2.1 & 2.2:

The code was written in a way so that the random forest and decision tree can be implemented in the same code. Hence, for 2.1 and 2.2, I am showing the same answers. The difference is in line 248 and 250 of the code. Feature selection for random forest is in line 111 of the code, and 114 for decision tree.

```
# Run once to process titanic data
# Run only once

# playground for loading titanic
# method is quite manual. I will have a handmade array of indicators, indicator will be 1 for a
# column that is number, and 0 for a column that is categorical
# I will have a separate array, with values indicating whether the data in that class is discreet
# or continuous


import csv
import sys
import scipy.io
from numpy import genfromtxt
import numpy as np
import math
ArrDtype =np.array([1,1,0,1,1,1,0,1,0,0],dtype = 'int32')
ArrClass =np.array([1,1,1,0,0,0,0,0,0,1],dtype = 'int32') # 1 for class, 0 for continuous

DropFeature = [7,9]
```

```
path_train = 'datasets/titanic/titanic_training.csv'

FeatureSize = 10

DataSize = 1000


X = np.zeros((DataSize,FeatureSize-len(DropFeature)-1))# 1 is to account for labels

k =0

for i in range(0,FeatureSize):

    if ((i+1) in DropFeature):

        continue

    if (ArrDtype[i]==1):

#data = genfromtxt(path_train, delimiter=',', dtype=str,usecols=8,skip_header=1)

        data = genfromtxt(path_train, delimiter=',',usecols=i,skip_header=1)

        if(ArrClass[i]==1):

            MedianClass = np.nanmedian(np.array(data))

            for j in range(0,len(data)):

                if math.isnan(data[j]):

                    data[j] = MedianClass

        else:

            MeanClass = np.nanmean(np.array(data))

            for j in range(0,len(data)):

                if math.isnan(data[j]):

                    data[j] = MeanClass

    else:

        data = list(genfromtxt(path_train, delimiter=',', dtype=str,usecols=i,skip_header=1))
```

```
        U =np.unique(data,return_counts = False)
        for j in range(0,len(data)):
            if(data[j]==''):
                data[j]=U[-1]       #for j in range(0,len(data)):
            data[j] = ord(data[j][0])

    if(i>0):
        X[:,k] = np.array(data)
        k = k+1
    else:
        y = np.array(data, dtype='int32')
# doing the same for test
ArrDtype =np.array([1,0,1,1,1,0,1,0,0],dtype = 'int32')
ArrClass =np.array([1,1,0,0,0,0,0,0,1],dtype = 'int32') # 1 for class, 0 for continuous

DropFeature = [6,8]

path_train = 'datasets/titanic/titanic_testing_data.csv'
FeatureSize = 9
DataSize = 310
Z = np.zeros((DataSize,FeatureSize-len(DropFeature)))
k =0

for i in range(0,FeatureSize):
    if ((i+1) in DropFeature):
```

```
        continue
    if (ArrDtype[i]==1):
#data = genfromtxt(path_train, delimiter=',', dtype=str,usecols=8,skip_header=1)
        data = genfromtxt(path_train, delimiter=',',usecols=i,skip_header=1)
        if(ArrClass[i]==1):
            MedianClass = np.nanmedian(np.array(data))
            for j in range(0,len(data)):
                if math.isnan(data[j]):
                    data[j] = MedianClass
        else:
            MeanClass = np.nanmean(np.array(data))
            for j in range(0,len(data)):
                if math.isnan(data[j]):
                    data[j] = MeanClass
    else:
        data = list(genfromtxt(path_train, delimiter=',', dtype=str,usecols=i,skip_header=1))
        U =np.unique(data,return_counts = False)


        for j in range(0,len(data)):
            if(data[j]==''):
                data[j]=U[-1]       #for j in range(0,len(data)):
            data[j] = ord(data[j][0])




    Z[:,k] = np.array(data)
```

```
    k = k+1
scipy.io.savemat('TitanicProcessed.mat',{'X':X,'y':y,'Z':Z,'ArrClass':ArrClass})
```

```
# Code for tree and forest
"""
This is the starter code and some suggested architecture we provide you with.
