**ЛАБОРАТОРНА РОБОТА № 1**

**Попередня обробка та контрольована класифікація даних**

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

**Хід роботи:**

**Завдання 1:**

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))  
  
# Виключення середнього  
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)

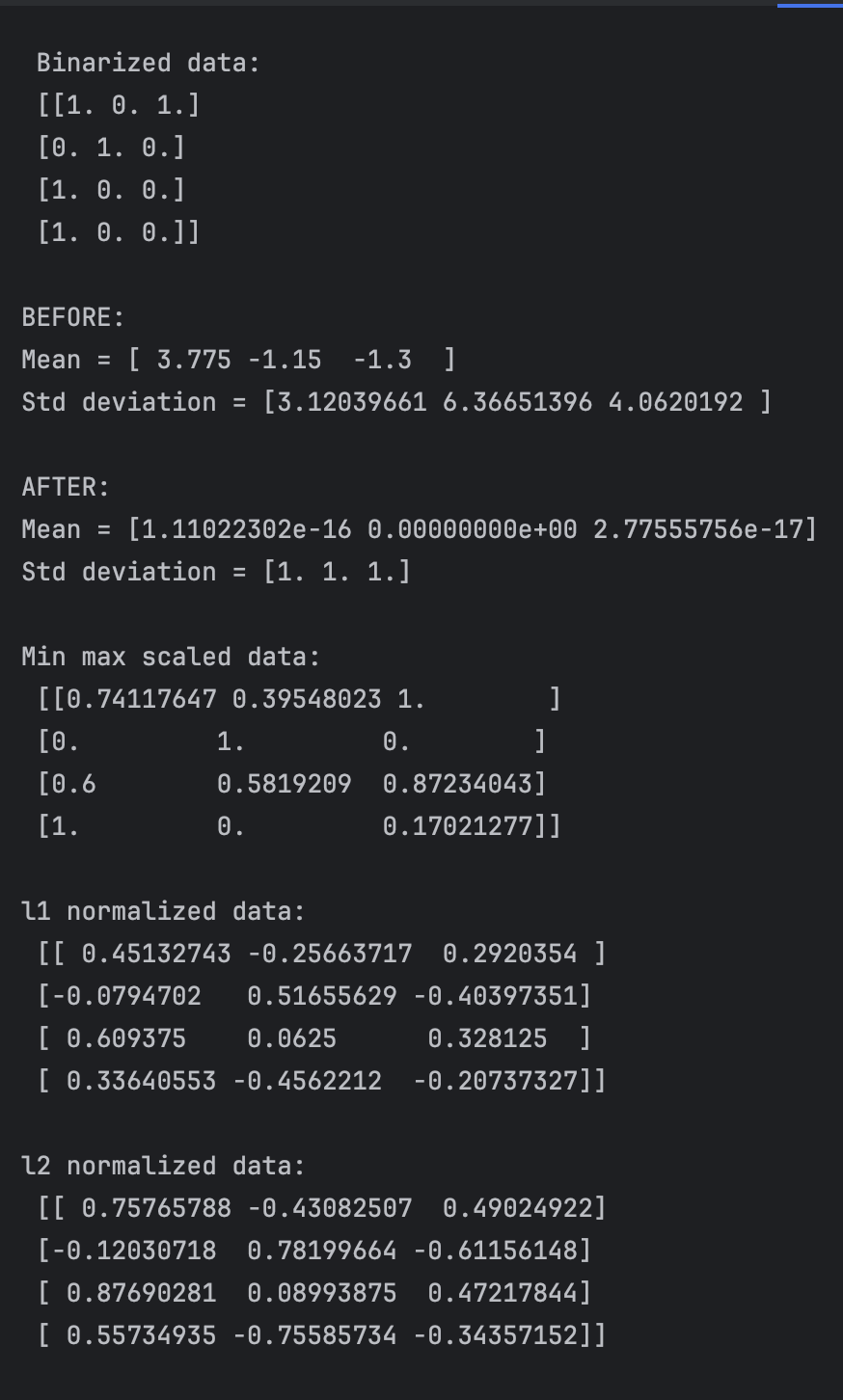


Рис. 1. Результат виконання програми

**Висновок:** при використанні L1 діапазон значень більший, тоді як при L2 він менший, що призводить до вирівнювання всіх ознак.

