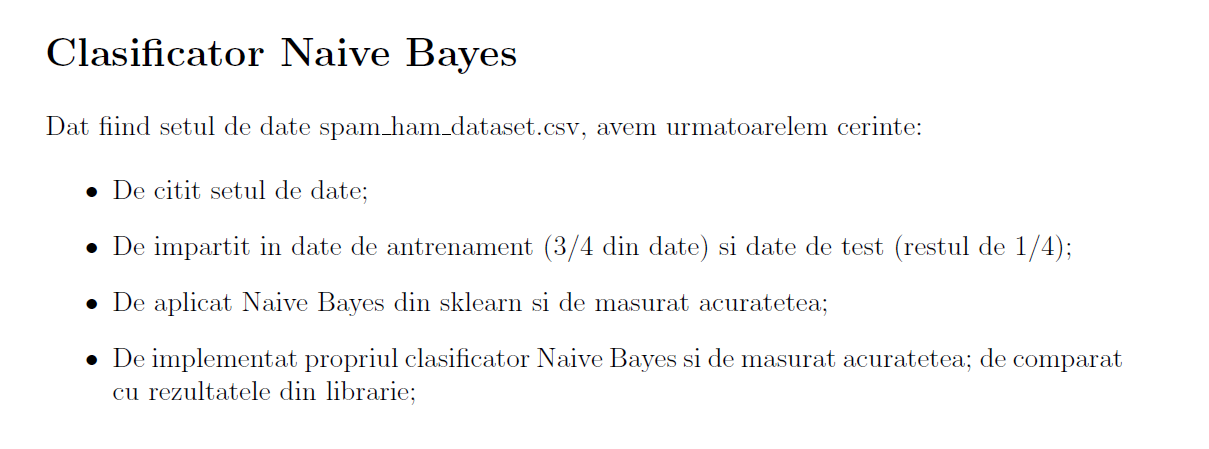
**Tema 4**



Continutul fisierului **tema4.py** unde am aplicat Naive Bayes preimplementat si am comparat cu Naive Bayes-ul implementat de mine (care se afla in fisierul my\_naive\_bayes.py) este:

import pandas as pd

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

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import GaussianNB

from my\_naive\_bayes import MyNaiveBayes

from sklearn.metrics import accuracy\_score

Data = pd.read\_csv("spam\_ham\_dataset.csv")

# convert text to vector

count\_vectorizer = CountVectorizer()

X = count\_vectorizer.fit\_transform(Data["text"]).toarray()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Data["label\_num"].tolist(), test\_size=0.25, random\_state=109)

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

y\_pred = gnb.predict(X\_test)

print("Accuracy of implemented Naive Bayes:", accuracy\_score(y\_test, y\_pred))

# compare it with my implementation

mnb = MyNaiveBayes()

y\_pred = mnb.implemented\_classifier(X\_train, y\_train, X\_test)

print("Accuracy of my implementation for Naive Bayes:", accuracy\_score(y\_test, y\_pred))

Continutul fisierului **my\_naive\_bayes.py** care contine implementarea mea pentru Naive Bayes este:

import math

class MyNaiveBayes:

def mean(self, nums):

return sum(nums) / float(len(nums))

def std\_dev(self, nums):

m = self.mean(nums)

sigma = math.sqrt(sum([pow(x - m, 2) for x in nums]) / float(len(nums) - 1))

return sigma

def mean\_and\_std\_dev(self, data):

info = [(self.mean(attr), self.std\_dev(attr)) for attr in zip(\*data) if self.mean(attr) != 0 and self.std\_dev(attr)]

return info

def separate\_by\_class(self, X\_train, y\_train):

dict = {}

for i in range(len(X\_train)):

if y\_train[i] not in dict:

dict[y\_train[i]] = []

dict[y\_train[i]].append(X\_train[i])

return dict

def mean\_and\_std\_dev\_for\_class(self, X\_train, y\_train):

info = {}

dict = self.separate\_by\_class(X\_train, y\_train)

for class\_val, instances in dict.items():

info[class\_val] = self.mean\_and\_std\_dev(instances)

return info

def calculate\_gaussian\_probability(self, 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 calculate\_class\_probabilities(self, info, test):