But feel free to do any modifications as you wish or just completely ignore
all of them and have your own implementations.
"""
# some inspirations were taken from
# https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/

from collections import Counter

import numpy as np
from numpy import genfromtxt
import scipy.io
from scipy import stats

import random

np.random.seed(1242)
random.seed(1242)

class DecisionTree:

    def __init__(self,CostMethod,MaxDepth,Type,Features):
        self.CostMethod = CostMethod
```

```
        self.MaxDepth = MaxDepth
        self.Type = Type
        self.Features = Features
        """
        TODO: initialization of a decision tree
        """

    @staticmethod
    def entropy(y):
        """
        TODO: implement a method that calculates the entropy given all the labels
        """
        y = np.array(y)
        # assuming binary class
        if (len(y)==0):
            H = 0
        else:
            Pc1 = np.sum(y[y==1])/len(y)
            Pc0 = 1-Pc1
            if ((Pc1==0) |(Pc0==0)):
                H=0
            else:
                H = - (Pc1*np.log(Pc1)+Pc0*np.log(Pc0))
        return H
```

```
    @staticmethod
    def information_gain(X, y, thresh):
        """
        TODO: implement a method that calculates information gain given a vector of features
        and a split threshold
        """
        return 0

    @staticmethod
    def gini_impurity(y):
        """
        TODO: implement a method that calculates the gini impurity given all the labels
        """
        return 0

    @staticmethod
    def gini_purification(X, y, thresh):
        """
        TODO: implement a method that calculates reduction in impurity gain given a vector of
features
        and a split threshold
        """
        return 0

    def split(self,SplitVal,SplitIndex, Data):
```

```
        """
        TODO: implement a method that return a split of the dataset given an index of the
feature and
        a threshold for it
        """
        left,right = list(), list()
        for DataRow in Data:
            if (DataRow[SplitIndex]<=SplitVal):
                left.append(DataRow)
            else:
                right.append(DataRow)
        return left,right

    def segmenter(self, Node):
        """
        TODO: compute entropy gain for all single-dimension splits,
        return the feature and the threshold for the split that
        has maximum gain
        """
        X = np.array(Node['Data'])
        U,V = np.shape(X)
        #print(U,V)
        Left,Right = self.split(X[0,0],0,X)
        if (self.CostMethod =='Entropy'):
            Uleft = len(Left)
```

```
            if(Uleft ==0):
                Hleft =0
            else:
                Hleft = self.entropy(np.array(Left)[:,-1])
            Uright = len(Right)
            if(Uright ==0):
                Hright =0
            else:
                Hright = self.entropy(np.array(Right)[:,-1])
            Cost = (Uright*Hright+Uleft*Hleft)/U

baselineVal,baselineFeatureIndex,baselineCost,baselineLeft,baselineRight=X[0,0],0,Cost,Lef
t,Right
        if (self.Type=='Forest'):
            #print(V-1)
            FeatureNos = np.array(random.sample(range(0,V-1),int(np.sqrt(V-1))))
            #print(FeatureNos)
        else:
            FeatureNos = np.linspace(0,V-2,V-1,dtype='int32')
        for i in range(0,U):
            for j in range(0,len(FeatureNos)):
                Left,Right = self.split(X[i,FeatureNos[j]],FeatureNos[j],X)
                if (self.CostMethod =='Entropy'):
                    Uleft = len(Left)