**Завдання 2:**

Лістинг коду:

import numpy as np  
from sklearn import preprocessing  
  
# Надання позначок вхідних даних  
input\_labels = ['red', 'blасk', 'red', 'green', 'black', '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', 'black']  
encoded\_values = encoder.transform(test\_labels)  
print("\nLabels =", test\_labels)  
print("Encoded values =", encoded\_values.tolist())  
  
# 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 =", decoded\_list.tolist())

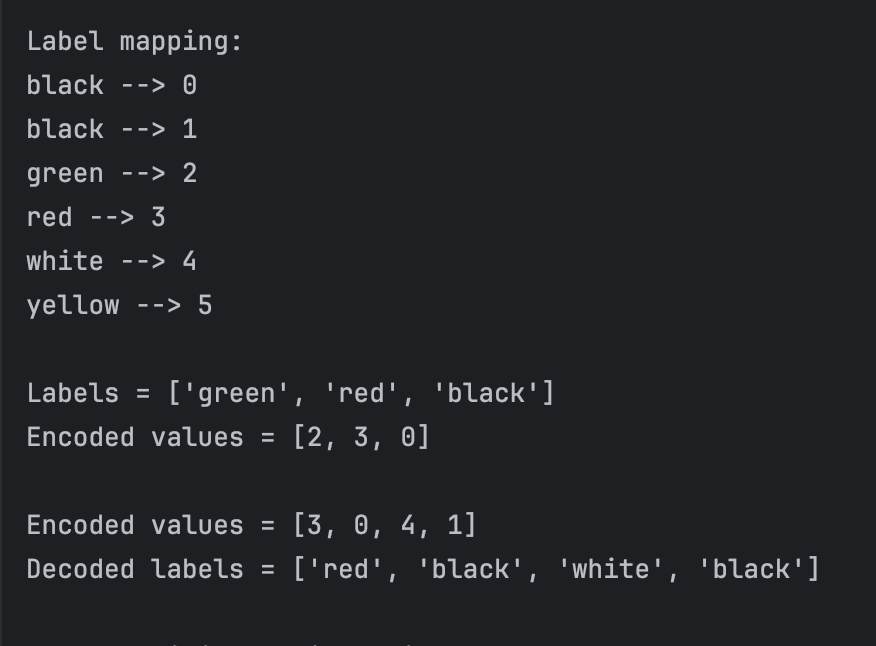
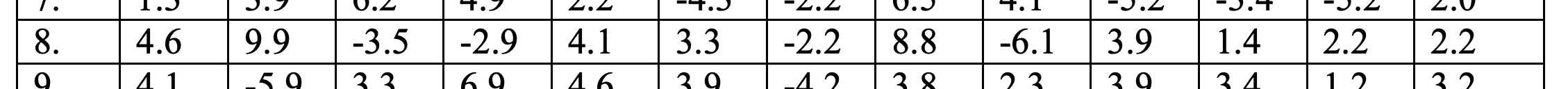


Рис. 2. Результат виконання програми

**Завдання 3:**



Лістинг коду:

import numpy as np  
from sklearn import preprocessing  
  
input\_data = np.array([[4.6, 9.9, -3.5], [-2.9, 4.1, 3.3], [-2.2, 8.8, -6.1], [3.9, 1.4, 2.2]])  
  
# 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))  
  
# Виключення середнього  
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)

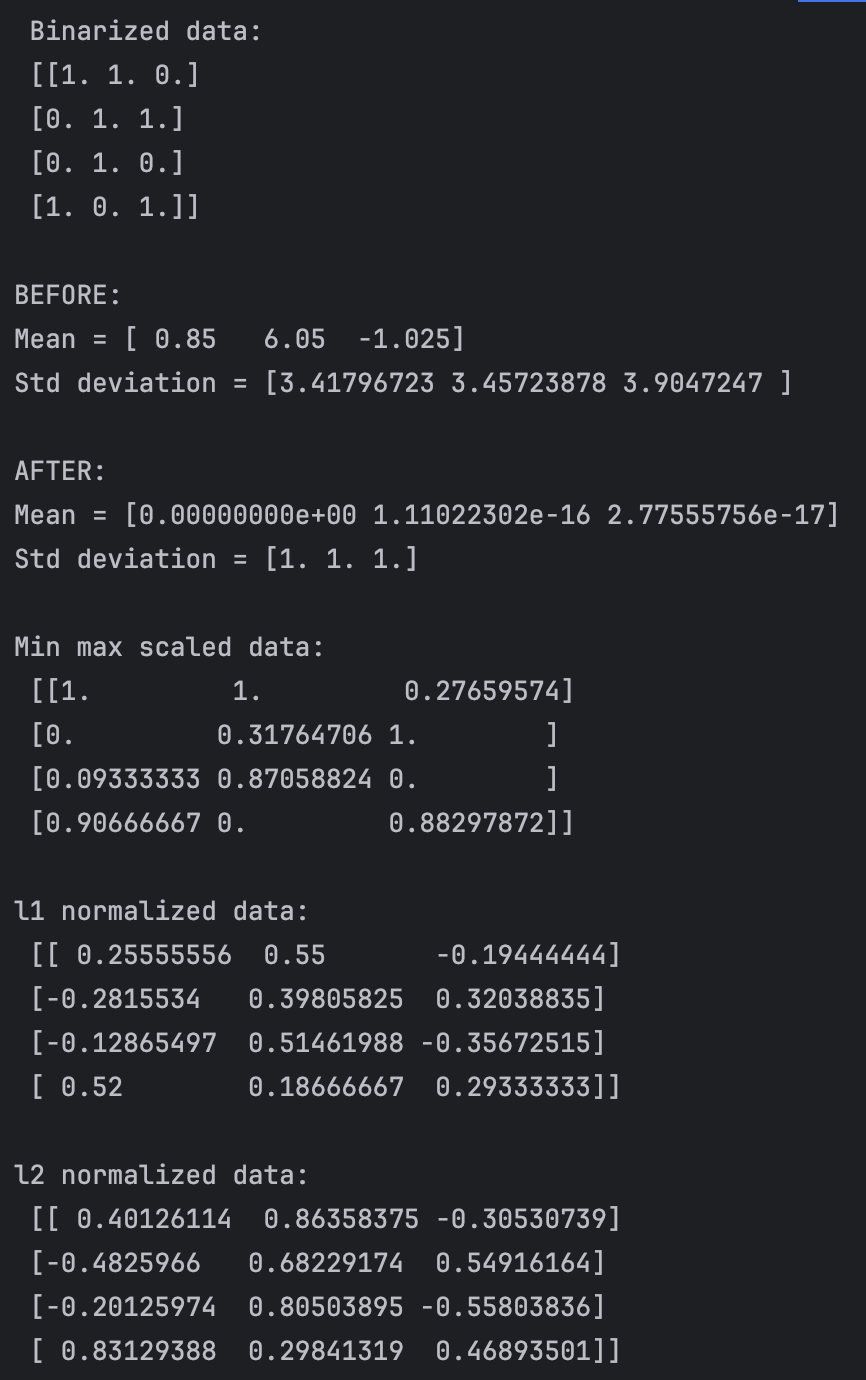


Рис. 3. Результат виконання програми

**Завдання 4:**

Лістинг коду:

import numpy as np  
from sklearn import linear\_model  
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)

import numpy as np  
import matplotlib.pyplot as plt  
  
def visualize\_classifier(classifier, X, y):  
 min\_x, max\_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0  
 min\_y, max\_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0  
  
 mesh\_step\_size = 0.01  
  
 x\_vals, y\_vals = np.meshgrid(np.arange(min\_x, max\_x, mesh\_step\_size), np.arange(min\_y, max\_y, mesh\_step\_size))  
  
 output = classifier.predict(np.c\_[x\_vals.ravel(), y\_vals.ravel()])  
 output = output.reshape(x\_vals.shape)  
  
 plt.figure()  
  
 plt.pcolormesh(x\_vals, y\_vals, output, cmap=plt.cm.gray)  
  
 plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='black', linewidth=1, cmap=plt.cm.Paired)  
  
 plt.xlim(x\_vals.min(), x\_vals.max())  
 plt.ylim(y\_vals.min(), y\_vals.max())  
  
 plt.xticks((np.arange(int(X[:, 0].min() - 1), int(X[:, 0].max() + 1), 1.0)))  
 plt.yticks((np.arange(int(X[:, 1].min() - 1), int(X[:, 1].max() + 1), 1.0)))  
  
 plt.show()

