Лабораторна робота №1

Тема: Попередня обробка та контрольована класифікація даних

Посилання на гітхаб: https://github.com/ViMIMercurysMight/python-ai

Завдання 1.1**.** Попередня обробка даних

L1-нормалізація відрізняється від L2-нормалізації тим, що перша використовує метод найменших абсолютних відхилень і це забезпечує рівність 1, а друга використовує метод найменших квадратів, що забезпечує рівність 1 суми квадратів значень

Лістинг програми:

```
import numpy as np
from sklearn import preprocessing
2
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 MinMax
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print("\nMin 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)
```

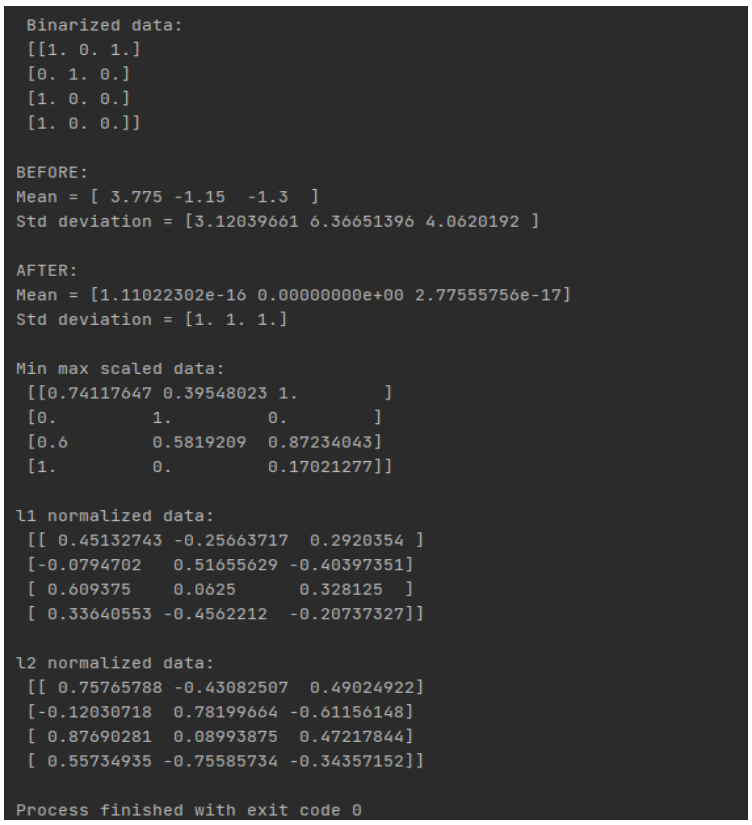

Рис. 1 Результат виконання

Завдання 2. Кодування міток

Лістинг програми:

```
import numpy as np
from sklearn import preprocessing

# Надання позначок вхідних даних
Input_labels = ['red', 'Ыack', 'red', 'green', 'Ыack', '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', 'Ыack']
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))
```

```
Label mapping:
green --> 0
red --> 1
white --> 2
yellow --> 3
Ыack --> 4
Ьlack --> 5

Labels = ['green', 'red', 'Ыack']
Encoded values = [0, 1, 4]

Encoded values = [3, 0, 4, 1]
Decoded labels = ['yellow', 'green', 'Ыack', 'red']
```

Рис.2 Результат виконання

Після процесу кодування кожної словесої мітки була отримана своя числова форма.

