ЛАБОРАТОРНА РОБОТА № 1

ОПЕРЕДНЯ 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.

```python
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 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:
[[1. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]

BEFORE:
Mean = [ 3.775 -1.15  -1.3  ]
Std deviation = [3.12039661 6.36651396 4.0620192 ]

AFTER:
Mean = [1.11022302e-16 0.00000000e+00 2.77555756e-17]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.74117647 0.39548023 1.        ]
 [0.         1.         0.        ]
 [0.6        0.5819209  0.87234043]
 [1.         0.         0.17021277]]

l1 normalized data:
 [[ 0.45132743 -0.25663717  0.2920354 ]
 [-0.0794702   0.51655629 -0.40397351]
 [ 0.609375    0.0625      0.328125  ]
 [ 0.33640553 -0.4562212  -0.20737327]]

l2 normalized data:
 [[ 0.75765788 -0.43082507  0.49024922]
 [-0.12030718  0.78199664 -0.61156148]
 [ 0.87690281  0.08993875  0.47217844]
 [ 0.55734935 -0.75585734 -0.34357152]]
```

Рис. 1. Код та результит програми

Висновок до завдання: основна відмінність полягає в тому, як обчислюється норма для нормалізації і в тому, як ці норми впливають на розподіл даних. L1-нормалізація призводить до розподілення даних більш рівномірно, тоді як L2-нормалізація спрямована на зменшення впливу великих відхилень.

python 1.py LR_1_task_1.py

```python
import numpy as np
from sklearn import preprocessing

# Надання позначок вхідних даних
input_labels = ['red', 'black', '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 =", 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:
black --> 0
black --> 1
green --> 2
red --> 3
white --> 4
yellow --> 5

Labels = ['green', 'red', 'black']
Encoded values = [2, 3, 1]

Encoded values = [3, 0, 4, 1]
Decoded labels = ['red', 'black', 'white', 'black']
```

Рис. 2. Код та результит програми

LR_1_task_2.py

```python
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], [4.2, -1.4, 6.1]])

# 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))
```

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

BEFORE:
Mean = [ 1.225 -2.     0.65 ]
Std deviation = [3.79958879 4.97543968 4.05123438]

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

Min max scaled data:
 [[0.         0.         0.67021277]
 [0.86315789 1.         0.        ]
 [0.88421053 0.43571429 0.0106383 ]
 [1.         0.53571429 1.        ]]

l1 normalized data:
 [[-0.30813953 -0.51744186  0.1744186 ]
 [ 0.25663717  0.45132743 -0.2920354 ]
 [ 0.34065934 -0.30769231 -0.35164835]
 [ 0.35897436 -0.11965812  0.52136752]]