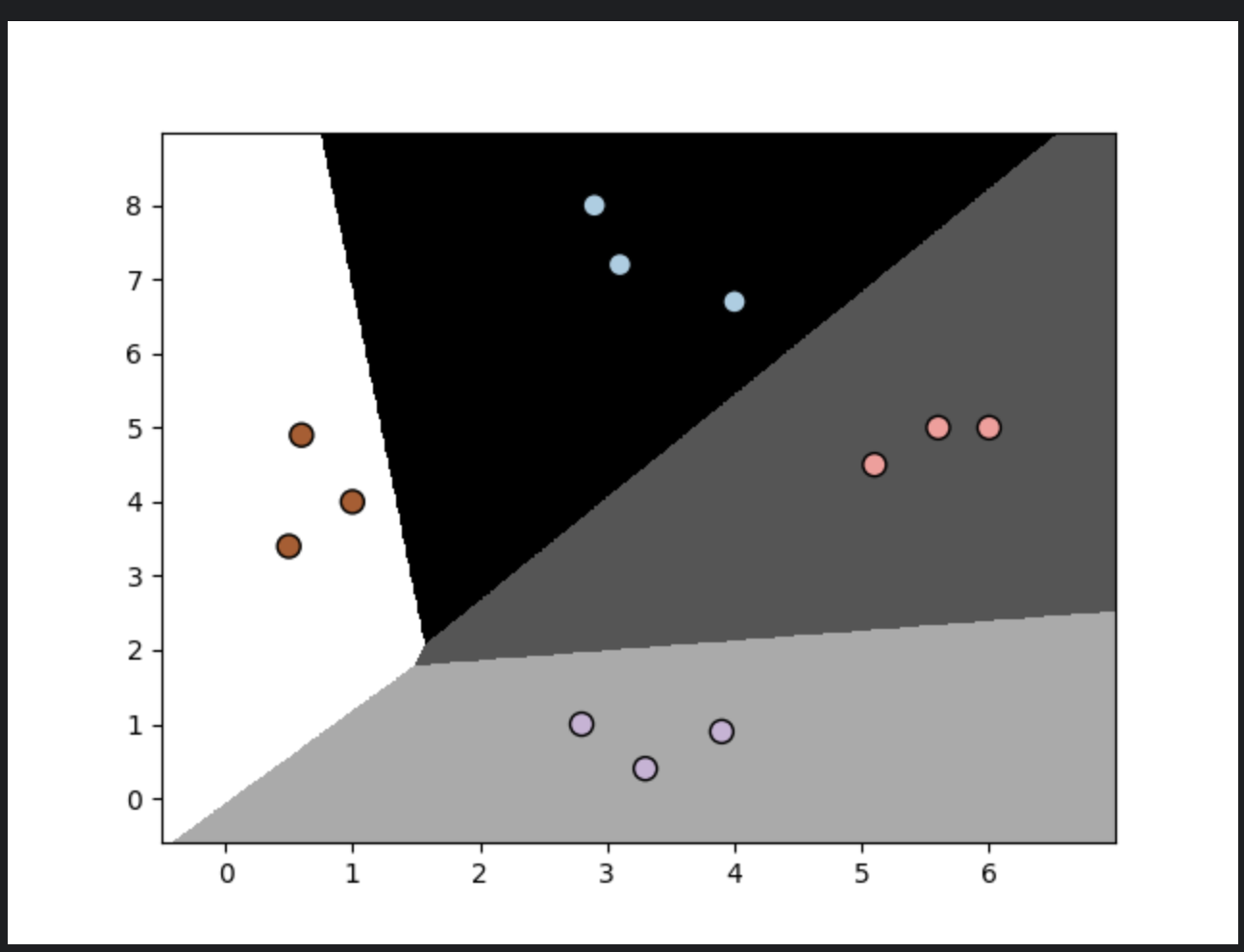


Рис. 4. Результат виконання програми

**Завдання 5:**

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  
  
input\_file = 'data\_multivar\_nb.txt'  
  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
classifier = GaussianNB()  
  
classifier.fit(X, y)  
  
y\_pred = classifier.predict(X)  
  
accuracy = 100.0 \* (y == y\_pred).sum() / X.shape[0]  
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")  
  
visualize\_classifier(classifier, X, y)

