**Лабораторна робота №1**

**ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ**

**Мета роботи:** використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити попередню обробку та класифікацію даних.

Хід роботи:

**Завдання 2.1** Попередня обробка даних

Лістинг коду:

import numpy as np

from sklearn import preprocessing

input\_data = np.array([[5.1, -2.9, 3.3],

 [-1.2, 7.8, -6.1],

 [3.9, 0.4, 2.1],

[7.3, -9.9, -4.5]])

# Бінаризація даних

data\_binarized = preprocessing.Binarizer(threshold=2.1).transform(input\_data)

print("\n Binarized data:\n", data\_binarized)

# Виведення середнього значення та стандартного відхилення

print("\nBEFORE: ")

print("Mean =", input\_data.mean(axis=0))

print("Std deviation =", input\_data.std(axis=0))

# Исключение среднего

data\_scaled = preprocessing.scale(input\_data)

print("\nAFTER: ")

print("Mean =", data\_scaled.mean(axis=0))

print("Std deviation =", data\_scaled.std(axis=0))

# Масштабування MinМax

data\_scaler\_minmax = preprocessing.MinMaxScaler(feature\_range=(0, 1))

data\_scaled\_minmax = data\_scaler\_minmax.fit\_transform(input\_data)

print("\nМin max scaled data:\n", data\_scaled\_minmax)

# Нормалізація даних

data\_normalized\_l1 = preprocessing.normalize(input\_data, norm='l1')

data\_normalized\_l2 = preprocessing.normalize(input\_data, norm='l2')

print("\nl1 normalized data:\n", data\_normalized\_l1)

print("\nl2 normalized data:\n", data\_normalized\_l2)

Результат виконання:

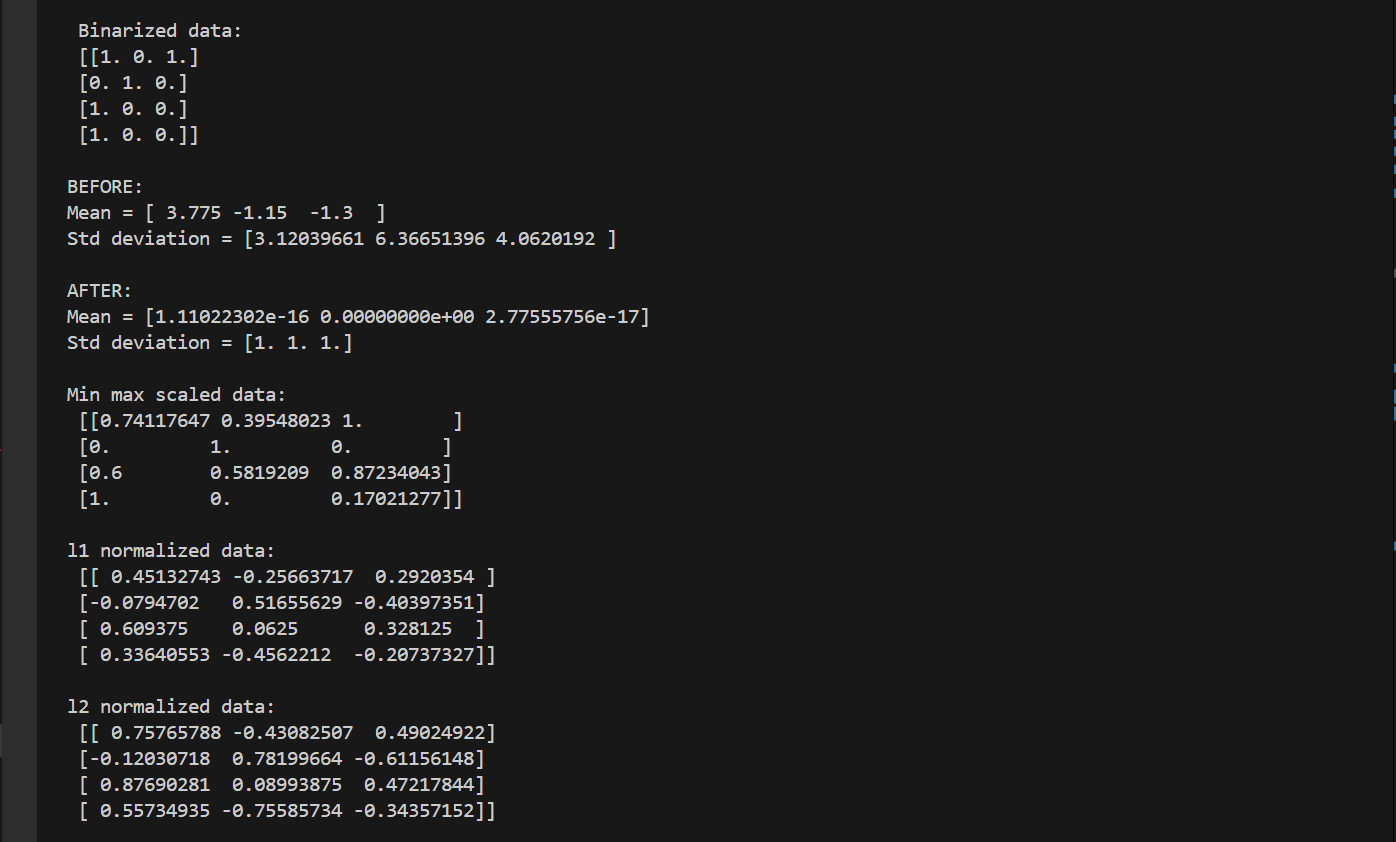


Рисунок 1- Результат виконання програми

L-1-нормалізація відрізняється від L-2-нормалізація природою її обчислення. Для обчислення **L-1-нормалізації** використовується метод найменших відхилень (Least Absolute Deviations), що забезпечує рівність 1 суми абсолютних значень в кожному ряду, в свою чергу розрахунок **L-2-нормалізації** здійснюється з використанням методу найменших квадратів, що забезпечує рівність 1 суми квадратів значень.

Лістинг коду:

#КОДУВАННЯ МІТОК

import numpy as np

from sklearn import preprocessing

# Надання позначок вхідних даних

Input\_labels = ['red', 'Ыасk', 'red', 'green', 'Ьlack', 'yellow', 'white']

# Створення кодувальника та встановлення відповідності

# між мітками та числами

encoder = preprocessing.LabelEncoder()

encoder.fit(Input\_labels)

# Виведення відображення

print("\nLabel mapping:")

for i, item in enumerate(encoder.classes\_):

    print(item, '-->', i)

# перетворення міток за допомогою кодувальника

test\_labels = ['green', 'red', 'Ыасk']

encoded\_values = encoder.transform(test\_labels )

print("\nLabels =", test\_labels )

print("Encoded values =", list (encoded\_values ) )

# Декодування набору чисел за допомогою декодера

encoded\_values = [3, 0, 4, 1]

decoded\_list = encoder.inverse\_transform(encoded\_values)

print("\nEncoded values =", encoded\_values)

print("Decoded labels =", list (decoded\_list ) )

Результат виконання:

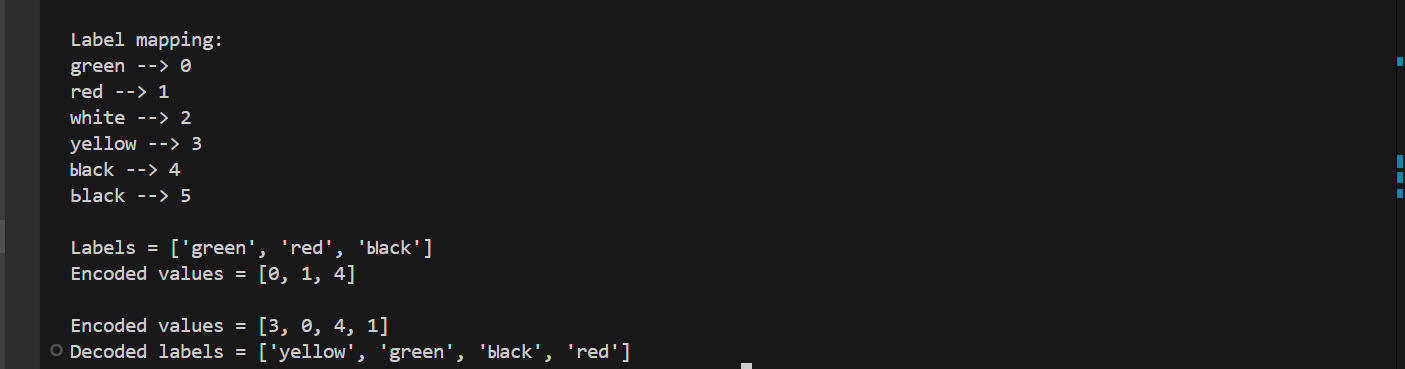


Рисунок 2- Результат виконання програми

**Завдання 2.2** Попередня обробка нових даних

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № варіанту | Значення змінної input\_data | | | | | | | | | | | | Поріг бінар изації |
| 7 | 1.3 | 3.9 | 6.2 | 4.9 | 2.2 | -4.3 | -2.2 | 6.5 | 4.1 | -5.2 | -3.4 | -5.2 | 2.0 |

Лістинг коду:

import numpy as np

from sklearn import preprocessing

input\_data = np.array([[1.3, 3.9, 6.2],

 [4.9, 2.2, -4.3],

 [-2.2, 6.5, 4.1],

[-5.2, -3.4, -5.2]])

# Бінаризація даних

data\_binarized = preprocessing.Binarizer(threshold=2.0).transform(input\_data)

print("\n Binarized data:\n", data\_binarized)

# Исключение среднего

data\_scaled = preprocessing.scale(input\_data)

print("\nAFTER: ")

print("Mean =", data\_scaled.mean(axis=0))

print("Std deviation =", data\_scaled.std(axis=0))

# Масштабування MinМax

data\_scaler\_minmax = preprocessing.MinMaxScaler(feature\_range=(0, 1))

data\_scaled\_minmax = data\_scaler\_minmax.fit\_transform(input\_data)

print("\nМin max scaled data:\n", data\_scaled\_minmax)

# Нормалізація даних

data\_normalized\_l1 = preprocessing.normalize(input\_data, norm='l1')

data\_normalized\_l2 = preprocessing.normalize(input\_data, norm='l2')

print("\nl1 normalized data:\n", data\_normalized\_l1)

print("\nl2 normalized data:\n", data\_normalized\_l2)

Результат виконання:

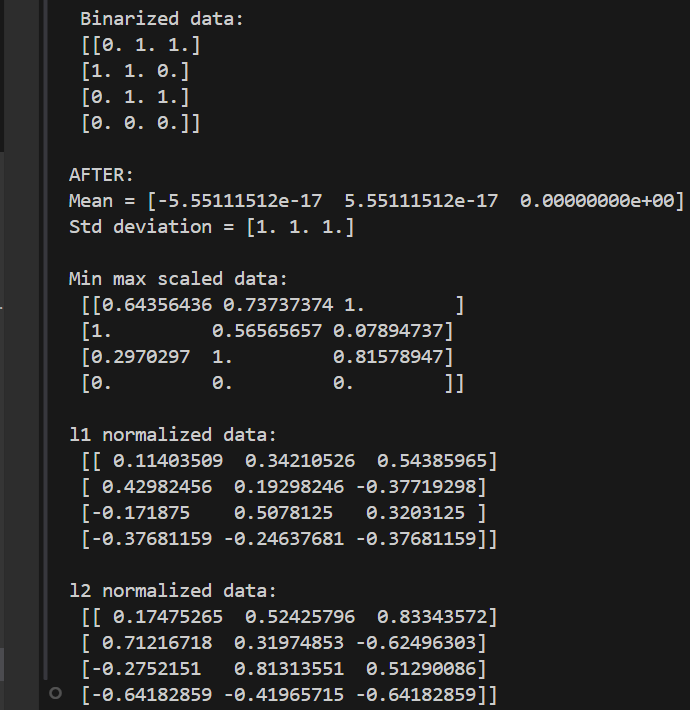


Рисунок 3- Результат виконання програми

**Завдання 2.3** Класифікація логістичною регресією або логістичний

класифікатор

Лістинг коду:

import numpy as np

from sklearn import linear\_model

import matplotlib.pyplot as plt

from utilities import visualize\_classifier

 # Визначення зразка вхідних даних

X = np.array([[3.1, 7.2], [4, 6.7], [2.9, 8], [5.1, 4.5],

 [6, 5], [5.6, 5], [3.3, 0.4],

 [3.9, 0.9], [2.8, 1],

 [0.5, 3.4], [1, 4], [0.6, 4.9]])

y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])

# Створення логістичного класифікатора

classifier =linear\_model.LogisticRegression(solver='liblinear',C=1)

# Тренування класифікатора

classifier.fit(X, y)

visualize\_classifier(classifier, X, y)

Результат виконання:

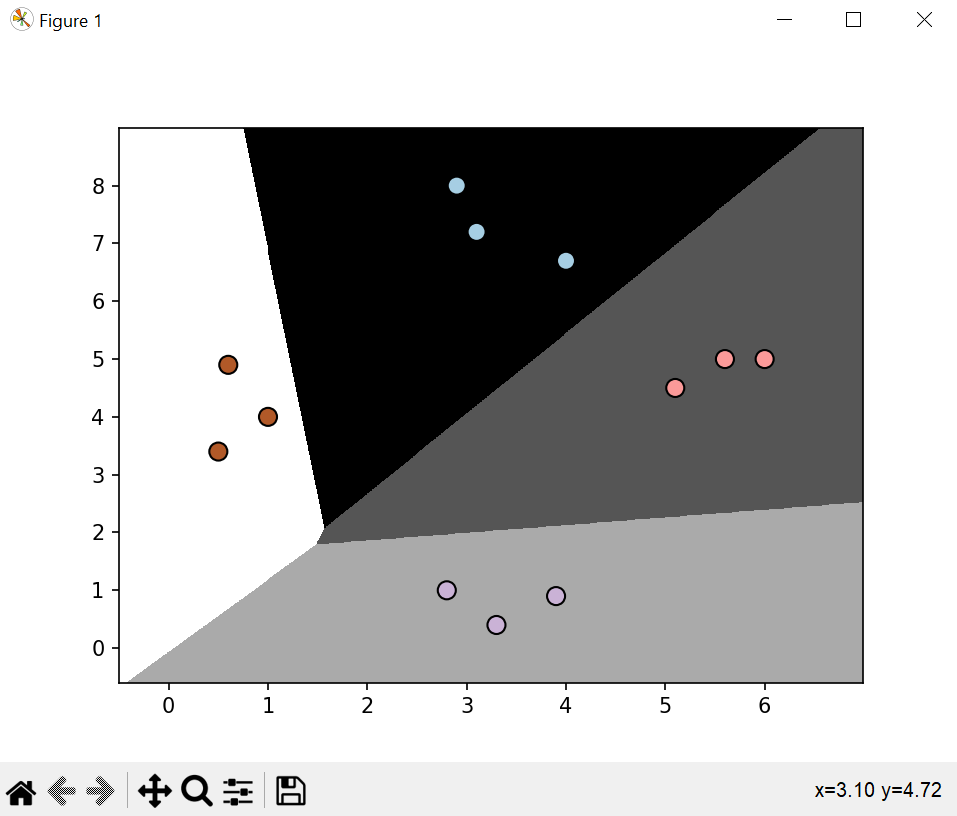


Рисунок 4- Результат виконання програми

**Завдання 2.4** Класифікація наївним байєсовським класифікатором

Лістинг коду:

import numpy as np

import matplotlib.pyplot as plt

from sklearn.naive\_bayes import GaussianNB

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

from utilities import visualize\_classifier

from sklearn.model\_selection import cross\_val\_score

# Вхідний файл, який містить дані

input\_file = 'data\_multivar\_nb.txt'

# Завантаження даних із вхідного файлу

data = np.loadtxt(input\_file, delimiter=',')

X, y = data[:, :-1], data[:, -1]

# Створення наївного байєсовського класифікатора

classifier = GaussianNB()

# Тренування класифікатора

classifier.fit(X, y)

# Прогнозування значень для тренувальних даних

y\_pred = classifier.predict(X)

# Обчислення якості класифікатора

accuracy = 100.0 \* (y == y\_pred).sum() / X.shape[0]

print("Accuracy of Naive Bayes classifier =", round(accuracy,2), "%")

# Візуалізація результатів роботи класифікатора

visualize\_classifier(classifier, X, y)

Результат виконання:

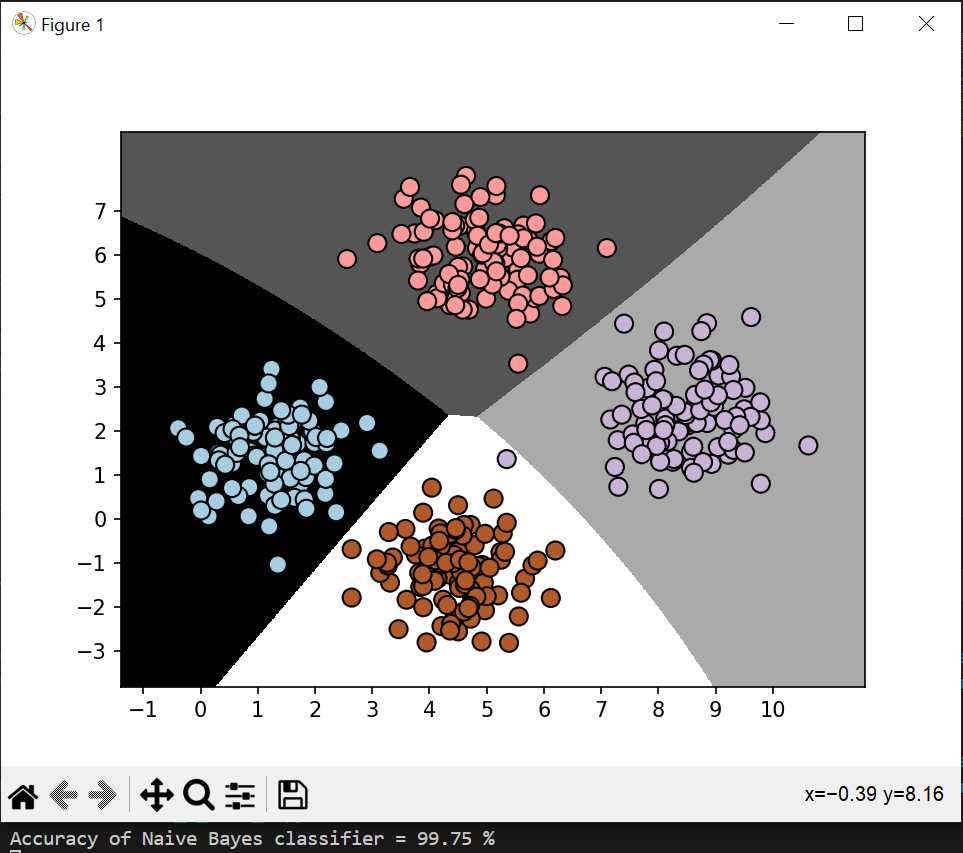


Рисунок 5- Результат виконання програми

Лістинг коду:

# Розбивка даних на навчальний та тестовий набори

X\_train, X\_test, y\_train, y\_test =train\_test\_split(X, y, test\_size=0.2, random\_state=3)

classifier\_new = GaussianNB()

classifier\_new.fit(X\_train, y\_train)

y\_test\_pred = classifier\_new.predict(X\_test)

# Обчислення якості класифікатора

accuracy = 100.0 \* (y\_test == y\_test\_pred).sum() / X\_test.shape[0]

print("Accuracy of the new classifier =", round(accuracy, 2),

"%")

# Візуалізація роботи класифікатора

visualize\_classifier(classifier\_new, X\_test, y\_test)

num\_folds = 3

accuracy\_values = cross\_val\_score(classifier, X, y, scoring='accuracy', cv=num\_folds)

print("Accuracy: " + str(round(100 \* accuracy\_values.mean(), 2)) + "%")

precision\_values = cross\_val\_score(classifier, X, y, scoring='precision\_weighted', cv=num\_folds)

print("Precision: " + str(round(100 \* precision\_values.mean(), 2)) + "%")

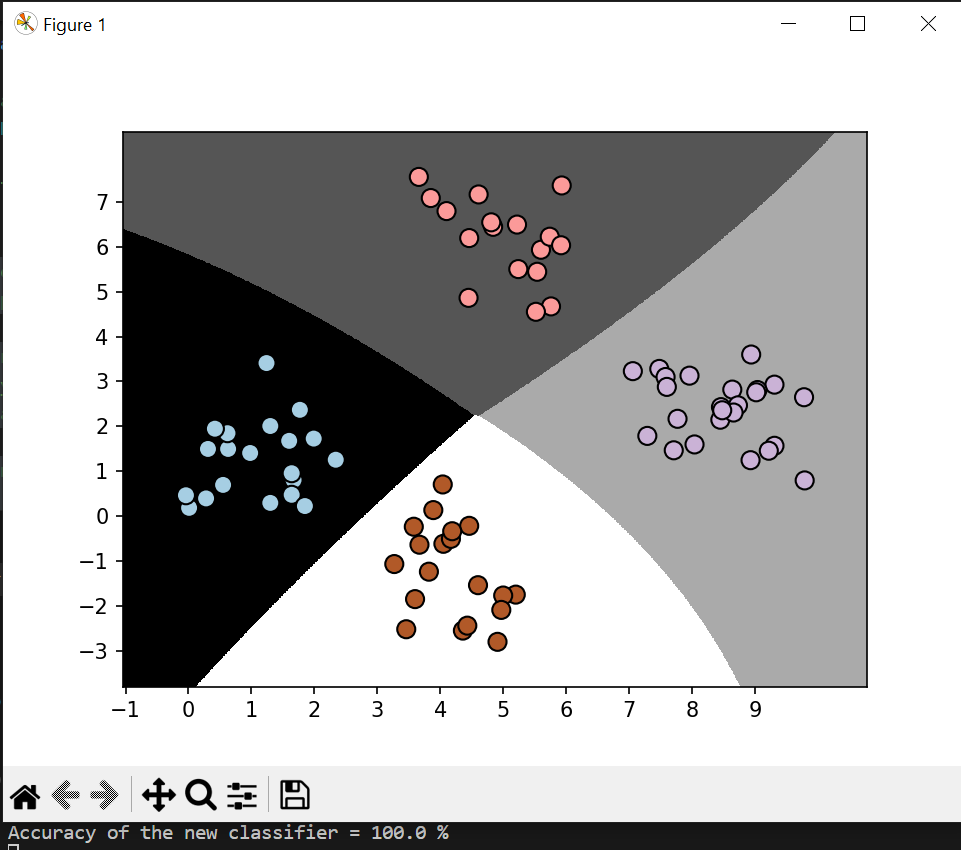
recall\_values = cross\_val\_score(classifier, X, y, scoring='recall\_weighted', cv=num\_folds)

print("Recall: " + str(round(100 \* recall\_values.mean(), 2)) + "%")

f1\_values = cross\_val\_score(classifier, X, y, scoring='f1\_weighted', cv=num\_folds)

print("F1: " + str(round(100 \* f1\_values.mean(), 2)) + "%")

Результат виконання:





Лістинг коду:

# Повторний прогін на новому поділі даних

X\_train\_new, X\_test\_new, y\_train\_new, y\_test\_new = train\_test\_split(X, y, test\_size=0.4, random\_state=3)

classifier\_new.fit(X\_train\_new, y\_train\_new)

y\_test\_pred\_new = classifier\_new.predict(X\_test\_new)

# Візуалізація результатів для нового прогону

visualize\_classifier(classifier\_new, X\_test\_new, y\_test\_new)

num\_folds = 3

accuracy\_values = cross\_val\_score(classifier, X, y, scoring='accuracy', cv=num\_folds)

print("Accuracy (new run): " + str(round(100 \* accuracy\_values.mean(), 2)) + "%")

precision\_values = cross\_val\_score(classifier, X, y, scoring='precision\_weighted', cv=num\_folds)

print("Precision (new run): " + str(round(100 \* precision\_values.mean(), 2)) + "%")

recall\_values = cross\_val\_score(classifier, X, y, scoring='recall\_weighted', cv=num\_folds)

print("Recall (new run): " + str(round(100 \* recall\_values.mean(), 2)) + "%")

f1\_values = cross\_val\_score(classifier, X, y, scoring='f1\_weighted', cv=num\_folds)

print("F1 (new run): " + str(round(100 \* f1\_values.mean(), 2)) + "%")

Результат виконання:

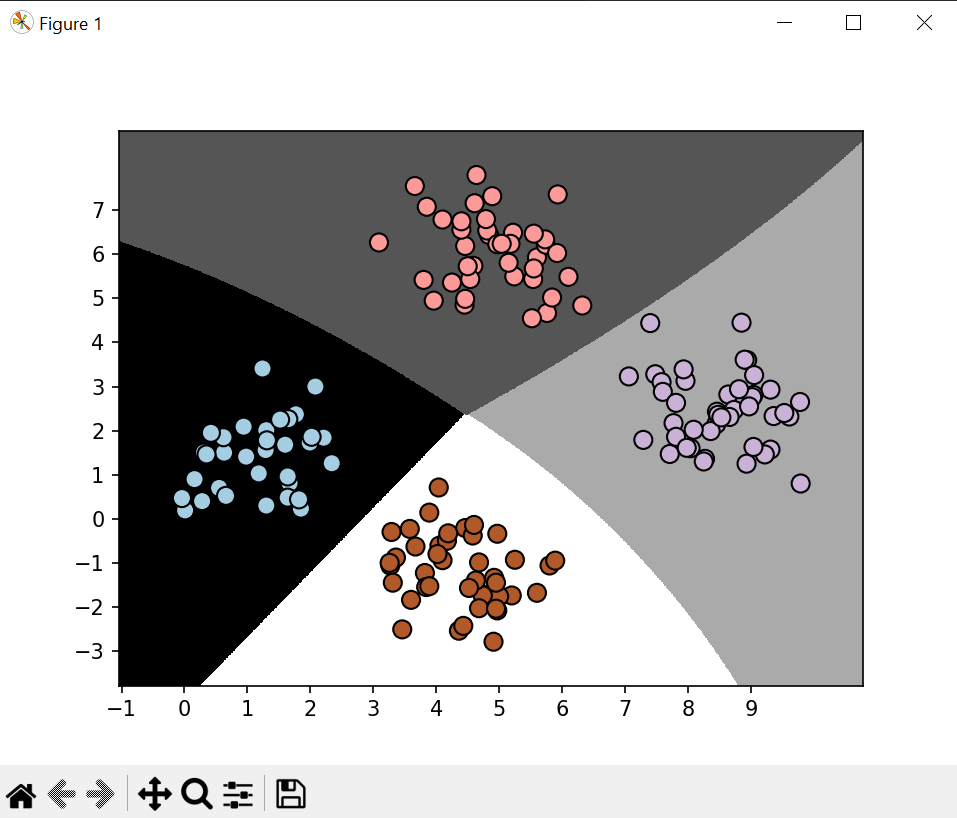




Рисунок 7- Результат виконання програми

У результаті обох прогонів класифікатора точність, precision, recall та F1-міра залишаються однаковими:

**Перший прогін:**

Accuracy: 99.75%

Precision: 99.76%

Recall: 99.75%

F1: 99.75%

**Другий прогін:**

Accuracy: 99.75%

Precision: 99.76%

Recall: 99.75%

F1: 99.75%

**Висновок:**

**Стабільність результатів:** Результати обох прогонів демонструють стабільність класифікації при зміні розміру тестової вибірки. Незважаючи на зміну параметра test\_size (з 0.2 на 0.4), показники точності, precision, recall та F1-міри не змінилися. Це свідчить про стійкість наївного байєсівського класифікатора до змін у поділі даних.

**Висока узагальненість:** Невелика різниця в розмірах навчальної та тестової вибірок не вплинула на результати класифікації, що свідчить про здатність моделі добре узагальнювати дані без підгонки під конкретні вибірки.

**Однорідність даних:** Той факт, що модель демонструє однакові результати при різних поділах даних, вказує на можливу однорідність даних та чітку структуру класифікації, яку модель легко розпізнає.

**Завдання 2.5:**

Лістинг коду:

import pandas as pd

import numpy as np

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

import math

import matplotlib.pyplot as plt

df = pd.read\_csv('data\_metrics.csv')

df.head()

thresh = 0.5

df['predicted\_RF'] = (df.model\_RF >= 0.5).astype('int')

df['predicted\_LR'] = (df.model\_LR >= 0.5).astype('int')

df.head()

confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)

def find\_TP(y\_true, y\_pred):

 # counts the number of true positives (y\_true = 1, y\_pred = 1)

    return sum((y\_true == 1) & (y\_pred == 1))

def find\_FN(y\_true, y\_pred):

 # counts the number of false negatives (y\_true = 1, y\_pred =0)

 return sum((y\_true == 1) & (y\_pred == 0))

def find\_FP(y\_true, y\_pred):

 # counts the number of false positives (y\_true = 0, y\_pred =1)

 return sum((y\_true == 0) & (y\_pred == 1))

def find\_TN(y\_true, y\_pred):

 # counts the number of true negatives (y\_true = 0, y\_pred =0)

 return sum((y\_true == 0) & (y\_pred == 0))

print('TP:',find\_TP(df.actual\_label.values, df.predicted\_RF.values))

print('FN:',find\_FN(df.actual\_label.values, df.predicted\_RF.values))

print('FP:',find\_FP(df.actual\_label.values, df.predicted\_RF.values))

print('TN:',find\_TN(df.actual\_label.values, df.predicted\_RF.values))

def find\_conf\_matrix\_values(y\_true,y\_pred):

 # calculate TP, FN, FP, TN

 TP = find\_TP(y\_true,y\_pred)

 FN = find\_FN(y\_true,y\_pred)

 FP = find\_FP(y\_true,y\_pred)

 TN = find\_TN(y\_true,y\_pred)

 return TP,FN,FP,TN

def volkov\_confusion\_matrix(y\_true, y\_pred):

 TP,FN,FP,TN = find\_conf\_matrix\_values(y\_true,y\_pred)

 return np.array([[TN,FP],[FN,TP]])

#Перевірка відповідності результатів зі створеною мною функції volkov\_confusion\_matrix

print('[TN, FP], \n[FN, TP] \n',volkov\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values))

assert np.array\_equal(volkov\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values),

confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)

), 'volkov\_confusion\_matrix() is not correct for RF'

assert np.array\_equal(volkov\_confusion\_matrix(df.actual\_label.values,

df.predicted\_LR.values),confusion\_matrix(df.actual\_label.values,

df.predicted\_LR.values) ), 'volkov\_confusion\_matrix() is not correct for LR'

print("Accurancy\_score -> ", accuracy\_score(df.actual\_label.values, df.predicted\_RF.values))

##################################################################

def volkov\_accuracy\_score(y\_true, y\_pred):

 TP,FN,FP,TN = find\_conf\_matrix\_values(y\_true,y\_pred)

 return (TP + TN) / (TP + TN + FP + FN)

assert volkov\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values) == accuracy\_score(df.actual\_label.values, df.predicted\_RF.values),'volkov\_recall\_score failed on'

assert volkov\_accuracy\_score(df.actual\_label.values,

df.predicted\_LR.values) == accuracy\_score(df.actual\_label.values, df.predicted\_LR.values), 'volkov\_recall\_score failed on LR'

print('Accuracy RF: %.3f'%(volkov\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('Accuracy LR: %.3f'%(volkov\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values)))

def volkov\_recall\_score(y\_true, y\_pred):

 # calculates the fraction of positive samples predicted correctly

 TP,FN,FP,TN = find\_conf\_matrix\_values(y\_true,y\_pred)

 return TP / (TP + FN)

assert volkov\_recall\_score(df.actual\_label.values, df.predicted\_RF.values) == recall\_score(df.actual\_label.values, df.predicted\_RF.values), 'volkov\_recall\_score failed on RF'

assert volkov\_recall\_score(df.actual\_label.values,

df.predicted\_LR.values) == recall\_score(df.actual\_label.values,

df.predicted\_LR.values), 'volkov\_recall\_score failed on LR'

print('Recall RF: %.3f'%(volkov\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('Recall LR: %.3f'%(volkov\_recall\_score(df.actual\_label.values, df.predicted\_LR.values)))

def volkov\_precision\_score(y\_true, y\_pred):

 # calculates the fraction of predicted positives samples that are actually positive

 TP,FN,FP,TN = find\_conf\_matrix\_values(y\_true,y\_pred)

 return TP / (TP + FP)

assert volkov\_precision\_score(df.actual\_label.values,

df.predicted\_RF.values) == precision\_score(df.actual\_label.values, df.predicted\_RF.values), 'volkov\_accuracy\_score failed on RF'

assert volkov\_precision\_score(df.actual\_label.values,

df.predicted\_LR.values) == precision\_score(df.actual\_label.values, df.predicted\_LR.values), 'volkov\_accuracy\_score failed on LR'

print('Precision RF: %.3f'%(volkov\_precision\_score(df.actual\_label.values,  df.predicted\_RF.values)))

print('Precision LR:%.3f'%(volkov\_precision\_score(df.actual\_label.values, df.predicted\_LR.values)))

def volkov\_f1\_score(y\_true, y\_pred):

