import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score,precision\_score,f1\_score,recall\_score

class NaiveBayes():

def \_\_init\_\_(self):

self.class\_prob={}

self.features\_prob={}

def fit(self,x,Y):

#calculate frequency of each

for value in Y:

if value in self.class\_prob.keys():

self.class\_prob[value] += 1

else:

self.class\_prob[value] = 1

counts = list(self.class\_prob.values()) ##counts

total\_samples = len(Y)

#calculate probability

for key in self.class\_prob:

self.class\_prob[key]=self.class\_prob[key]/total\_samples

#probabioty of feature given class

for c in self.class\_prob.keys():

self.features\_prob[c] = {} #dict inside dict for features

for feature in x.columns:

self.features\_prob[c][feature] = {} #one more dict inside to store

unique\_values = x[feature].unique() #unique values of that feature

for value in unique\_values:

count = np.sum((x[feature] == value) & (Y == c))

self.features\_prob[c][feature][value] = count / counts[c]

# for feature in x.columns:

# self.features\_prob[c][feature] = {} # one more dict inside to store

# unique\_values = [] # to store unique values of that feature

# for value in x[feature]:

# if value not in unique\_values:

# unique\_values.append(value)

# for value in unique\_values:

# count = 0

# for i in range(len(x[feature])):

# if x[feature][i] == value and Y[i] == c:

# count += 1

# self.features\_prob[c][feature][value] = count / counts[c]

def predict(self, x):

predictions = []

for i in range(len(x)):

row = x.iloc[i]

max\_prob = -1

predicted\_class = None

for c in self.class\_prob:

prob = self.class\_prob[c]

for feature in x.columns:

value = row[feature]

if value in self.features\_prob[c][feature]:

prob \*= self.features\_prob[c][feature][value]

else:

prob \*= 0

if prob > max\_prob:

max\_prob = prob

predicted\_class = c

predictions.append(predicted\_class)

return predictions

data = pd.read\_csv("Social\_Network\_Ads.csv")

data["Gender"] = np.where(data["Gender"] == 'Male', 1, 0)

X = data.iloc[:,1:4]

y = data['Purchased']

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,y,test\_size=0.2,random\_state=0)

data

model=NaiveBayes()

model.fit(X\_train,Y\_train)

Y\_pred=model.predict(X\_test)

len(Y\_pred) , len(Y\_test)

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

accuracy = accuracy\_score(Y\_pred, Y\_test)

precision = precision\_score(Y\_pred, Y\_test)

recall = recall\_score(Y\_pred, Y\_test)

f1 = f1\_score(Y\_pred, Y\_test)

print("Validation Set Metrics:")

print("Accuracy: {:.2f}".format(accuracy))

print("Precision: {:.2f}".format(precision))

print("Recall: {:.2f}".format(recall))

print("F1 Score: {:.2f}".format(f1))

confusion = confusion\_matrix(Y\_pred, Y\_test)

print(confusion)

valid = data.sample(n=20)

X\_valid = valid.iloc[:,1:4]

y\_valid = valid['Purchased']

# X\_valid

y\_val = model.predict(X\_valid)

y\_val

y\_valid = y\_valid

y\_val, y\_valid

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

accuracy = accuracy\_score(y\_val, y\_valid)

precision = precision\_score(y\_val, y\_valid)

recall = recall\_score(y\_val, y\_valid)

f1 = f1\_score(y\_val, y\_valid)

print("Validation Set Metrics:")

print("Accuracy: {:.2f}".format(accuracy))

print("Precision: {:.2f}".format(precision))

print("Recall: {:.2f}".format(recall))

print("F1 Score: {:.2f}".format(f1))

confusion = confusion\_matrix(y\_val, y\_valid)

print(confusion)

print("Class 0 predicted and true : ")

print(confusion[0][0])

print("Class 0 predicted and false : ")

print(confusion[0][1])

print("Class 1 predicted and false : ")

print(confusion[1][0])

print("Class 1 predicted and true : ")

print(confusion[1][1])

a = pd.DataFrame()

a["Gender"] = ["Male"]

a["Age"] = [1]

a["EstimatedSalary"] = [00]

model.predict(a)