Завдання 2.2. Попередня обробка нових даних

11.	-5.3	-8.9	3.0	2.9	5.1	-3.3	3.1	-2.8	-3.2	2.2	-1.4	5.1	2.0

Лістинг програми:

```
import numpy as np
from sklearn import preprocessing

input_data = np.array([[-5.3, -8.9, 3.0],
                       [2.9, 5.1, -3.3],
                       [3.1, -2.8, -3.2],
                       [2.2, -1.4, 5.1]])

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.0).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 MinMax
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print("\nMin 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)
```

```
 Binarized data:
 [[0. 0. 1.]
 [1. 1. 0.]
 [1. 0. 0.]
 [1. 0. 1.]]

BEFORE:
Mean = [ 0.725 -2.     0.4  ]
Std deviation = [3.49454933 4.97543968 3.72491611]

AFTER:
Mean = [-2.77555756e-17 -2.42861287e-17  0.00000000e+00]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.         0.         0.75      ]
 [0.97619048 1.         0.        ]
 [1.         0.43571429 0.01190476]
 [0.89285714 0.53571429 1.        ]]

l1 normalized data:
 [[-0.30813953 -0.51744186  0.1744186 ]
 [ 0.25663717  0.45132743 -0.2920354 ]
 [ 0.34065934 -0.30769231 -0.35164835]
 [ 0.25287356 -0.16091954  0.5862069 ]]

l2 normalized data:
 [[-0.49145755 -0.82527777  0.27818352]
 [ 0.43082507  0.75765788 -0.49024922]
 [ 0.58911518 -0.53210404 -0.6081189 ]
 [ 0.38407812 -0.24441335  0.89036291]]
```

Рисунок 3. Результат виконання

Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор

Лістинг програми:

```python
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)
```

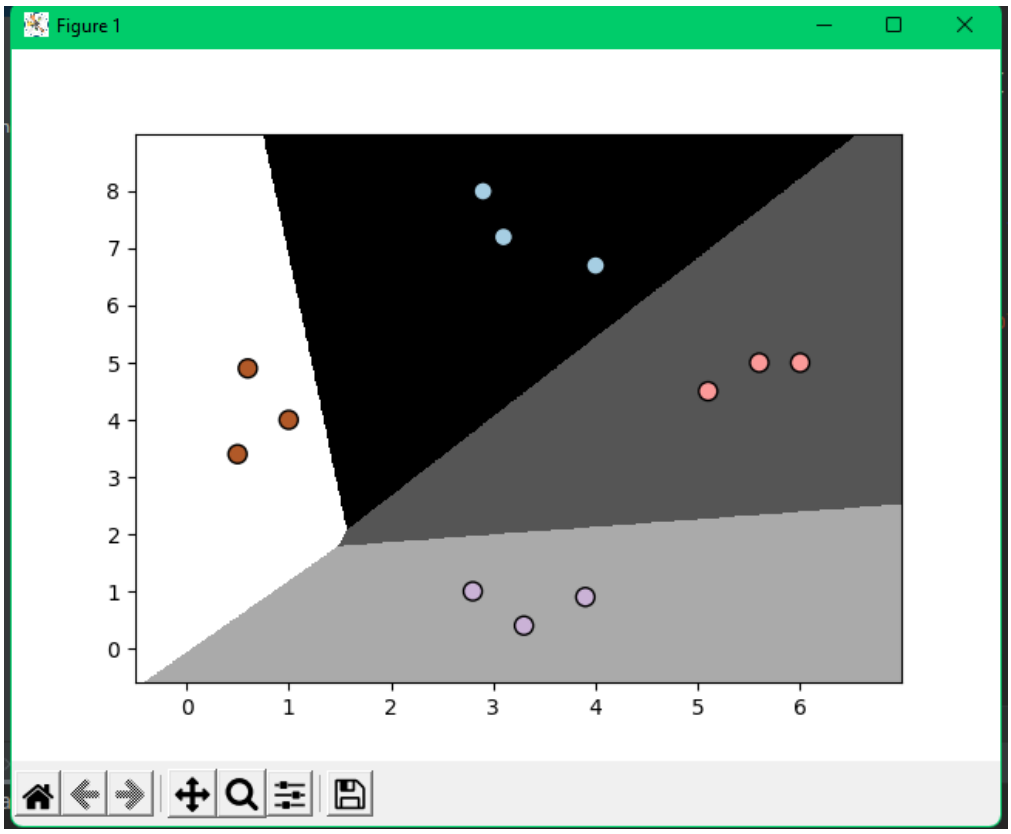

Рис. 4 Результат виконання

Завдання 2.4. Класифікація наївним байєсовським класифікатором

Лістинг програми:

```
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

# 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)
```

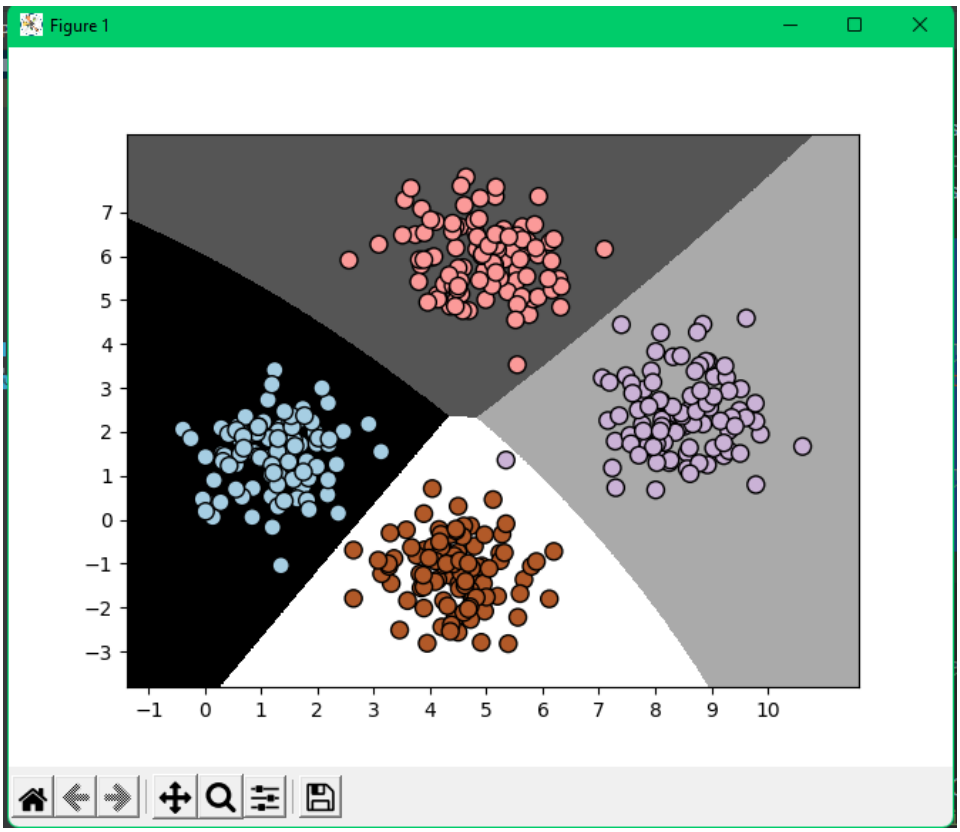

Рис. 5 Результат виконання

Лістинг програми:

```
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

# 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)

# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran-
dom_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 = train_test_split.cross_val_score(classifier, X, y, scoring='ac-
curacy', cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
precision_values = train_test_split.cross_val_score(classifier, X, y, scor-
ing='precision_weighted', cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")
recall_values = train_test_split.cross_val_score(classifier, X, y, scoring='re-
call_weighted', cv=num_folds)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) + "%")
f1_values = train_test_split.cross_val_score(classifier, X, y, scor-
ing='f1_weighted', cv=num_folds)
print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")
```

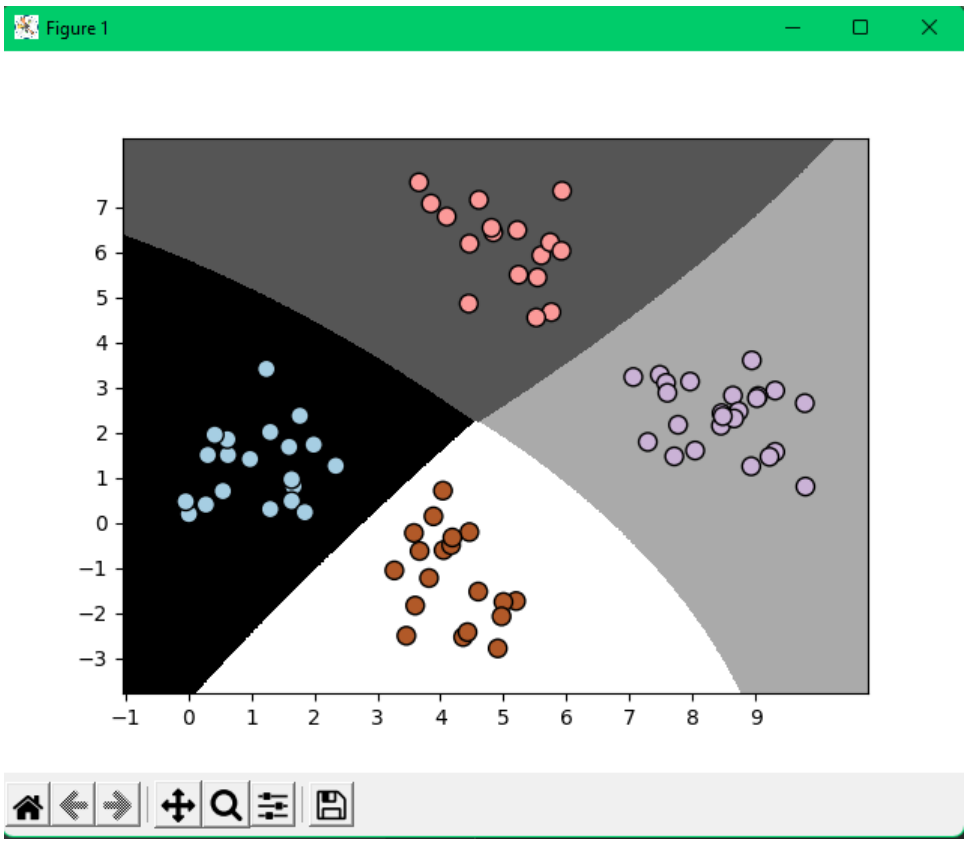

Рис. 6 Результат виконання

Завдання 2.5. Вивчити метрики якості класифікації

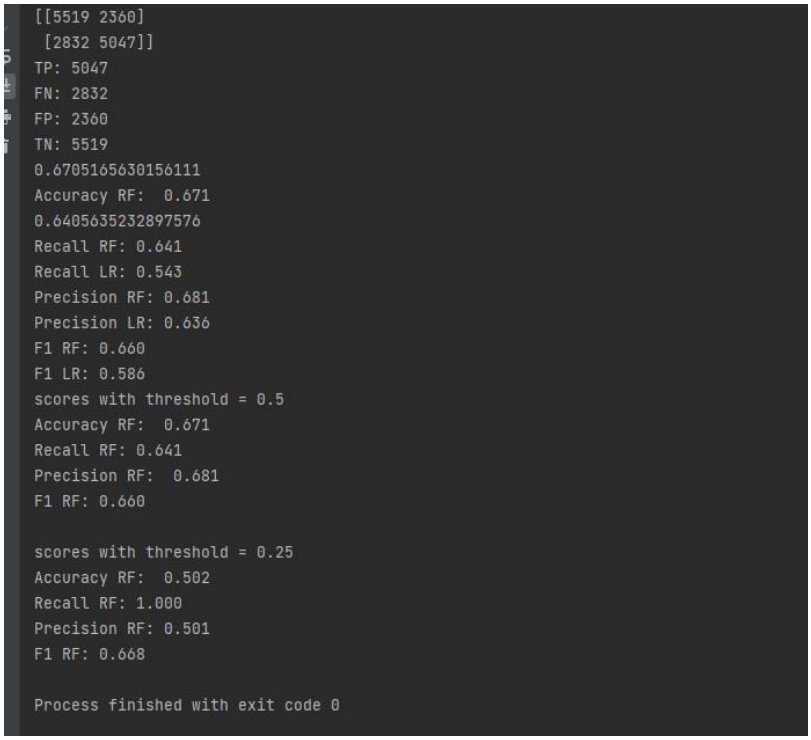

Рис. 7 Результат для порогу 0,25

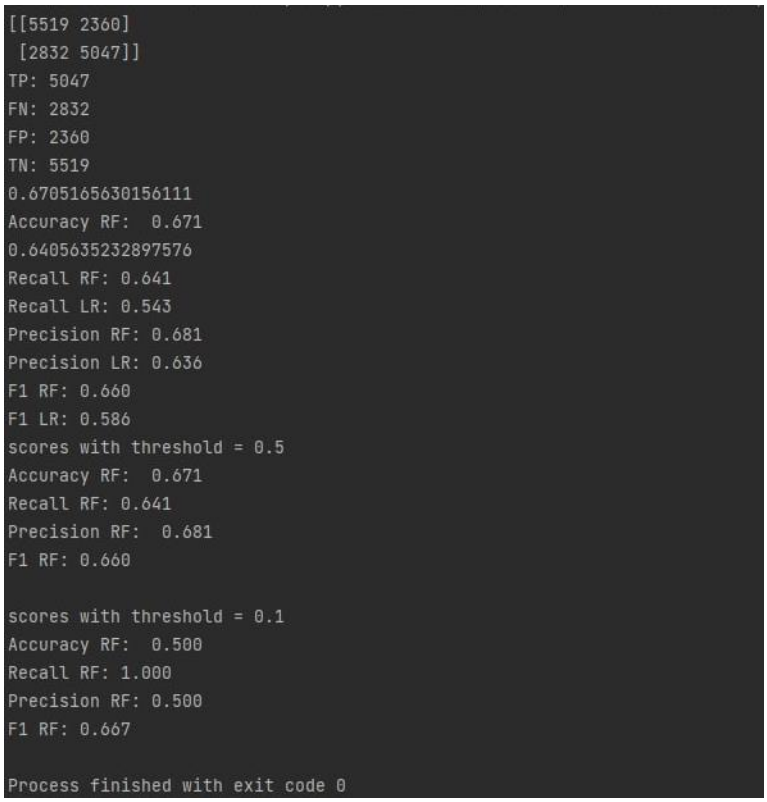

Рис. 8 Результат для порогу 0.10

```
[[5519 2360]
 [2832 5047]]
TP: 5047
FN: 2832
FP: 2360
TN: 5519
0.6705165630156111
Accuracy RF:  0.671
0.6405635232897576
Recall RF: 0.641
Recall LR: 0.543
Precision RF: 0.681
Precision LR: 0.636
F1 RF: 0.660
F1 LR: 0.586
scores with threshold = 0.5
Accuracy RF:  0.671
Recall RF: 0.641
Precision RF:  0.681
F1 RF: 0.660

scores with threshold = 0.75
Accuracy RF:  0.512
Recall RF: 0.025
Precision RF: 0.995
F1 RF: 0.049

Process finished with exit code 0
```

Рис. 9 Результат для порогу 0.75

Висновки: в результаті збільшення порогу, F1 міра зменшується.