probabilities = {}

for class\_val, class\_summaries in info.items():

probabilities[class\_val] = 1

for i in range(len(class\_summaries)):

mean, std\_dev = class\_summaries[i]

x = test[i]

probabilities[class\_val] \*= self.calculate\_gaussian\_probability(x, mean, std\_dev)

return probabilities

def predict(self, info, test):

probabilities = self.calculate\_class\_probabilities(info, test)

best\_label, best\_prob = None, -1

for class\_val, probability in probabilities.items():

if best\_label is None or probability > best\_prob:

best\_prob = probability

best\_label = class\_val

return best\_label

def get\_predictions(self, info, test):

predictions = []

for i in range(len(test)):

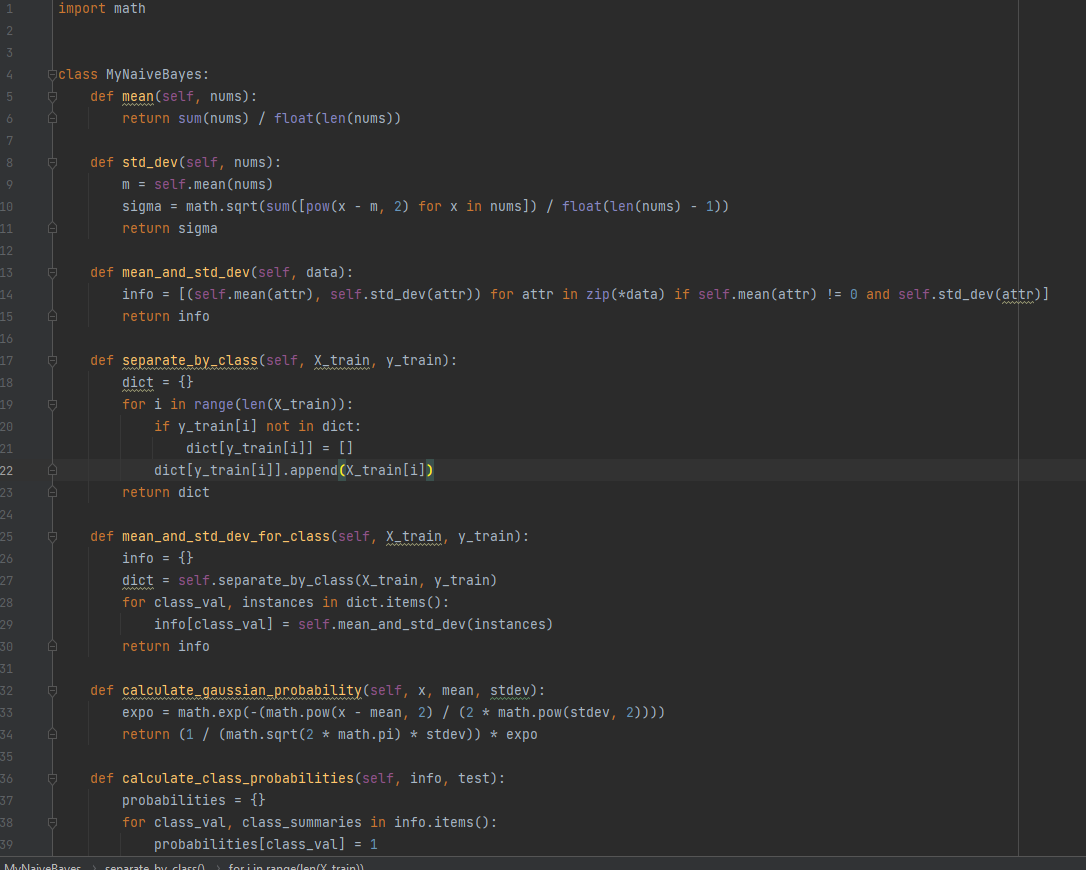
result = self.predict(info, test[i])

predictions.append(result)

return predictions

def implemented\_classifier(self, X\_train, y\_train, X\_test):

info = self.mean\_and\_std\_dev\_for\_class(X\_train, y\_train)

 return self.get\_predictions(info, X\_test)

Cand rulez fisierul **tema4.py** se vor afisa urmatoarele rezultate:

