

ЛАБОРАТОРНА РОБОТА № 1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

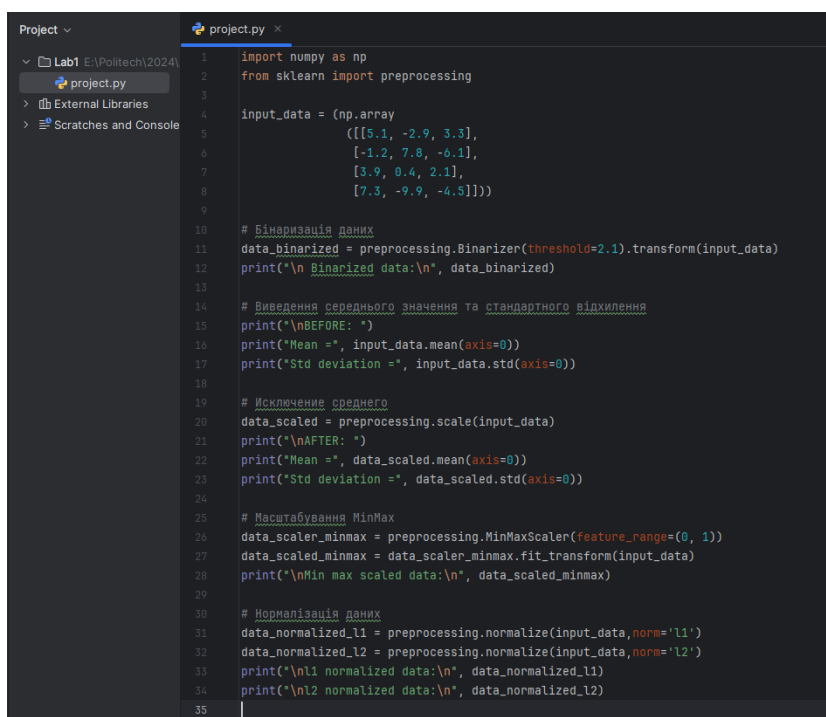
Мета роботи: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити попередню обробку та класифікацію даних.

Посилання на гітхаб:

<https://github.com/PanchukPetro/SShILabsPanchuk/tree/main/Labs/Lab1>

Завдання 2.1. Попередня обробка даних

Завдання 2.1.1 – 2.1.4



```
1 import numpy as np
2 from sklearn import preprocessing
3
4 input_data = (np.array
5               ([[5.1, -2.9, 3.3],
6                [-1.2, 7.8, -6.1],
7                [3.9, 0.4, 2.1],
8                [7.3, -9.9, -4.5]]))
9
10 # Бinarизация даних
11 data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
12 print("\n Binarized data:\n", data_binarized)
13
14 # Виведення середнього значення та стандартного відхилення
15 print("\nBEFORE: ")
16 print("Mean =", input_data.mean(axis=0))
17 print("Std deviation =", input_data.std(axis=0))
18
19 # Исключение среднего
20 data_scaled = preprocessing.scale(input_data)
21 print("\nAFTER: ")
22 print("Mean =", data_scaled.mean(axis=0))
23 print("Std deviation =", data_scaled.std(axis=0))
24
25 # Масштабирование MinMax
26 data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
27 data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
28 print("\nMin max scaled data:\n", data_scaled_minmax)
29
30 # Нормализация даних
31 data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
32 data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
33 print("\nl1 normalized data:\n", data_normalized_l1)
34 print("\nl2 normalized data:\n", data_normalized_l2)
35
```

Рис 1.1 Код програми

					ДУ «Житомирська політехніка».24.121.02.000 – ІПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Панчук П.С			СШІ Лабораторна робота №1	Лім.	Арк.
Перевір.		Голенко М.Ю					Аркушів
Керівник							1
Н. контр.							10
Зав. каф.						ФІКТ Гр. ІПЗ-21-3	

```

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 2.7755756e-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]]

Process finished with exit code 0

```

Рис 1.2 Результат

Завдання 2.1.5.

```

1  import numpy as np
2  from sklearn import preprocessing
3
4  input_labels = ['red', 'black', 'red', 'green', 'black', 'yellow', 'white']
5
6  # Створення кодувальника та встановлення відповідності
7  # між мітками та числами
8  encoder = preprocessing.LabelEncoder()
9  encoder.fit(input_labels)
10
11 # Виведення відображення
12 print("\nLabel mapping:")
13 for i, item in enumerate(encoder.classes_) : print(item, '-->', i)
14
15 # перетворення міток за допомогою кодувальника
16 test_labels = ['green', 'red', 'black']
17 encoded_values = encoder.transform(test_labels)
18 print("\nLabels =", test_labels)
19 print("Encoded values =", list(encoded_values))
20
21 # Декодування набору чисел за допомогою декодера
22 encoded_values = [3, 0, 4, 1]
23 decoded_list = encoder.inverse_transform(encoded_values)
24 print("\nEncoded values =", encoded_values)
25 print("Decoded labels =", list(decoded_list))
26

