```
#decision trees

import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

def importdata():
  balance_data = pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-'+
'databases/balance-scale/balance-scale.data',
  sep= ',', header = None)

  print ("Dataset Length: ", len(balance_data))
  print ("Dataset Shape: ", balance_data.shape)

  print ("Dataset: ",balance_data.head())
  return balance_data

def splitdataset(balance_data):

  X = balance_data.values[:, 1:5]
  Y = balance_data.values[:, 0]

  X_train, X_test, y_train, y_test = train_test_split(
  X, Y, test_size = 0.3, random_state = 100)

  return X, Y, X_train, X_test, y_train, y_test

def train_using_gini(X_train, X_test, y_train):

  clf_gini = DecisionTreeClassifier(criterion = "gini",
      random_state = 100,max_depth=3, min_samples_leaf=5)

  clf_gini.fit(X_train, y_train)
  return clf_gini

def tarin_using_entropy(X_train, X_test, y_train):

  clf_entropy = DecisionTreeClassifier(
      criterion = "entropy", random_state = 100,
      max_depth = 3, min_samples_leaf = 5)

  clf_entropy.fit(X_train, y_train)
  return clf_entropy
```

```
def prediction(X_test, clf_object):

  y_pred = clf_object.predict(X_test)
  print("Predicted values:")
  print(y_pred)
  return y_pred

def cal_accuracy(y_test, y_pred):

  print("Confusion Matrix: ",
    confusion_matrix(y_test, y_pred))

  print ("Accuracy : ",
  accuracy_score(y_test,y_pred)*100)

  print("Report : ",
  classification_report(y_test, y_pred))

def main():

  data = importdata()
  X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
  clf_gini = train_using_gini(X_train, X_test, y_train)
  clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

  print("Results Using Gini Index:")

  y_pred_gini = prediction(X_test, clf_gini)
  cal_accuracy(y_test, y_pred_gini)

  print("Results Using Entropy:")
  y_pred_entropy = prediction(X_test, clf_entropy)
  cal_accuracy(y_test, y_pred_entropy)


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

```
   Dataset Length:  625
   Dataset Shape:  (625, 5)
   Dataset:     0  1  2  3  4
   0  B  1  1  1  1
   1  R  1  1  1  2
   2  R  1  1  1  3
   3  R  1  1  1  4
   4  R  1  1  1  5
   Results Using Gini Index:
   Predicted values:
   ['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
    'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
    'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
```

```
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 67 18]
 [ 0 19 71]]
Accuracy :  73.40425531914893
Report :                precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.73      0.79      0.76        85
           R       0.74      0.79      0.76        90

    accuracy                           0.73       188
   macro avg       0.49      0.53      0.51       188
weighted avg       0.68      0.73      0.71       188

Results Using Entropy:
Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 63 22]
 [ 0 20 70]]
Accuracy :  70.74468085106383
Report :                precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.71      0.74      0.72        85
```

```
#naive bayes classifier

import math
import random
import csv


def encode_class(mydata):
    classes = []
    for i in range(len(mydata)):
```

```
        if mydata[i][-1] not in classes:
            classes.append(mydata[i][-1])
    for i in range(len(classes)):
        for j in range(len(mydata)):
            if mydata[j][-1] == classes[i]:
                mydata[j][-1] = i
    return mydata


def splitting(mydata, ratio):
    train_num = int(len(mydata) * ratio)
    train = []
    test = list(mydata)
    while len(train) < train_num:
        index = random.randrange(len(test))
        train.append(test.pop(index))
    return train, test


def groupUnderClass(mydata):
    dict = {}
    for i in range(len(mydata)):
        if (mydata[i][-1] not in dict):
            dict[mydata[i][-1]] = []
        dict[mydata[i][-1]].append(mydata[i])
    return dict


def mean(numbers):
    return sum(numbers) / float(len(numbers))

def std_dev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)
    return math.sqrt(variance)

def MeanAndStdDev(mydata):
    info = [(mean(attribute), std_dev(attribute)) for attribute in zip(*mydata)]
    del info[-1]
    return info

def MeanAndStdDevForClass(mydata):
    info = {}
    dict = groupUnderClass(mydata)
    for classValue, instances in dict.items():
        info[classValue] = MeanAndStdDev(instances)
    return info

def calculateGaussianProbability(x, mean, stdev):
    expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * expo
```

```
def calculateClassProbabilities(info, test):
    probabilities = {}
    for classValue, classSummaries in info.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, std_dev = classSummaries[i]
            x = test[i]
            probabilities[classValue] *= calculateGaussianProbability(x, mean, std_dev)
    return probabilities

def predict(info, test):
    probabilities = calculateClassProbabilities(info, test)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel


def getPredictions(info, test):
    predictions = []
    for i in range(len(test)):
        result = predict(info, test[i])
        predictions.append(result)
    return predictions

def accuracy_rate(test, predictions):
    correct = 0
    for i in range(len(test)):
        if test[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(test))) * 100.0

filename = 'mydata.csv'


mydata = csv.reader(open(filename, "rt"))
mydata = list(mydata)
mydata = encode_class(mydata)
for i in range(len(mydata)):
    mydata[i] = [float(x) for x in mydata[i]]


ratio = 0.7
train_data, test_data = splitting(mydata, ratio)
print('Total number of examples are: ', len(mydata))
print('Out of these, training examples are: ', len(train_data))
print("Test examples are: ", len(test_data))
```

```
info = MeanAndStdDevForClass(train_data)

predictions = getPredictions(info, test_data)
accuracy = accuracy_rate(test_data, predictions)
print("Accuracy of your model is: ", accuracy)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-3-97d2185f03fd> in <module>
    125
    126 # load the file and store it in mydata list
--> 127 mydata = csv.reader(open(filename, "rt"))
    128 mydata = list(mydata)
    129 mydata = encode_class(mydata)

FileNotFoundError: [Errno 2] No such file or directory: 'E:\\user\\MACHINE
LEARNING\\machine learning algos\\Naive bayes\\filedata.csv'
```

SEARCH STACK OVERFLOW

Double-click (or enter) to edit