                    if(Uleft ==0):
```

```
            Hleft =0
        else:
            Hleft = self.entropy(np.array(Left)[:,-1])
        Uright = len(Right)
        if(Uright ==0):
            Hright =0
        else:
            Hright = self.entropy(np.array(Right)[:,-1])
        Cost = (Uright*Hright+Uleft*Hleft)/U
    if (Cost<baselineCost):
        #print(X[i,FeatureNos[j]],FeatureNos[j])

baselineVal,baselineFeatureIndex,baselineCost,baselineLeft,baselineRight=X[i,FeatureNos[j]
                                                                      ],FeatureNos[j],Cost,Left,Right

Node['Left']   ={'Data':baselineLeft,
            'Label':'Node',
            'Threshold':'nan',
            'Index' : 'nan',
            'Left':'nan',
            'Right' : 'nan',
            'LeafLabel' : 'nan',
            }
Node['Right']  ={'Data':baselineRight,
            'Label':'Node',
```

```
                    'Threshold':'nan',
                    'Index' : 'nan',
                    'Left':'nan',
                    'Right' : 'nan',
                    'LeafLabel' : 'nan',
                   }
    Node['Threshold'] = baselineVal
    Node['Index'] = baselineFeatureIndex
    return
def leaf(self,X):
    X['Label'] = 'Leaf'
    y = np.array(X['Data'])[:,-1]
    if (np.count_nonzero(y==1)>np.count_nonzero(y==0)):
        X['LeafLabel']=1
    else:
        X['LeafLabel']=0
    return
def CreateBranch(self,X,DepthLevel):
    if((DepthLevel+1)>self.MaxDepth):
        self.leaf(X)
    elif((X['Left']!= 'nan') &(X['Right']!= 'nan') &((len(X['Left'])==0)|(len(X['Right'])==0))):
        self.leaf(X)
    else:
        self.segmenter(X)
```

```
        if(len(X['Left']['Data'])==0):  # see if there is no segmentation, if so then end the node
here
          self.leaf(X['Right'])
          X['Left']['Label'] = 'Leaf'
          X['Left']['LeafLabel'] = X['Right']['LeafLabel']
        elif(len(X['Right']['Data'])==0):  # see if there is no segmentation, if so then end the
node here
          self.leaf(X['Left'])
          X['Right']['Label'] = 'Leaf'
          X['Right']['LeafLabel'] = X['Left']['LeafLabel']
        else:
          self.CreateBranch(X['Left'],DepthLevel+1)
          self.CreateBranch(X['Right'],DepthLevel+1)
      return

    def fit(self, X):
      """
      TODO: fit the model to a training set. Think about what would be
      your stopping criteria
      """
      NodeLevel =0

      Root ={'Data':X,
          'Label':'Node',
          'Threshold':'nan',
```

```
            'Index' : 'nan',
            'Left':'nan',
            'Right' : 'nan',
            'LeafLabel' : 'nan',
            }
        #SplitVal,SplitIndex,LeftGroup,RightGroup = self.segmenter(X,NodeLevel)
        self.CreateBranch(Root,NodeLevel)

        #self.__repr__(Root,'\t') # if you want to visualize the tree

        return Root

    def predict(self, Node,X):
        """
        TODO: predict the labels for input data
        """
        if(Node['Label']=='Leaf'):
            Label = Node['LeafLabel']
        else:
            y = Node['Index']
            if (X[y]>Node['Threshold']):
                Label = self.predict(Node['Right'],X)
            else:
                Label = self.predict(Node['Left'],X)
        return Label
```

```
    def __repr__(self,Node,tab):
        """
        TODO: one way to visualize the decision tree is to write out a __repr__ method
        that returns the string representation of a tree. Think about how to visualize
        a tree structure. You might have seen this before in CS61A.