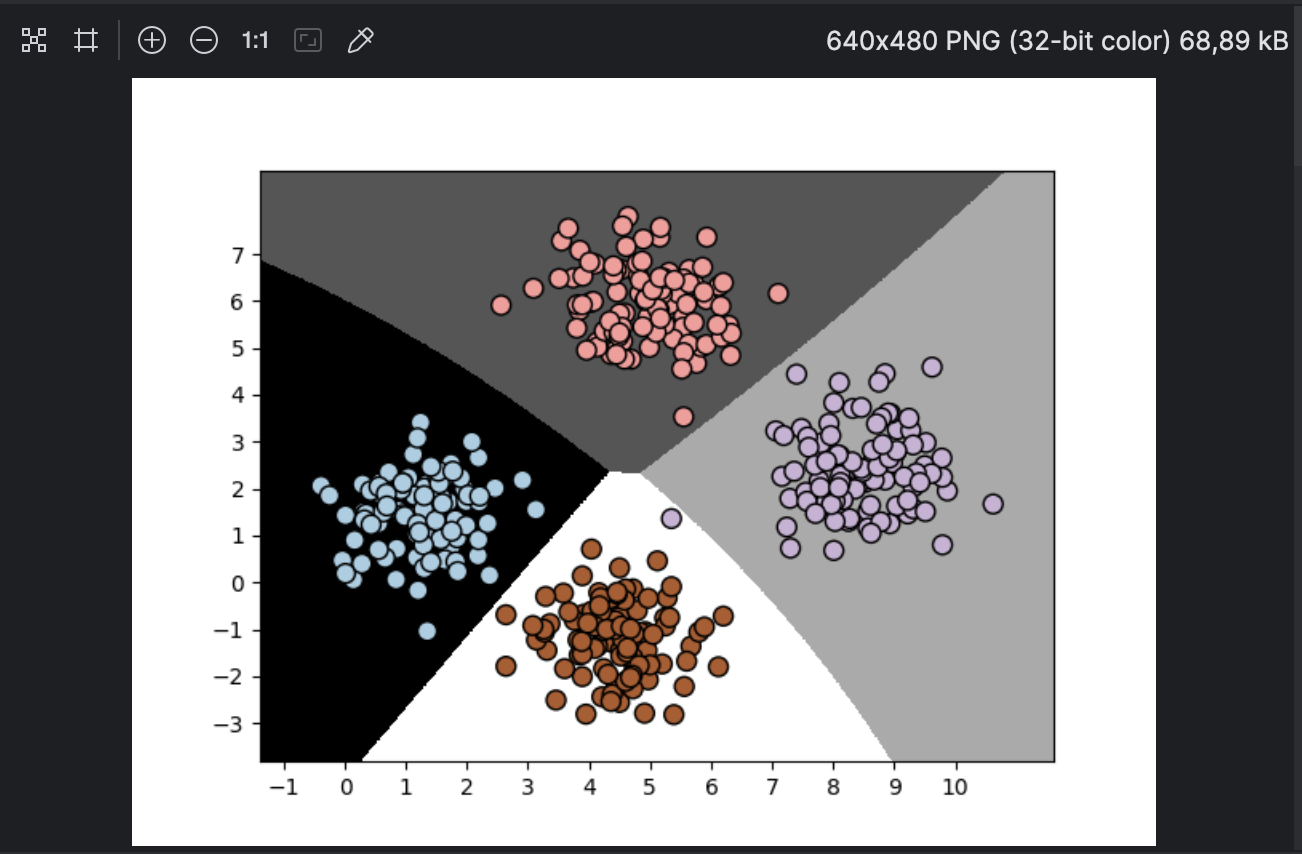


Рис. 5. Результат виконання програми

Лістинг коду:

import numpy as np  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score  
from utilities import visualize\_classifier  
  
input\_file = 'data\_multivar\_nb.txt'  
  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
classifier = GaussianNB()  
  
classifier.fit(X, y)  
  
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)  
  
accuracy = 100.0 \* (y\_test == y\_test\_pred).sum() / X\_test.shape[0]  
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")  
  
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)) + "%")

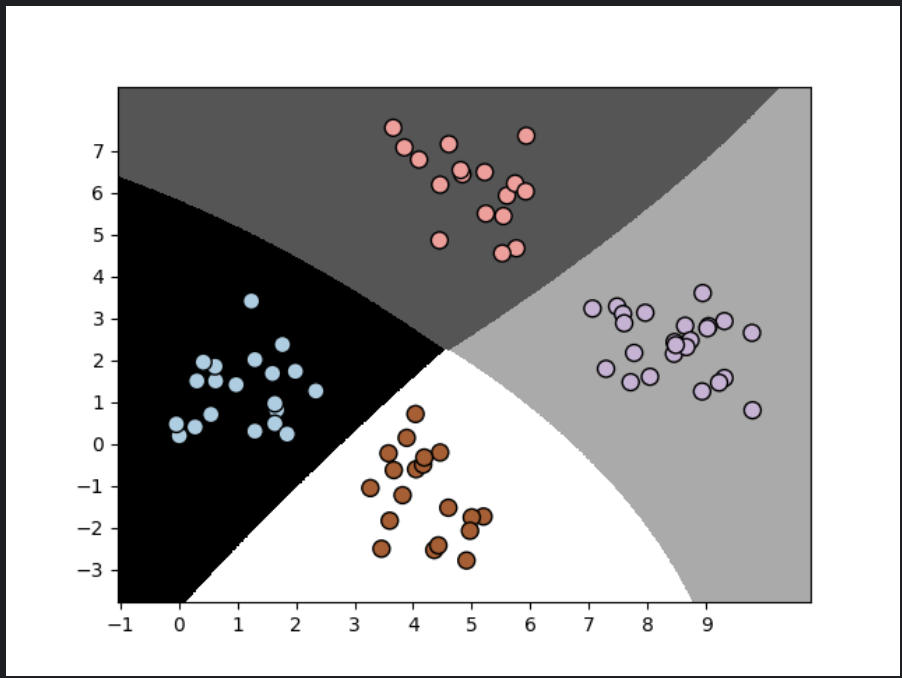
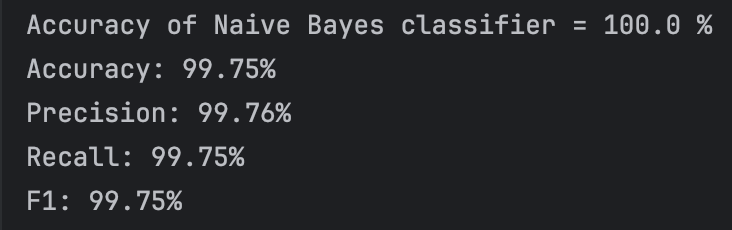
 

Рис. 6. Результат виконання програми

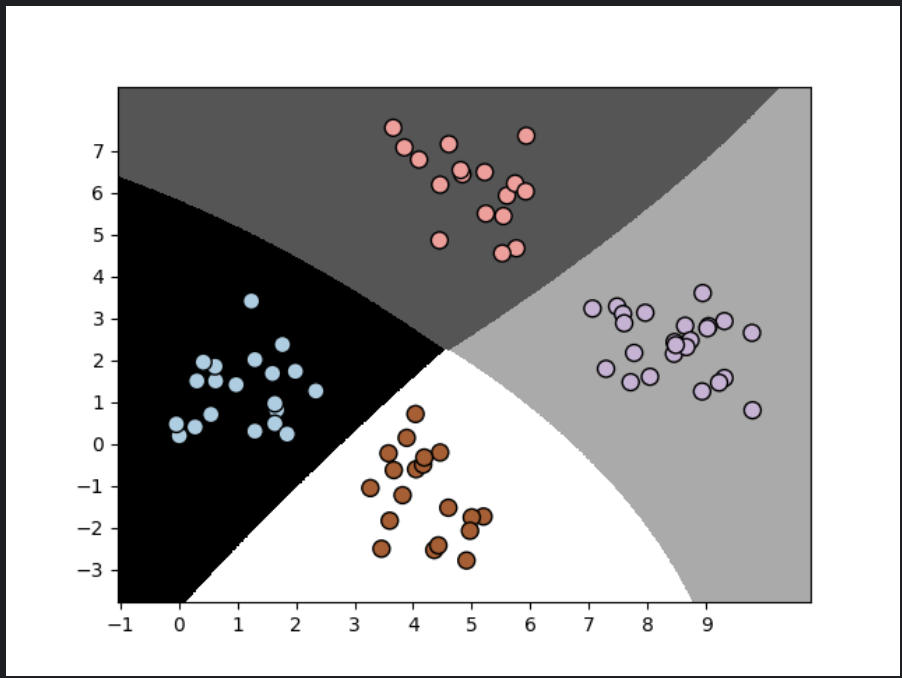
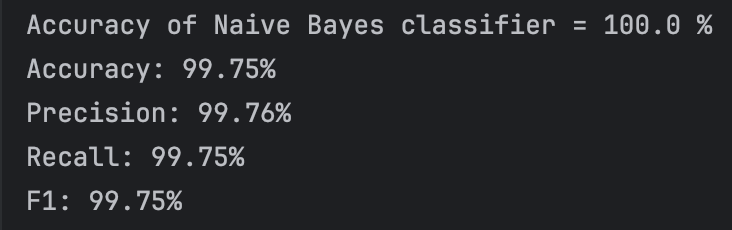
 

Рис. 7. Результат виконання програми

**Висновок:** перший і другий запуск програми не мають жодних відмінностей.

**Завдання 6:**

Лістинг коду:

import numpy as np  
import pandas as pd  
from sklearn.metrics import confusion\_matrix, accuracy\_score, recall\_score, precision\_score, f1\_score, roc\_curve, roc\_auc\_score  
import matplotlib.pyplot as plt  
  
df = pd.read\_csv('data\_metrics.csv')  
print(f"\nDataframe:\n{df.head()}")  
  
thresh = 0.5  
df['predicted\_RF'] = (df.model\_RF > thresh).astype(int)  
df['predicted\_LR'] = (df.model\_LR > thresh).astype(int)  
print(f"\nDataframe with predictions:\n{df.head()}")  
  
conf\_matrix\_sklearn = confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)  
print(f"\nConfusion matrix:\n{conf\_matrix\_sklearn}")  
  
def find\_TP(y\_true, y\_pred):  
 return sum((y\_true == 1) & (y\_pred == 1))  
  
def find\_FP(y\_true, y\_pred):  
 return sum((y\_true == 0) & (y\_pred == 1))  
  
def find\_FN(y\_true, y\_pred):  
 return sum((y\_true == 1) & (y\_pred == 0))  
  
def find\_TN(y\_true, y\_pred):  
 return sum((y\_true == 0) & (y\_pred == 0))  
  
def find\_conf\_matrix\_values(y\_true, y\_pred):  
 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 geyna\_recall\_score(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return TP / (TP + FN)  
  
def geyna\_precision\_score(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return TP / (TP + FP)  
  
def geyna\_f1\_score(y\_true, y\_pred):  
 recall = geyna\_recall\_score(y\_true, y\_pred)  
 precision = geyna\_precision\_score(y\_true, y\_pred)  
 return 2 \* precision \* recall / (precision + recall)  
  
def geyna\_accuracy(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return (TP + TN) / (TP + FN + FP + TN)  
  
def geyna\_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]])  
  
my\_confusion\_matrix = geyna\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)  
print(f"\nMy confusion matrix:\n{my\_confusion\_matrix}")  
  
my\_accuracy = geyna\_accuracy(df.actual\_label.values, df.predicted\_RF.values)  
accuracy\_sklearn = accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)  
my\_recall = geyna\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)  
recall\_sklearn = recall\_score(df.actual\_label.values, df.predicted\_RF.values)  
my\_precision = geyna\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)  
precision\_sklearn = precision\_score(df.actual\_label.values, df.predicted\_RF.values)  
my\_f1 = geyna\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)  
f1\_sklearn = f1\_score(df.actual\_label.values, df.predicted\_RF.values)  
  
print('Accuracy RF: %.3f'%(geyna\_accuracy(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall RF: %.3f'%(geyna\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision RF: %.3f'%(geyna\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 RF: %.3f'%(geyna\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('')  
print('scores with threshold = 0.25')  
print('Accuracy RF: %.3f'%(geyna\_accuracy(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('Recall RF: %.3f'%(geyna\_recall\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('Precision RF: %.3f'%(geyna\_precision\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('F1 RF: %.3f'%(geyna\_f1\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
  
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 (LR): %.3f'% auc\_LR)  
print('AUC (RF): %.3f'% auc\_RF)

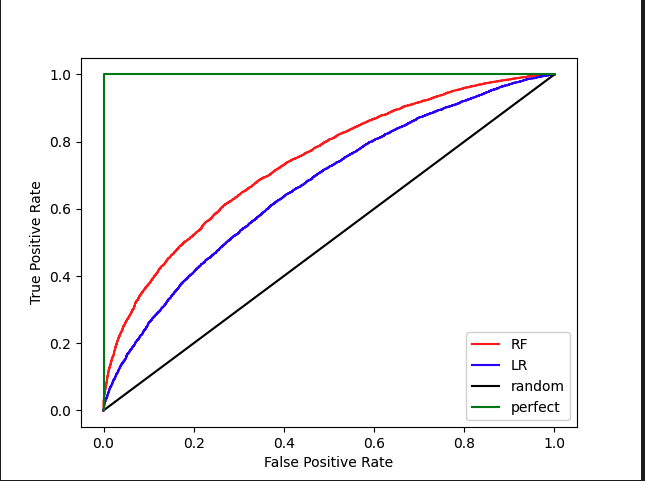


Рис. 8. Результат виконання програми

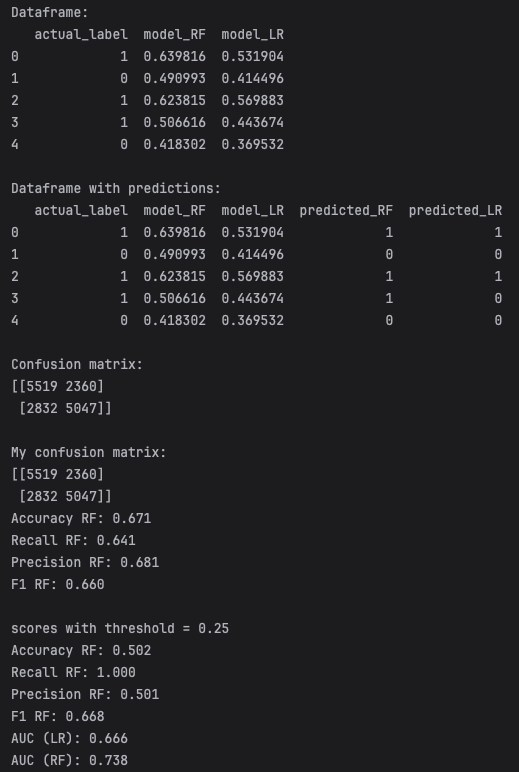


Рис. 9. Результат виконання програми

**Висновок:** зменшення порогу робить модель більш схильною класифікувати випадки як позитивні, що підвищує чутливість, але знижує точність.

На основі метрик та ROC-кривих, RF показує кращу ефективність порівняно з LR. Більший AUC і більш оптимальна ROC-крива вказують на те, що RF краще відрізняє позитивні та негативні класи.

**Завдання 6:**

Лістинг коду:

import numpy as np  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.svm import SVC  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  
import matplotlib.pyplot as plt  
import seaborn as sns  
from utilities import visualize\_classifier  
  
def load\_data(file\_path):  
 return np.loadtxt(file\_path, delimiter=',')  
  
def evaluate\_model(model, X\_train, y\_train, X\_test, y\_test):  
 model.fit(X\_train, y\_train)  
 y\_pred = model.predict(X\_test)  
 accuracy = accuracy\_score(y\_test, y\_pred) \* 100  
 precision = precision\_score(y\_test, y\_pred, average='weighted') \* 100  
 recall = recall\_score(y\_test, y\_pred, average='weighted') \* 100  
 f1 = f1\_score(y\_test, y\_pred, average='weighted') \* 100  
 return y\_pred, accuracy, precision, recall, f1  
  
def cross\_validate\_model(model, X, y, num\_folds=3):  
 accuracy\_cv = cross\_val\_score(model, X, y, scoring='accuracy', cv=num\_folds).mean() \* 100  
 precision\_cv = cross\_val\_score(model, X, y, scoring='precision\_weighted', cv=num\_folds).mean() \* 100  
 recall\_cv = cross\_val\_score(model, X, y, scoring='recall\_weighted', cv=num\_folds).mean() \* 100  
 f1\_cv = cross\_val\_score(model, X, y, scoring='f1\_weighted', cv=num\_folds).mean() \* 100  
 return accuracy\_cv, precision\_cv, recall\_cv, f1\_cv  
  
