```
from decision_tree import DecisionTree
import csv
import numpy as np  # http://www.numpy.org
import ast
from datetime import datetime

# This starter code does not run. You will have to add your changes and
# turn in code that runs properly.

"""
Here,
1. X is assumed to be a matrix with n rows and d columns where n is the
number of total records and d is the number of features of each record.
2. y is assumed to be a vector of labels of length n.
3. XX is similar to X, except that XX also contains the data label for each
record.
"""

"""
This skeleton is provided to help you implement the assignment.You must
implement the existing functions as necessary. You may add new functions
as long as they are called from within the given classes.

VERY IMPORTANT!
Do NOT change the signature of the given functions.
Do NOT change any part of the main function APART from the forest_size
parameter.
"""


class RandomForest(object):
    num_trees = 0
    decision_trees = []

    # the bootstrapping datasets for trees
    # bootstraps_datasets is a list of lists, where each list in
bootstraps_datasets is a bootstrapped dataset.
    bootstraps_datasets = []

    # the true class labels, corresponding to records in the bootstrapping
datasets
    # bootstraps_labels is a list of lists, where the 'i'th list contains the
labels corresponding to records in
    # the 'i'th bootstrapped dataset.
    bootstraps_labels = []

    def __init__(self, num_trees):
        # Initialization done here
        self.num_trees = num_trees
        self.decision_trees = [DecisionTree(max_depth=10) for i in
range(num_trees)]

    def _bootstrapping(self, XX, n):
        # Reference: https://en.wikipedia.org/wiki/Bootstrapping_(statistics)
        #
        # TODO: Create a sample dataset of size n by sampling with replacement
        #       from the original dataset XX.
        # Note that you would also need to record the corresponding class labels
        # for the sampled records for training purposes.

        samples = [] # sampled dataset
        labels = []  # class labels for the sampled records
        ### Implement your code here
        ################################################
```

```
        for i in range(n):
            index = np.random.randint(len(XX))
            row = XX[index]
            samples.append(row[:-1])
            labels.append(row[-1])
        #pass
        ##############################################
        return (samples, labels)

    def bootstrapping(self, XX):
        # Initializing the bootstap datasets for each tree
        for i in range(self.num_trees):
            data_sample, data_label = self._bootstrapping(XX, len(XX))
            self.bootstraps_datasets.append(data_sample)
            self.bootstraps_labels.append(data_label)

    def fitting(self):
        # TODO: Train `num_trees` decision trees using the bootstraps datasets
        # and labels by calling the learn function from your DecisionTree class.
        ### Implement your code here
        ##############################################
        for i in range(self.num_trees):
            self.decision_trees[i].learn(self.bootstraps_datasets[i],
self.bootstraps_labels[i])
        return(self.decision_trees[0].tree)
        #pass
        ##############################################

    def voting(self, X):
        y = []

        for record in X:
            # Following steps have been performed here:
            #   1. Find the set of trees that consider the record as an
            #      out-of-bag sample.
            #   2. Predict the label using each of the above found trees.
            #   3. Use majority vote to find the final label for this recod.
            votes = []

            for i in range(len(self.bootstraps_datasets)):
                dataset = self.bootstraps_datasets[i]

                if record not in dataset:
                    OOB_tree = self.decision_trees[i]
                    effective_vote = OOB_tree.classify(record)
                    votes.append(effective_vote)

            counts = np.bincount(votes)

            if len(counts) == 0:
                # TODO: Special case
                #  Handle the case where the record is not an out-of-bag sample
                #  for any of the trees.
                ### Implement your code here
                ##############################################
                i = self.bootstraps_datasets[0].index(record)
                y = np.append(y, self.bootstraps_labels[0][i])
                #pass
                ##############################################
            else:
                y = np.append(y, np.argmax(counts))

        return y
```

```
    def user(self):
        """
        :return: string
        your GTUsername, NOT your 9-Digit GTId
        """
        ### Implement your code here
        ############################################
        return ''
        ############################################



# DO NOT change the main function apart from the forest_size parameter!
def main():

    # start time
    start = datetime.now()

    X = list()
    y = list()
    XX = list()  # Contains data features and data labels
    numerical_cols = set([i for i in range(0, 10)])  # indices of numeric
attributes (columns)

    # Loading data set
    print("reading the data")
    with open("Churn.csv") as f:
        next(f, None)
        for line in csv.reader(f, delimiter=","):
            xline = []
            for i in range(len(line)):
                if i in numerical_cols:
                    xline.append(ast.literal_eval(line[i]))
                else:
                    xline.append(line[i])

            X.append(xline[:-1])
            y.append(xline[-1])
            XX.append(xline[:])

    # TODO: Initialize according to your implementation
    # VERY IMPORTANT: Minimum forest_size should be 10
    forest_size = 10

    # Initializing a random forest.
    randomForest = RandomForest(forest_size)

    # printing the name
    print("__Name: " + randomForest.user()+"__")

    # Creating the bootstrapping datasets
    print("creating the bootstrap datasets")
    randomForest.bootstrapping(XX)

    # Building trees in the forest
    print("fitting the forest")
    randomForest.fitting()

    # Calculating an unbiased error estimation of the random forest
    # based on out-of-bag (OOB) error estimate.
    y_predicted = randomForest.voting(X)

    # Comparing predicted and true labels
    results = [prediction == truth for prediction, truth in zip(y_predicted, y)]
```

```
    # Accuracy
    accuracy = float(results.count(True)) / float(len(results))

    print("accuracy: %.4f" % accuracy)
    print("OOB estimate: %.4f" % (1 - accuracy))

    # end time
    print("Execution time: " + str(datetime.now() - start))


if __name__ == "__main__":
    main()
```