        """
        if(Node['Label']=='Node'):
            print(tab,'if X[',self.Features[Node['Index']],']>',Node['Threshold'])
            self.__repr__(Node['Right'],tab+'\t')
            print(tab,'else')
            self.__repr__(Node['Left'],tab+'\t')
        else:
            print(tab,'y=',Node['LeafLabel'])
        return 0


def DataSplit(data, labels, val_size):
    num_items = len(data)
    assert num_items == len(labels)
    assert val_size >= 0
    if val_size < 1.0:
        val_size = int(num_items * val_size)
    train_size = num_items - val_size
    idx = np.random.permutation(num_items)
    data_train1 = data[idx][:train_size]
```

```
    label_train = labels[idx][:train_size]

    data_val = data[idx][train_size:]

    label_val = labels[idx][train_size:]

    return data_train1, data_val, label_train, label_val

if __name__ == "__main__":

    dataset = "spam"

    Bagging = 'Off'

    Validation = 'On' # if off that means we are doing the test data

    ClassifierType = 'Tree'

    DataFileName = 'SPAMDTTestV1.mat'


    #FeatureName = []

    if dataset == "titanic":

        # Load titanic data

        data1 =scipy.io.loadmat('TitanicProcessed.mat')

        X = data1["X"]

        y = data1["y"]

        Z = data1["Z"]

        y = y.reshape(np.size(y),1)

        FeatureName =['pclass','sex','age','sibsp','parch','fare','embarked']# for titanic

        # TODO: preprocess titanic dataset

        # Notes:

        # 1. Some data points are missing their labels

        # 2. Some features are not numerical but categorical

        # 3. Some values are missing for some features
```

```
elif dataset == "spam":
    features = [
        "pain", "private", "bank", "money", "drug", "spam", "prescription",
        "creative", "height", "featured", "differ", "width", "other",
        "energy", "business", "message", "volumes", "revision", "path",
        "meter", "memo", "planning", "pleased", "record", "out",
        "semicolon", "dollar", "sharp", "exclamation", "parenthesis",
        "square_bracket", "ampersand"
    ]
    assert len(features) == 32
    FeatureName = features
    # Load spam data
    path_train = 'datasets/spam-dataset/spam_data.mat'
    data = scipy.io.loadmat(path_train)
    X = data['training_data']
    y = np.squeeze(data['training_labels'])
    Z = data['test_data']
    class_names = ["Ham", "Spam"]

else:
    raise NotImplementedError("Dataset %s not handled" % dataset)
# splitting data into training and validation
if (Validation == 'On'):
    ValidationSize = int(len(y)*0.20)
```

```
        TrainingData,ValidationData,TrainingLabel,ValidationLabel =
DataSplit(X,y,ValidationSize)
        # converting the data into array for easier preprocessing
        TrainingData = np.array(TrainingData)
        TrainingLabel = np.array(TrainingLabel)
        ValidationData = np.array(ValidationData)
        ValidationLabel = np.array(ValidationLabel)
    else:
        TrainingData = np.array(X)
        TrainingLabel = np.array(y)
        ValidationData = np.array(Z)
        ValidationSize = len(Z)
    #print(np.shape(TrainingData))
    TrainingData =
np.concatenate((TrainingData,TrainingLabel.reshape(len(TrainingLabel),1)),axis=1)
    #print(np.shape(TrainingData))
    #return
    #creating an instance of the classifier
    if (Bagging=='Off'):

        if (Validation == 'On'):
            MaxDepth = np.linspace(1,1,1)
            ValidationError = np.zeros((len(MaxDepth),1),dtype ='float64')
            TrainingError = np.zeros((len(MaxDepth),1),dtype ='float64')
        else:
```

```
        MaxDepth = np.linspace(1,1,1)


    for j in range(0,len(MaxDepth)):
        classifier = DecisionTree('Entropy',MaxDepth[j],ClassifierType,FeatureName)


        Root= classifier.fit(TrainingData)
        ValidationPredicted = np.zeros((ValidationSize,1),dtype ='int64')
