# Tema 4

## Clasificator Naive Bayes

Dat fiind setul de date spam_ham_dataset.csv, avem urmatoarelem cerinte:

- De citit setul de date;
- De impartit in date de antrenament (3/4 din date) si date de test (restul de 1/4);
- De aplicat Naive Bayes din sklearn si de masurat acuratetea;
- De implementat propriul clasificator Naive Bayes si de masurat acuratetea; de comparat cu rezultatele din librarie;

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)
```

```python
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
```

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

```python
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))
```

```
C:\Users\Ana\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\Ana\Desktop\FMI\Year II\sem 1\Probabilitati si statistica\laborator\Teme\tema4.py"
Accuracy of implemented Naive Bayes: 0.9605568445475638
Accuracy of my implementation for Naive Bayes: 0.7107501933488012

Process finished with exit code 0
```


```
C:\Users\Ana\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\Ana\Desktop\FMI\Year II\sem 1\Probabilitati si statistica\laborator\Teme\tema4.py"
Accuracy of implemented Naive Bayes: 0.9605568445475638
Accuracy of my implementation for Naive Bayes: 0.7107501933488012

Process finished with exit code 0
```