2(1/prob)
```

```
class TreeNode:

  def __init__(self, feature_column, threshold):
    self.feature_column = feature_column
    self.threshold = threshold
    self.left = None
    self.right = None
    self.isLeaf = False
    self.Y_pred = 0
import graphviz as gv

# Define a graph object
class VisualizeGraph:
  def __init__(self, root, features):
    self.node_index = 1
    self.graph = gv.Digraph()
    self.graph.node(str(self.node_index), features[root.feature_column])
    self.node_index += 1
    self.features = features
    self.build_graph(root, 1)


# Add nodes
  def build_graph(self,node, parent_index):
    #left node
    if node.left.isLeaf:
      value = 'survived' if node.left.Y_pred == 1 else 'deceased'
      self.graph.node(str(self.node_index), value)
      self.graph.edge(str(parent_index), str(self.node_index), label=f'<={node.t
      self.node_index += 1
    else:
      self.graph.node(str(self.node_index), label = self.features[node.left.featu
      self.graph.edge( str(parent_index), str(self.node_index), label=f'<={node.
      self.node_index += 1
      self.build_graph(node.left, self.node_index-1)


    #right node
    if node.right.isLeaf:
      value = 'survived' if node.right.Y_pred == 1 else 'deceased'
      self.graph.node(str(self.node_index), value)
      self.graph.edge(str(parent_index), str(self.node_index), label=f'>{node.th
      self.node_index += 1
    else:
      self.graph.node(str(self.node_index), label = self.features[node.right.feat
      self.graph.edge(str(parent_index), str(self.node_index), label=f'>{node.th
```

```
      self.node_index += 1
      self.build_graph(node.right, self.node_index-1)

################################################
#section 4.5
#10 fold cross validation for decision tree
folds = 10
validation_ratio = folds/100
num_samples = len(df)
test_block_size = int(num_samples*validation_ratio)

accuracy = np.zeros(folds)
#for each fold find the accuracy
for i in range(folds):
  mask = df.index.isin(range(i*test_block_size, (i+1)*test_block_size))
  train_data, test_data = df.iloc[~mask, :].to_numpy(), df.iloc[mask, :].to_nump
  X_train, Y_train = train_data[:, 1:], train_data[:, :1]
  X_test, Y_test = test_data[:, 1:], test_data[:, :1]

  #build decision tree
  tree = DecisionTree(3)
  tree.fit_training_data(X_train, Y_train)

  #predict and find accuracy
  accuracy[i] = test(tree, X_test, Y_test)

mean_accuracy = np.mean(accuracy)
print(mean_accuracy)
#################################################
#section 4.6
X_new = np.array([2,
    0,
    25,
    0,
    0,
    20]).reshape([6,1])
node = decisionTree.root
while not node.isLeaf:
  if X_new[node.feature_column] <= node.threshold:
    node = node.left
  else:
    node = node.right
print(node.Y_pred)
####################################################
#section 4.7 a
#section 4.7 a
```

```
#random forest
def random_forest(df):

  descision_trees = []
  for i in range(5):
    #random sampling 80% of dataset
    df1 = df.sample(frac=0.8)
    X_rf_1, Y_rf_1 = df1.iloc[:,1:].to_numpy(), df1.iloc[:,0].to_numpy()

    #build decision tree
    decisionTree = DecisionTree(3)
    decisionTree.fit_training_data(X_rf_1, Y_rf_1)

    descision_trees.append(decisionTree)
    #save the decision tree as image
    # vg_graph = VisualizeGraph(decisionTree.root, features)
    # vg_graph.graph.render(f"graph_rf{i+1}.png")
  return descision_trees

random_forest(df)

#section 4.7 b
#section 4.7 b
#10 fold cross validation for decision tree

def predict(decisionTree, X_test):
  y_pred = []
  for i in range(X_test.shape[0]):
    node = decisionTree.root
    while not node.isLeaf:
      if X_test[i, node.feature_column] <= node.threshold:
        node = node.left
      else:
        node = node.right

    y_pred.append(node.Y_pred)

  #apredictions
  return np.array(y_pred)

folds = 10
validation_ratio = folds/100
num_samples = len(df)
test_block_size = int(num_samples*validation_ratio)

accuracy = np.zeros(folds)
```

```
#for each fold find the accuracy
for i in range(folds):
  mask = df.index.isin(range(i*test_block_size, (i+1)*test_block_size))
  train_data, test_data = df.iloc[~mask, :].to_numpy(), df.iloc[mask, :].to_nump
  X_train, Y_train = train_data[:, 1:], train_data[:, :1]
  X_test, Y_test = test_data[:, 1:], test_data[:, :1]

  #build random forests
  trees = random_forest(df.iloc[~mask, :])
  y_pred = np.zeros(test_block_size)

  for tree in trees:
    y_pred += predict(tree, X_test)

  pred_threshold = (1+len(trees))//2

  y_pred_cummulative = [1 if y_pred[i] >= pred_threshold else 0 for i in range(t
  #find accuracy
  correct_predictions = 0
  for j in range(test_block_size):
    if y_pred_cummulative[j] == Y_test[j]:
      correct_predictions += 1
  accuracy[i] = correct_predictions/test_block_size

mean_accuracy = np.mean(accuracy)
print(mean_accuracy)
#############################################
#section 4.7 c
for tree in trees:
  print(predict(tree, X_new.reshape([1,-1])))
ouput : [[0],[0],[0],[0],[0]]
############################################
#section 4.8 a
#random forest with excluding feature
def random_forest_with_leave_out(df):

  descision_trees = []
  for i in range(6):
    #dropping feature using feature map which containes features to index map
    df1 = df.drop(columns=features[i])
    feature_map = np.delete(features, i)
    X_rf_1,Y_rf_1 = df1.iloc[:,1:].to_numpy(), df1.iloc[:,0].to_numpy()

    #build decision tree
    decisionTree = DecisionTree(3)
    decisionTree.fit_training_data(X_rf_1, Y_rf_1)
```

```
    descision_trees.append(decisionTree)
    #save the decision tree as image
    # vg_graph = VisualizeGraph(decisionTree.root, feature_map)
    # vg_graph.graph.render(f"graph_rf_leave_out{i+1}.png")
  return descision_trees

random_forest_with_leave_out(df)
##############################################################
#section 4.8 b
#section 4.8 b

folds = 10
validation_ratio = folds/100
num_samples = len(df)
test_block_size = int(num_samples*validation_ratio)

accuracy = np.zeros(folds)
#for each fold find the accuracy
for i in range(folds):
  mask = df.index.isin(range(i*test_block_size, (i+1)*test_block_size))
  train_data, test_data = df.iloc[~mask, :].to_numpy(), df.iloc[mask, :].to_nump
  X_train, Y_train = train_data[:, 1:], train_data[:, :1]
  X_test, Y_test = test_data[:, 1:], test_data[:, :1]

  #build random forests
  trees = random_forest_with_leave_out(df.iloc[~mask, :])
  y_pred = np.zeros(test_block_size)

  for index,tree in enumerate(trees):
    #delete the column with index
    X_data_sliced = np.delete(X_test, index, axis=1)
    y_pred += predict(tree, X_data_sliced)

  pred_threshold = (1+len(trees))//2

  y_pred_cummulative = [1 if y_pred[i] >= pred_threshold else 0 for i in range(t
  #find accuracy
  correct_predictions = 0
  for j in range(test_block_size):
    if y_pred_cummulative[j] == Y_test[j]:
      correct_predictions += 1
  accuracy[i] = correct_predictions/test_block_size

mean_accuracy = np.mean(accuracy)
print(mean_accuracy)
```

```
##########################################################
#section 4.8 c
for tree in trees:
  print(predict(tree, X_new.reshape([1,-1])))
```