        TrainingPredicted = np.zeros((len(TrainingLabel),1),dtype ='int64')
        for i in range(0,len(ValidationData)):
            ValidationPredicted[i] = classifier.predict(Root,ValidationData[i,:])
        for i in range(0,len(TrainingLabel)):
            TrainingPredicted[i] = classifier.predict(Root,TrainingData[i,:])
        if (Validation=='On'):
            ValidationError[j] = np.sum(np.abs((ValidationLabel.reshape(ValidationSize,1)
                                    -ValidationPredicted.reshape(ValidationSize,1))
                                   .reshape(ValidationSize,1)))/ValidationSize
            TrainingError[j] = np.sum(np.abs((TrainingLabel.reshape(len(TrainingLabel),1)
                                  -TrainingPredicted.reshape(len(TrainingLabel),1))
                                 .reshape(len(TrainingLabel),1)))/len(TrainingLabel)



scipy.io.savemat(DataFileName,{'ValidationError':ValidationError,'TrainingError':TrainingE
rror,'Tree Depth':MaxDepth})
        else:
            flat_list = [item for sublist in ValidationPredicted for item in sublist]
```

```
            scipy.io.savemat(DataFileName,{'Tree':Root,'Prediction':flat_list})
            import save_csv
            save_csv.results_to_csv(flat_list)
    else:
        BagNos = 1
        BagSize = int(1*len(TrainingData))
        if (Validation == 'On'):
            MaxDepth = np.linspace(1,1,1)
            ValidationError = np.zeros((len(MaxDepth),1),dtype ='float64')
            TrainingError = np.zeros((len(MaxDepth),1),dtype ='float64')
        else:
            MaxDepth = np.linspace(1,1,1)
        for j in range(0,len(MaxDepth)):
            print('Depth',MaxDepth[j])
            ValidationPredicted = np.zeros((ValidationSize,BagNos),dtype ='float64')
            TrainingPredicted = np.zeros((len(TrainingLabel),BagNos),dtype ='float64')
            for k in range(0,BagNos):
                print('Bag',k)
                np.random.seed(k)
                BagVariable=np.random.randint(BagSize,size=BagSize)
                TrainingBagData = TrainingData[BagVariable,:]
                classifier = DecisionTree('Entropy',MaxDepth[j],ClassifierType,FeatureName) #
type either Forest or Tree
                #import time
                #start = time.time()
```

```
        #Level = 0
        Root= classifier.fit(TrainingBagData)
        #ValidationPredicted = np.zeros((ValidationSize,1),dtype ='int64')
        for i in range(0,ValidationSize):
            #print(i,k)
            ValidationPredicted[i,k] = classifier.predict(Root,ValidationData[i,:])
        for i in range(0,len(TrainingLabel)):
            TrainingPredicted[i] = classifier.predict(Root,TrainingData[i,:])
    ValidationPredicted =
np.round(np.sum(ValidationPredicted,axis=1)/BagNos,decimals =0)
    TrainingPredicted = np.round(np.sum(TrainingPredicted,axis=1)/BagNos,decimals
=0)
    if (Validation=='On'):
        ValidationError[j] = np.sum(np.abs((ValidationLabel.reshape(ValidationSize,1)
                                    -ValidationPredicted.reshape(ValidationSize,1))
                                    .reshape(ValidationSize,1)))/ValidationSize
        TrainingError[j] = np.sum(np.abs((TrainingLabel.reshape(len(TrainingLabel),1)
                              -TrainingPredicted.reshape(len(TrainingLabel),1))
                              .reshape(len(TrainingLabel),1)))/len(TrainingLabel)


scipy.io.savemat(DataFileName,{'ValidationError':ValidationError,'TrainingError':TrainingE
rror,'Tree Depth':MaxDepth,
                                       'BagSize':BagSize,'BagNos':BagNos})
```

```
    else:
        ValidationPredicted.flatten
        scipy.io.savemat(DataFileName,{'Tree':Root,'Prediction':ValidationPredicted})
        import save_csv
        save_csv.results_to_csv(ValidationPredicted)


"""
TODO: train decision tree/random forest on different datasets and perform the tasks
in the problem
"""
```