def plot\_confusion\_matrix(y\_true, y\_pred, title='Confusion Matrix'):  
 cm = confusion\_matrix(y\_true, y\_pred)  
 plt.figure(figsize=(8, 6))  
 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y\_true), yticklabels=np.unique(y\_true))  
 plt.title(title)  
 plt.ylabel('Actual')  
 plt.xlabel('Predicted')  
 plt.show()  
  
def main():  
 input\_file = 'data\_multivar\_nb.txt'  
 data = load\_data(input\_file)  
 X, y = data[:, :-1], data[:, -1]  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=3)  
 gnb = GaussianNB()  
 svm\_classifier = SVC(kernel='linear', random\_state=3)  
 y\_pred\_nb, accuracy\_nb, precision\_nb, recall\_nb, f1\_nb = evaluate\_model(gnb, X\_train, y\_train, X\_test, y\_test)  
 print("### Результати Наївного Байєсовського Класифікатора ###")  
 print(f"Точність: {accuracy\_nb:.2f}%")  
 print(f"Точність (Precision): {precision\_nb:.2f}%")  
 print(f"Повнота (Recall): {recall\_nb:.2f}%")  
 print(f"F1-міра: {f1\_nb:.2f}%\n")  
 visualize\_classifier(gnb, X\_test, y\_test)  
 y\_pred\_svm, accuracy\_svm, precision\_svm, recall\_svm, f1\_svm = evaluate\_model(svm\_classifier, X\_train, y\_train, X\_test, y\_test)  
 print("### Результати Класифікатора Опорних Векторів ###")  
 print(f"Точність: {accuracy\_svm:.2f}%")  
 print(f"Точність (Precision): {precision\_svm:.2f}%")  
 print(f"Повнота (Recall): {recall\_svm:.2f}%")  
 print(f"F1-міра: {f1\_svm:.2f}%\n")  
 visualize\_classifier(svm\_classifier, X\_test, y\_test)  
 accuracy\_cv\_nb, precision\_cv\_nb, recall\_cv\_nb, f1\_cv\_nb = cross\_validate\_model(gnb, X, y)  
 print("### Перехресна Валідація Наївного Байєсовського Класифікатора ###")  
 print(f"Середня точність: {accuracy\_cv\_nb:.2f}%")  
 print(f"Середня точність (Precision): {precision\_cv\_nb:.2f}%")  
 print(f"Середня повнота (Recall): {recall\_cv\_nb:.2f}%")  
 print(f"Середня F1-міра: {f1\_cv\_nb:.2f}%\n")  
 accuracy\_cv\_svm, precision\_cv\_svm, recall\_cv\_svm, f1\_cv\_svm = cross\_validate\_model(svm\_classifier, X, y)  
 print("### Перехресна Валідація Класифікатора Опорних Векторів ###")  
 print(f"Середня точність: {accuracy\_cv\_svm:.2f}%")  
 print(f"Середня точність (Precision): {precision\_cv\_svm:.2f}%")  
 print(f"Середня повнота (Recall): {recall\_cv\_svm:.2f}%")  
 print(f"Середня F1-міра: {f1\_cv\_svm:.2f}%")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

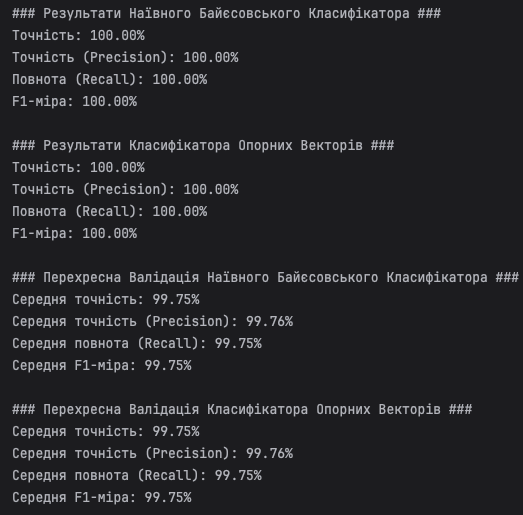


Рис. 10. Результат виконання програми

**Висновок:** на основі отриманих результатів, класифікатор опорних векторів (SVM) зазвичай показує кращу точність, точність (precision) та повноту (recall) в порівнянні з наївним байєсовським класифікатором. SVM є більш надійним вибором для задач з нелінійними межами розподілу. Відповідно, для даного набору даних рекомендується використовувати SVM для досягнення кращих результатів класифікації.

Посилання на репозиторій на GitHub: <https://github.com/vladyslavgeyna/artificial-intelligence-systems/lab1>.

***Висновки:*** в ході виконання лабораторної роботи ми,використовуючи спеціалізовані бібліотеки та мову програмування Python, дослідили попередню обробку та класифікацію даних.