l2 normalized data:
 [[-0.49145755 -0.82527777  0.27818352]
 [ 0.43082507  0.75765788 -0.49024922]
 [ 0.58911518 -0.53210404 -0.6081189 ]
 [ 0.55723309 -0.18574436  0.80931472]]
```

Рис. 3. Код та результит програми

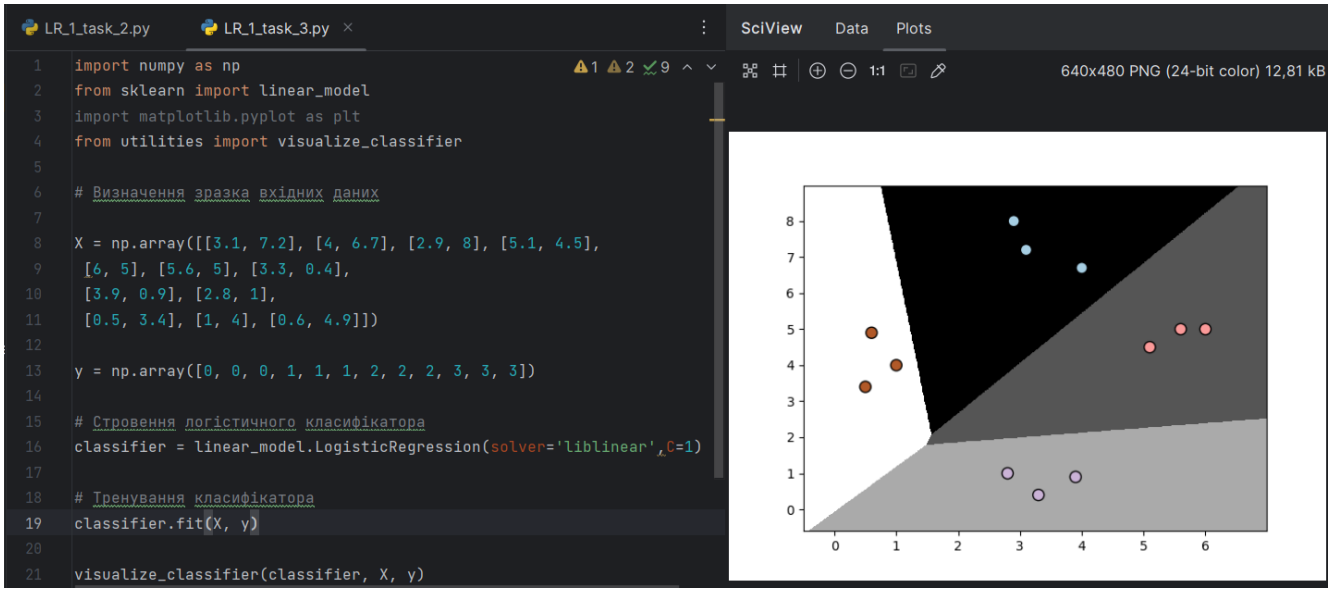


Рис. 4. Код та результит програми

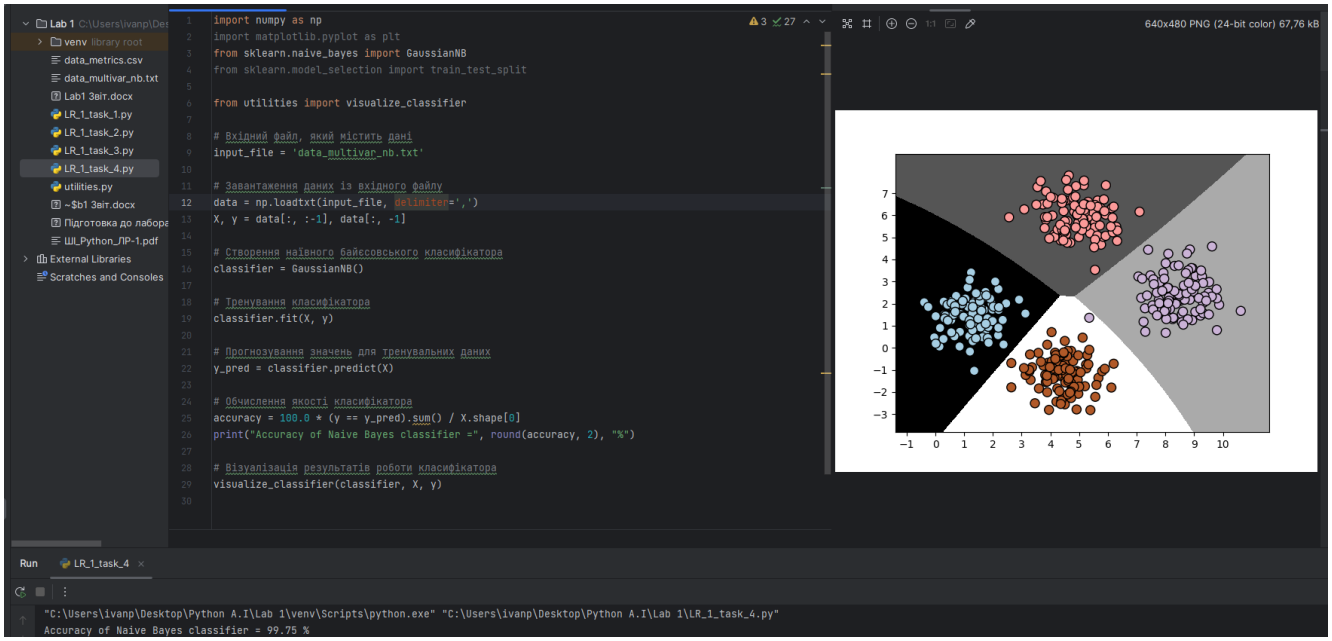


Рис. 5. Код та результит програми

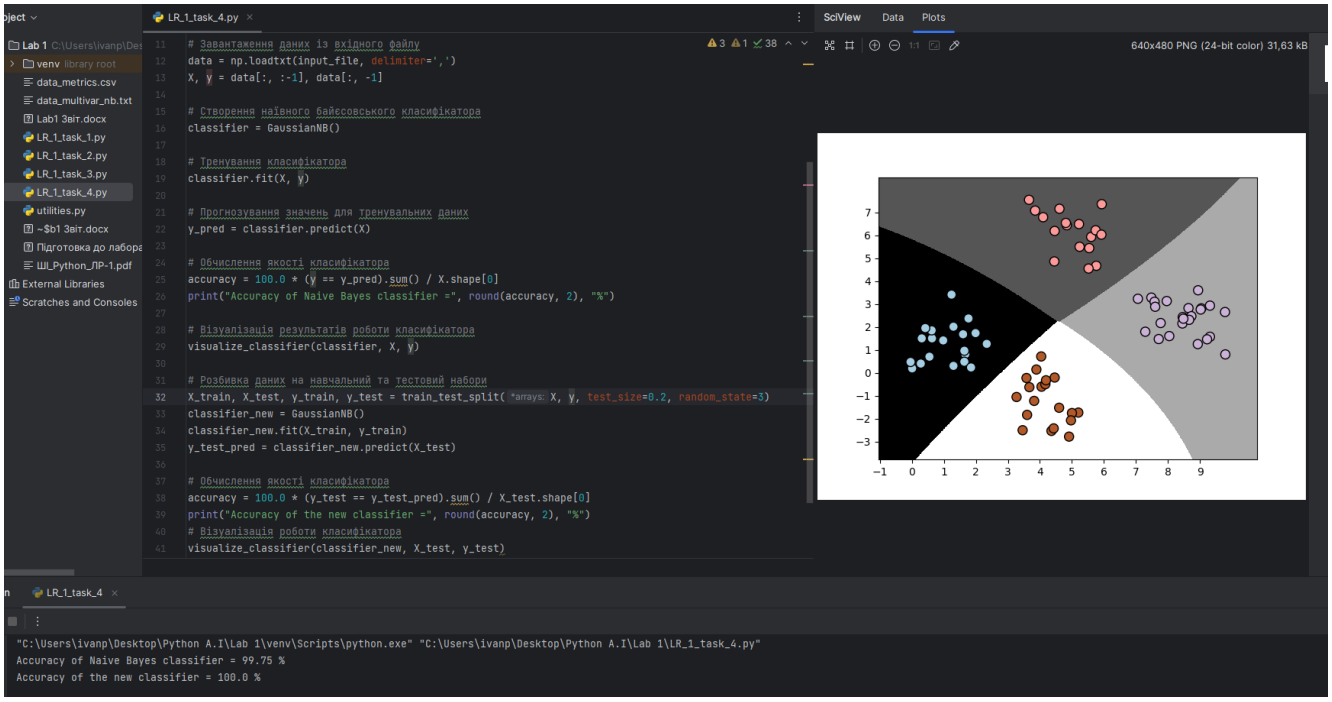


Рис. 5. Код та результит програми