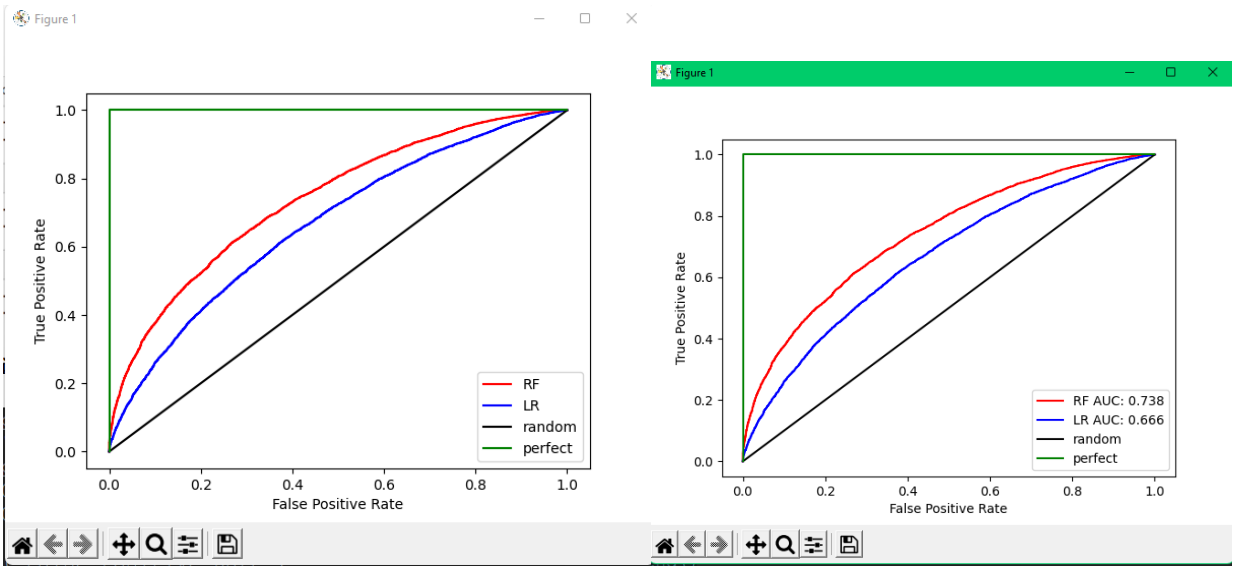

Рис. 10 ROC-крива

Як бачимо з графіку RF модель має більшу точність, аніж LR модель. Звісно, що можуть бути ситуації, коли LR має переваги перед RF, але я думаю, що важливіше, що слід враховувати, — це складність моделі. У лінійних моделей дуже мало параметрів, у RF набагато більше. Це означає, що випадкові ліси будуть легше переміщуватися, ніж LR.

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

Так як в методичних рекомендаціях до першої роботи не було наведено як формувати SVM інформацію було узято з відкритих джерел.

Лістинг програми**:**

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics

# Вхідний файл, який містить дані
from utilities import visualize_classifier

input_file = 'data_multivar_nb.txt'

# Завантаження даних із вхідного файлу
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y.astype(int),
test_size=0.2, random_state=3)

cls = svm.SVC(kernel='linear')

cls.fit(X_train, y_train)

pred = cls.predict(X_test)

print("Accuracy:", metrics.accuracy_score(y_test, y_pred=pred))

print("Precision: ", metrics.precision_score(y_test, y_pred=pred, aver-
age='macro'))

print("Recall", metrics.recall_score(y_test, y_pred=pred, average='macro'))
print(metrics.classification_report(y_test, y_pred=pred))

visualize_classifier(cls, X_test, y_test)
```

```
Accuracy: 1.0
Precision:  1.0
Recall 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00        17
           2       1.00      1.00      1.00        24
           3       1.00      1.00      1.00        19

    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80
```

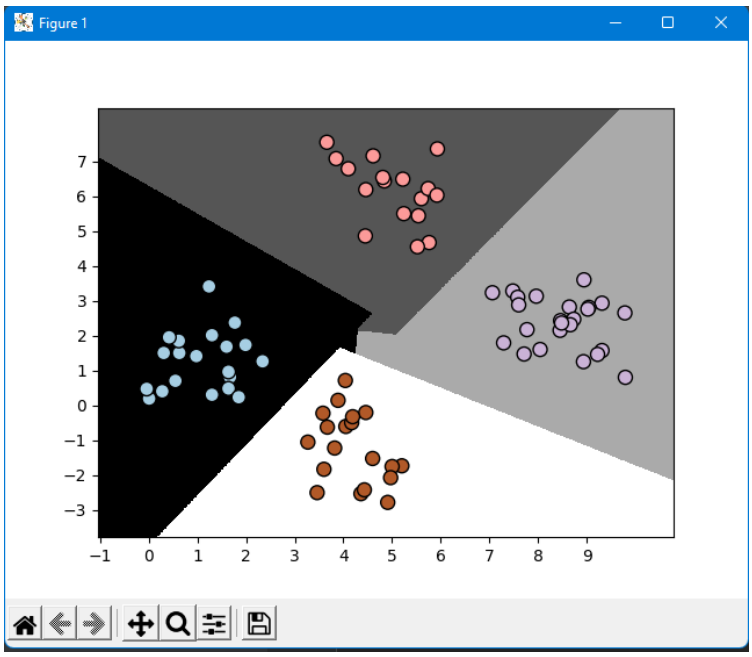

Рис. 13 Графічне відображення

Результат порівняння: Наївний класифікатор байесовського і метод опорних векторів (SVM) мають різні параметри, включаючи вибір функції ядра для кожного з них (показано в лабораторній 2). Обидва алгоритми є дуже чутливими до оптимізації параметрів, тобто вибір різних параметрів може суттєво змінити їхній вихід. Результат наразі показує, що NBC працює краще, ніж SVM, однак це вірно тільки для вибраних поточних налаштувань системи користувачем і може змінитися при зміні вхідного набору даних.

Висновок: при виконанні лабораторної роботи було досліджено попередню обробку та класифікацію даних з використанням спеціалізованих бібліотек та мови програмування Python.