 # calculates the F1 score

 recall = volkov\_recall\_score(y\_true,y\_pred)

 precision = volkov\_precision\_score(y\_true,y\_pred)

 return (2.0 \* (precision \* recall)) / (precision + recall)

# # В мене відбулася похибка на 16 знаку після коми в 1 одиницю тому нижче врахую її

# assert volkov\_f1\_score(df.actual\_label.values, df.predicted\_RF.values) == f1\_score(df.actual\_label.values, df.predicted\_RF.values),'volkov\_accuracy\_score failed on RF'

# assert volkov\_f1\_score(df.actual\_label.values, df.predicted\_LR.values) == f1\_score(df.actual\_label.values, df.predicted\_LR.values), 'volkov\_accuracy\_score failed on LR'

assert math.isclose(

    volkov\_f1\_score(df.actual\_label.values, df.predicted\_RF.values),

    f1\_score(df.actual\_label.values, df.predicted\_RF.values),

    rel\_tol=1e-15  # допустима відносна похибка

), 'volkov\_accuracy\_score failed on RF'

assert math.isclose(

    volkov\_f1\_score(df.actual\_label.values, df.predicted\_LR.values),

    f1\_score(df.actual\_label.values, df.predicted\_LR.values),

    rel\_tol=1e-15  # допустима відносна похибка

), 'volkov\_accuracy\_score failed on LR'

print('F1 RF: %.3f'%(volkov\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('F1 LR: %.3f'%(volkov\_f1\_score(df.actual\_label.values, df.predicted\_LR.values)))

print('scores with threshold = 0.5')

print('Accuracy RF: %.3f'%(volkov\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('Recall RF: %.3f'%(volkov\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('Precision RF: %.3f'%(volkov\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('F1 RF: %.3f'%(volkov\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('')

print('scores with threshold = 0.25')

print('Accuracy RF: %.3f'%(volkov\_accuracy\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))

print('Recall RF: %.3f'%(volkov\_recall\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))

print('Precision RF: %.3f'%(volkov\_precision\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))

print('F1 RF: %.3f'%(volkov\_f1\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))

Результати виконання:

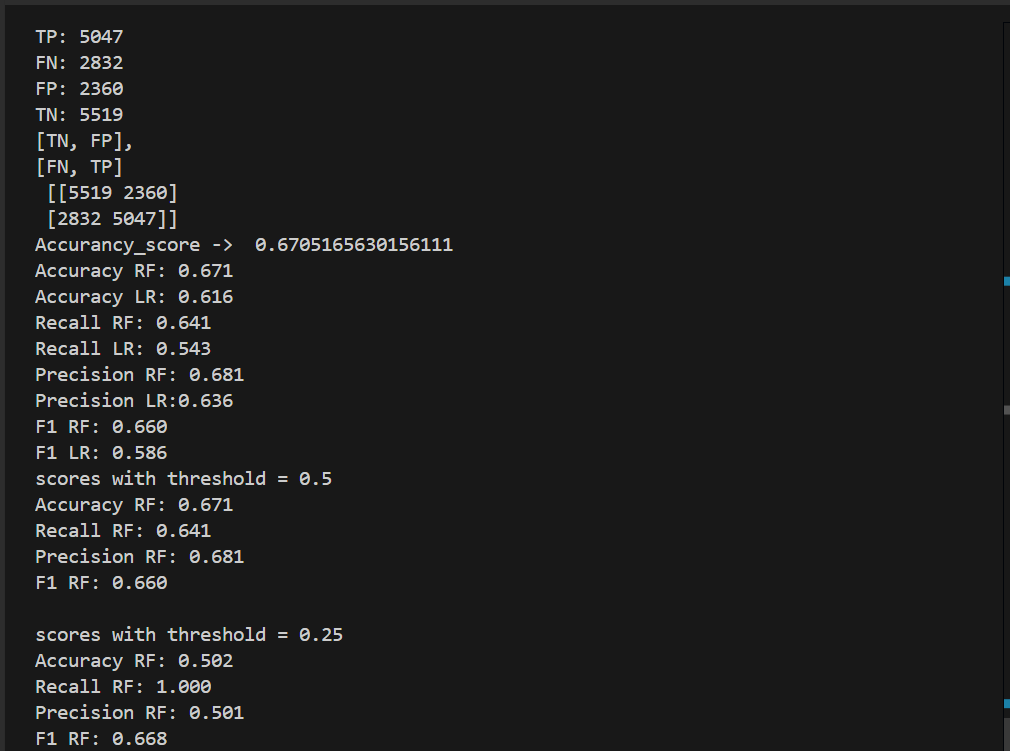


Рисунок 8- Результат виконання програми

**Порівняйте результати для різних порогів та зробіть висновки.**

**Висновки:**

1. **Поріг 0.5:**

* В цьому випадку значення accuracy є вищим, оскільки модель робить більше правильних передбачень, але компроміс полягає в нижчому recall — модель втрачає частину позитивних подій, яку ми правильно передбачили.
* Precision також залишається високим, оскільки серед передбачуваних позитивних прикладів модель досить добре справляється з передбаченням правильних (positive).

1. **Поріг 0.25:**

* Recall (Повнота) досягає максимального значення (1.000), тобто модель передбачає всі позитивні події як позитивні, тобто FN не знайдені.
* **Precision (Точність)** знижується до 0.501, що означає, що багато з передбачених позитивних подій насправді є хибними позитивами (false positives) - FP. Модель "схильна" передбачати більше позитивних результатів, що може призвести до багатьох помилкових спрацювань. Тобто серед передбачених позитивних прикладів модель помиляється приблизно у половині випадків.
* **Accuracy** також знижується до 0.502 через велику кількість хибнопозитивних передбачень. Це пов'язано з тим, що модель частіше видає позитивні результати, знижуючи загальну кількість правильних передбачень як для позитивних, так і для негативних класів.

**Загальний висновок:**

Зменшення порогу передбачень призводить до підвищення recall, але за рахунок зниження precision та accuracy. Це підходить у тих випадках, коли важливо уникнути пропуску позитивних прикладів, але в інших випадках (наприклад, де важлива загальна точність) кращим може бути вищий поріг, як 0.5.