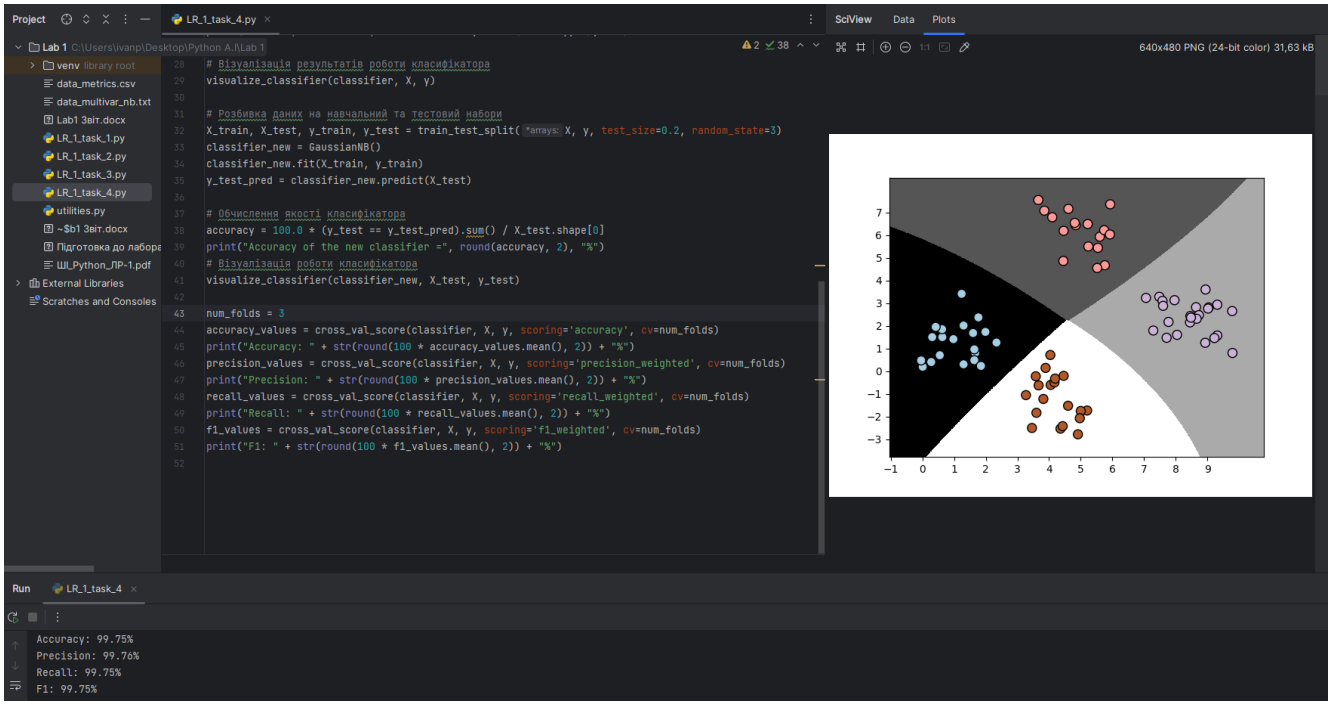


Рис. 6. Код та результит програми

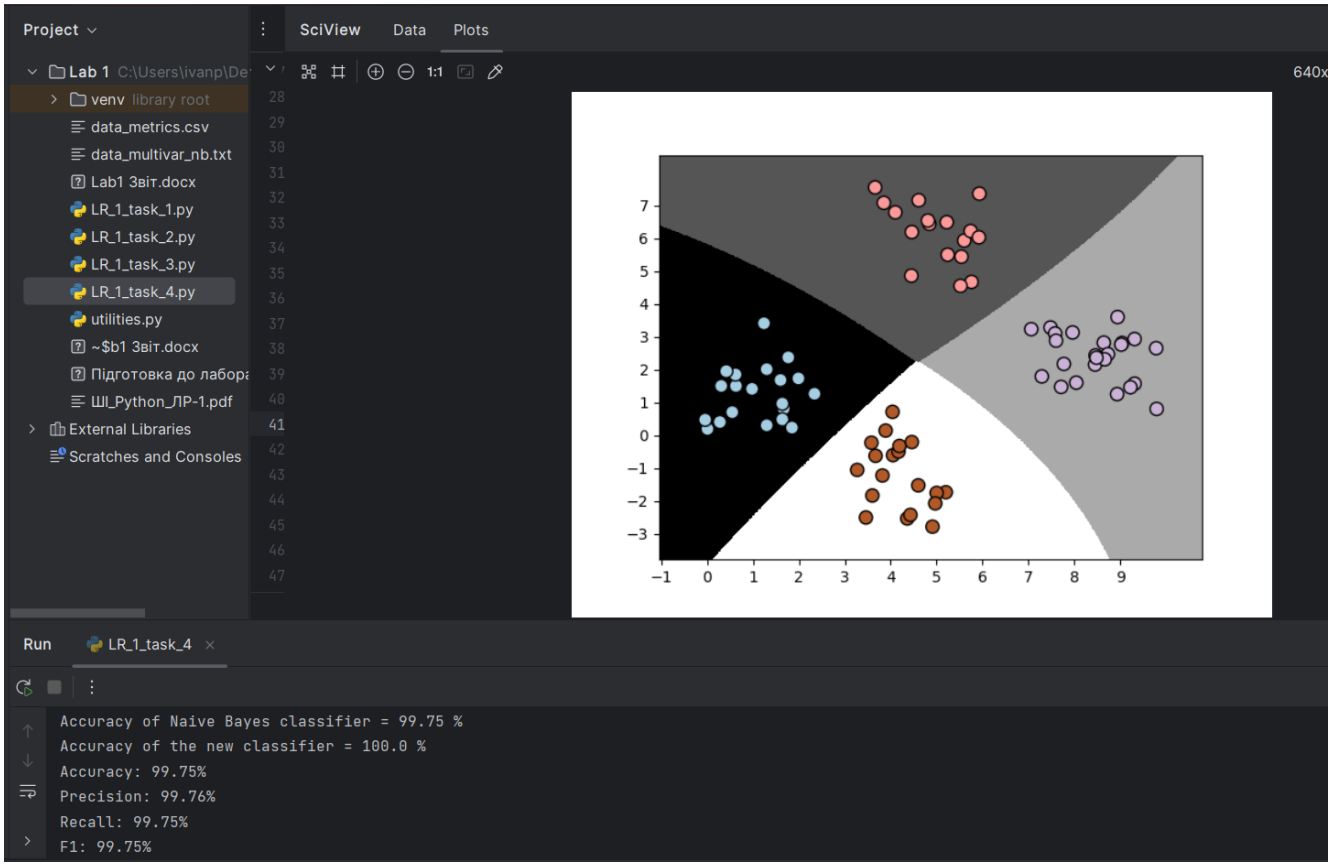
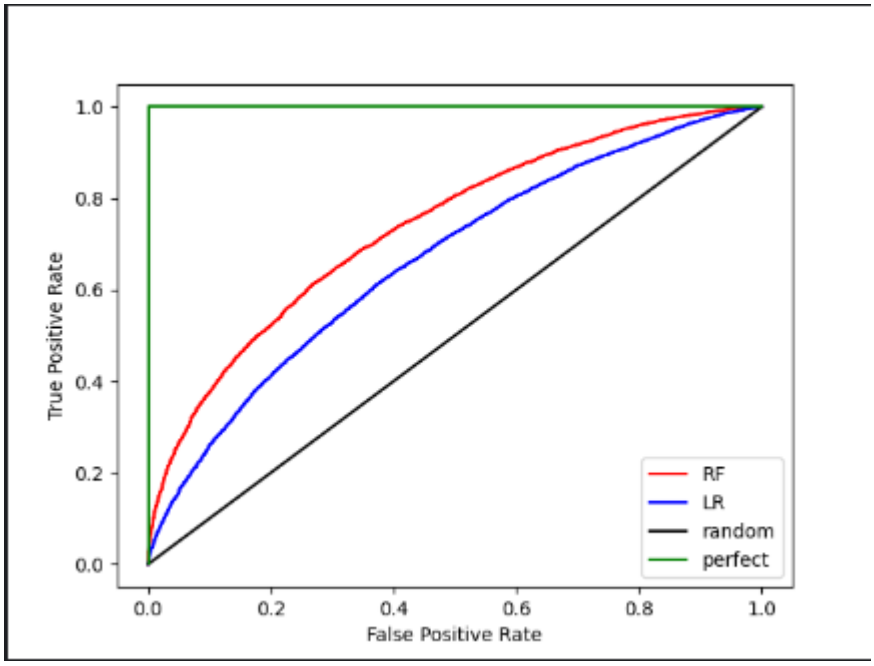
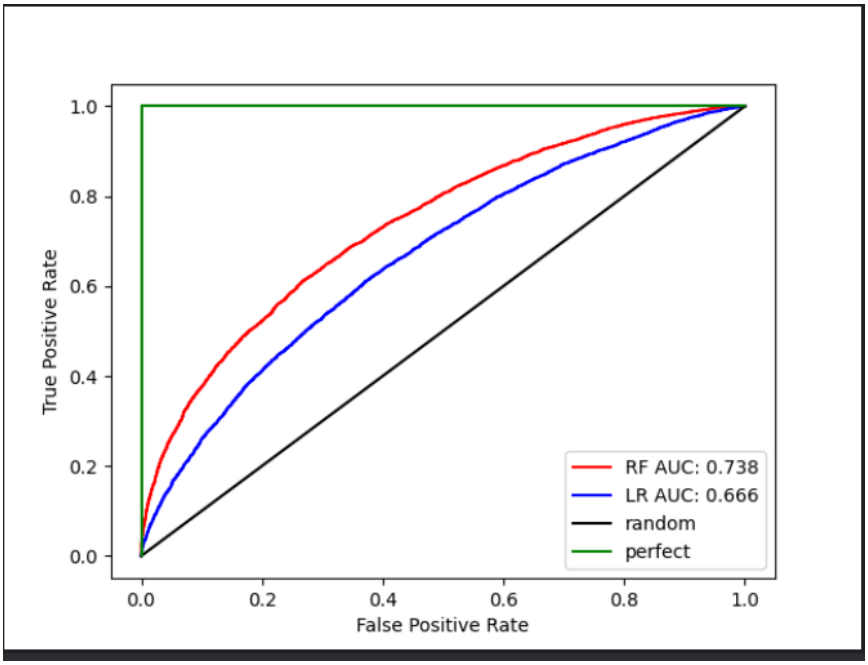


Рис. 7. Результит програми

Висновок: За цими результатами можна зробити висновок, що наївний байєсівський класифікатор добре справляється з класифікацією даних і показує стабільну та високу точність на різних наборах даних.

Висновок до пункту 2.5.1: Ці результати показують, що модель RF превосходить модель LR в усіх метриках при порозі 0.5. RF має вищу точність (Accuracy), вищий відновлення (Recall), більшу точність (Precision) і більш високу F1-меру порівняно з LR. Важливо враховувати, що вибір порогу залежить від конкретної задачі і вимог до моделі. Встановлення більшого порогу може бути важливим, якщо точність є пріоритетом і допустимі ложні позитиви обмежені, в той час як менший поріг може бути корисним, якщо важливіше уникнути ложних негативів і відновити більше позитивних класів. В даному випадку, модель RF виявилася більш ефективною з точки зору цих метрик при порозі 0.5.





Висновок до пункту 2.5.2: На основі метрик і площі під ROC-кривими можна зробити висновок, що RF є кращою моделлю у цьому конкретному наборі даних і задачі класифікації. RF має кращу точність, полноту та загальну здатність до класифікації, ніж Logistic Regression (LR).

Висновок до пункту 2.6: Кращий вибір моделі може залежати від конкретних обставин вашого завдання. У даному випадку, якщо точність є основним критерієм і

ресурси для обчислень не обмежені, то модель SVM може бути кращим варіантом. Однак, якщо у вас обмежені обчислювальні ресурси і модель байєсівського класифікатора видає задовільну точність, то вона також може бути прийнятним вибором.

Посилання на репозиторій GitHub: https://github.com/IvanPaliy/A.I.-Lab-1-IPZ-Palii.git