Лістинг коду:

fpr\_RF, tpr\_RF,  thresholds\_RF = roc\_curve(df.actual\_label.values, df.model\_RF.values)

fpr\_LR, tpr\_LR, thresholds\_LR = roc\_curve(df.actual\_label.values, df.model\_LR.values)

plt.plot(fpr\_RF, tpr\_RF,'r-',label='RF')

plt.plot(fpr\_LR,tpr\_LR,'b-', label='LR')

plt.plot([0,1],[0,1],'k-',label='random')

plt.plot([0,0,1,1],[0,1,1,1],'g-',label='perfect')

plt.legend()

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

auc\_RF = roc\_auc\_score(df.actual\_label.values, df.model\_RF.values)

auc\_LR = roc\_auc\_score(df.actual\_label.values, df.model\_LR.values)

print('AUC RF:%.3f'% auc\_RF)

print('AUC LR:%.3f'% auc\_LR)

plt.plot(fpr\_RF, tpr\_RF,'r-',label = 'RF AUC: %.3f'%auc\_RF)

plt.plot(fpr\_LR,tpr\_LR,'b-', label= 'LR AUC: %.3f'%auc\_LR)

plt.plot([0,1],[0,1],'k-',label='random')

plt.plot([0,0,1,1],[0,1,1,1],'g-',label='perfect')

plt.legend()

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

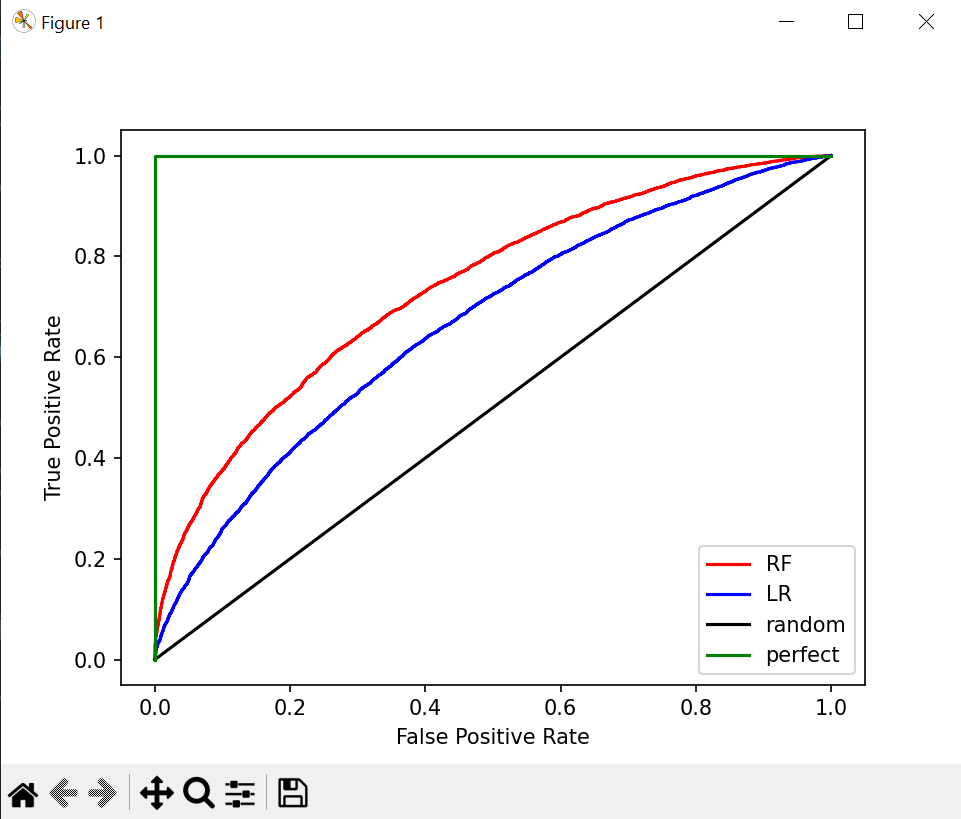


Рисунок 8- Результат виконання програми

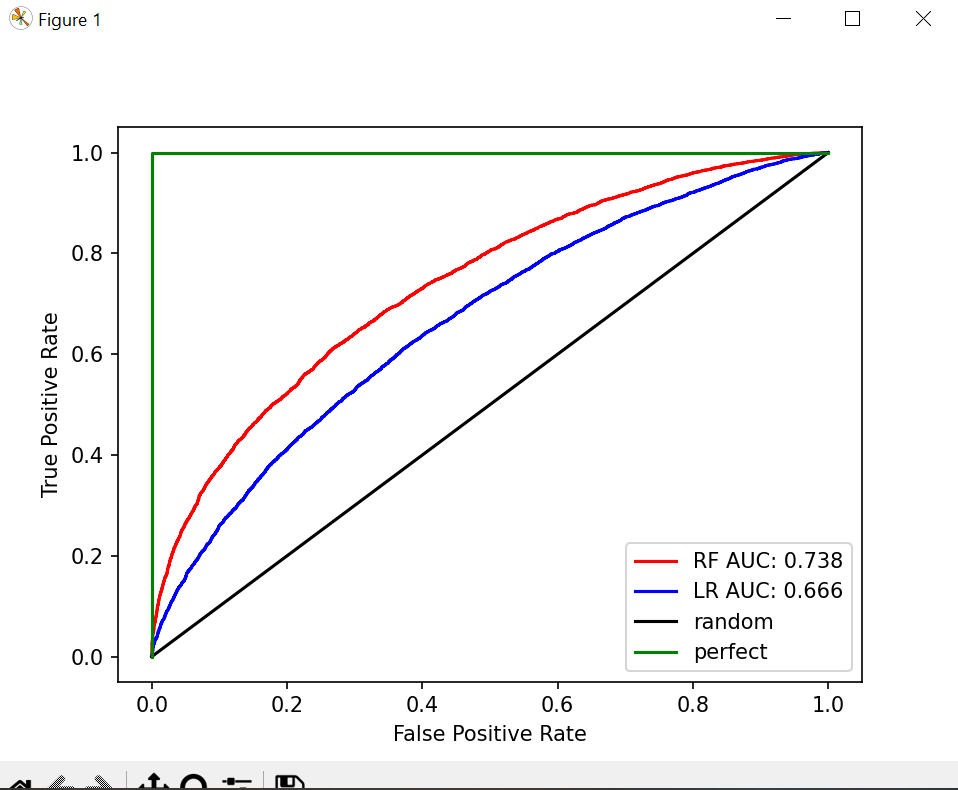


Рисунок 9- Результат виконання програми

**Висновок:** **RF** є кращою моделлю, оскільки її AUC більше, ніж у **LR**. Вона краще відрізняє позитивні і негативні класи.

**Завдання 2.6.** Розробіть програму класифікації даних в файлі data\_multivar\_nb.txt за допомогою машини опорних векторів (Support Vector Machine - SVМ). Розрахуйте показники якості класифікації. Порівняйте їх з показниками наївного байєсівського класифікатора. Зробіть висновки яку модель класифікації краще обрати і чому.

Лістинг коду:

import numpy as np

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

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

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Вхідний файл, який містить дані

data =  np.loadtxt('data\_multivar\_nb.txt', delimiter=',')

X = data[:, :-1]  # Ознаки обєкту

y = data[:, -1]   # Мітки класів

# Розбивка даних на навчальний та тестовий набори

X\_train, X\_test, y\_train, y\_test =train\_test\_split(X, y, test\_size=0.2, random\_state=3)

# Ініціалізація моделі SVM

svm\_model = SVC(kernel='linear', random\_state=3)

# Навчання моделі на навчальних даних

svm\_model.fit(X\_train, y\_train)

# Прогнозування на тестових даних

y\_pred = svm\_model.predict(X\_test)

# Оцінка якості класифікації

svm\_accuracy = accuracy\_score(y\_test, y\_pred)

svm\_conf\_matrix = confusion\_matrix(y\_test, y\_pred)

svm\_classification\_rep = classification\_report(y\_test, y\_pred)

print('Accuracy: {svm\_accuracy}')

print('Confusion Matrix:')

print(svm\_conf\_matrix)

print('Classification Report:')

print(svm\_classification\_rep)

# Ініціалізація моделі SVM

nb\_model = GaussianNB()

# Навчання моделі на навчальних даних

nb\_model.fit(X\_train, y\_train)

# Прогнозування на тестових даних

nb\_model.predict(X\_test)

# Оцінка якості класифікації

nb\_accuracy = accuracy\_score(y\_test, y\_pred)

nb\_conf\_matrix = confusion\_matrix(y\_test, y\_pred)

nb\_classification\_rep = classification\_report(y\_test, y\_pred)

print('Accuracy: {nb\_accuracy}')

print('Confusion Matrix:')

print(nb\_conf\_matrix)

print('Classification Report:')

print(nb\_classification\_rep)

#Порівняння моделей

if(nb\_accuracy>svm\_accuracy):

   print('Модель \'Gaussian Naive Bayes (GNB)\' за точністю є біль кращою')

elif(nb\_accuracy < svm\_accuracy):

   print('Модель \'(SupportVector Machine - SVМ\' за точністю є біль кращою')

else:

   print('Обидві моделі є однакові за точністю')

Результат виконання програми

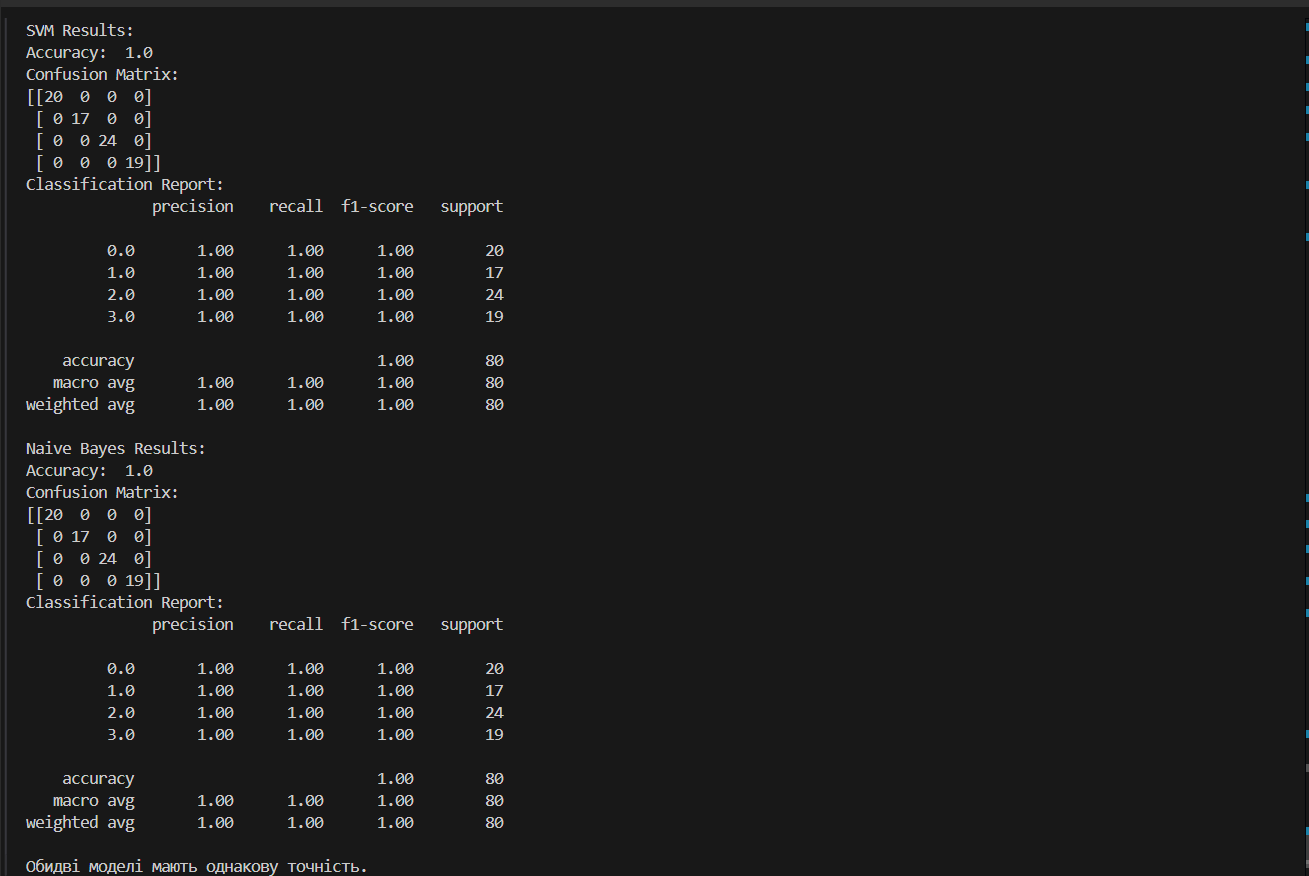


Рисунок 10- Результат виконання програми

В данному випадку обидві моделі — SVM і Naive Bayes — показали однакові результати з точністю 1.0, тобто обидві ідеально впоралися з класифікацією даних. Але для того, щоб вибрати, яку з них краще обрати, варто врахувати кілька речей:

* Якщо потрібна модель, яку легко налаштувати і швидко запустити, Naive Bayes може бути кращим вибором. Він швидкий і добре працює, коли немає складних залежностей між ознаками.
* Якщо дані складні або мають багато особливостей, краще підійде SVM. Ця модель може краще працювати, коли дані нелінійні або мають нечіткі або зайві данні.
* На великих наборах даних Naive Bayes працює швидше, але SVM може дати точніші результати, хоча працюватиме трохи повільніше.

Оскільки, у нас немає складних данних, на мою думку, сюди краще підходить модель Naive Bayes.

**Висновок:** використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив попередню обробку та класифікацію даних.