```

Рис 1.3 Код програми

		Панчук П.С			ДУ «Житомирська політехніка».24.121.02.000 – ІПЗ	Арк.
		.Голенко М.Ю				2
Змн.	Арк.	№ докум.	Підпис	Дата		

```

Label mapping:
black --> 0
green --> 1
red --> 2
white --> 3
yellow --> 4

Labels = ['green', 'red', 'black']
Encoded values = [np.int64(1), np.int64(2), np.int64(0)]

Encoded values = [3, 0, 4, 1]
Decoded labels = [np.str_('white'), np.str_('black'), np.str_('yellow'), np.str_('green')]

Process finished with exit code 0

```

Рис 1.4 Результат

В програмі ми закодували слова(в даному випадку кольори) числами. Не дивлячись на те, що деякі мітки повторюються, енкодер закодує їх один раз. Також, схоже що даний енкодер використовує алфавітний порядок. Коли ми перевіряємо роботу кодування та декодування на тестових мітках або числах, програма видає коректний результат.

		Панчук П.С			ДУ «Житомирська політехніка».24.121.02.000 – ІПЗ	Арк.
		.Голенко М.Ю				3
Змн.	Арк.	№ докум.	Підпис	Дата		

Завдання 2.2. Попередня обробка нових даних

16.	-3.3	-1.6	6.1	2.4	-1.2	4.3	-3.2	5.5	-6.1	-4.4	1.4	-1.2	2.1
-----	------	------	-----	-----	------	-----	------	-----	------	------	-----	------	-----

```

1 import numpy as np
2 from sklearn import preprocessing
3
4 #Покищо списка групи нема, використаю 16 варіант
5
6 input_data = (np.array
7               ([[-3.3, -1.6, 6.1],
8                 [2.4, -1.2, 4.3],
9                 [-3.2, 5.5, -6.1],
10                [-4.4, 1.4, -1.2]]))
11
12 # Бінаризація даних
13 data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
14 print("\n Binarized data:\n", data_binarized)
15
16 # Виведення середнього значення та стандартного відхилення
17 print("\nBEFORE: ")
18 print("Mean =", input_data.mean(axis=0))
19 print("Std deviation =", input_data.std(axis=0))
20
21 # Исключение среднего
22 data_scaled = preprocessing.scale(input_data)
23 print("\nAFTER: ")
24 print("Mean =", data_scaled.mean(axis=0))
25 print("Std deviation =", data_scaled.std(axis=0))
26
27 # Масштабування MinMax
28 data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
29 data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
30 print("\nMin max scaled data:\n", data_scaled_minmax)
31
32 # Нормалізація даних
33 data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
34 data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
35 print("\nl1 normalized data:\n", data_normalized_l1)
36 print("\nl2 normalized data:\n", data_normalized_l2)

```

Рис 2.1 Код програми

```

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

BEFORE:
Mean = [-2.125  1.025  0.775]
Std deviation = [2.65459507 2.82875856 4.79446295]

AFTER:
Mean = [ 2.77555756e-17 -5.55111512e-17  6.93889390e-17]
Std deviation = [1. 1. 1.]

Min max scaled data:
[[0.16176471 0.         1.         ]
 [1.         0.05633803 0.85245902]
 [0.17647059 1.         0.         ]
 [0.         0.42253521 0.40163934]]

l1 normalized data:
[[-0.3      -0.14545455  0.55454545]
 [ 0.30379747 -0.15189873  0.5443038 ]
 [-0.21621622  0.37162162 -0.41216216]
 [-0.62857143  0.2       -0.17142857]]

l2 normalized data:
[[-0.46364048 -0.22479538  0.8570324 ]
 [ 0.47351004 -0.23675502  0.84837215]
 [-0.36302745  0.62395344 -0.69202108]
 [-0.92228798  0.29345527 -0.25153308]]

```

Рис 2.2 Результат

		Панчук П.С			ДУ «Житомирська політехніка».24.121.02.000 – ІПЗ	Арк.
		.Голенко М.Ю				4
Змн.	Арк.	№ докум.	Підпис	Дата		

Завдання 2.3

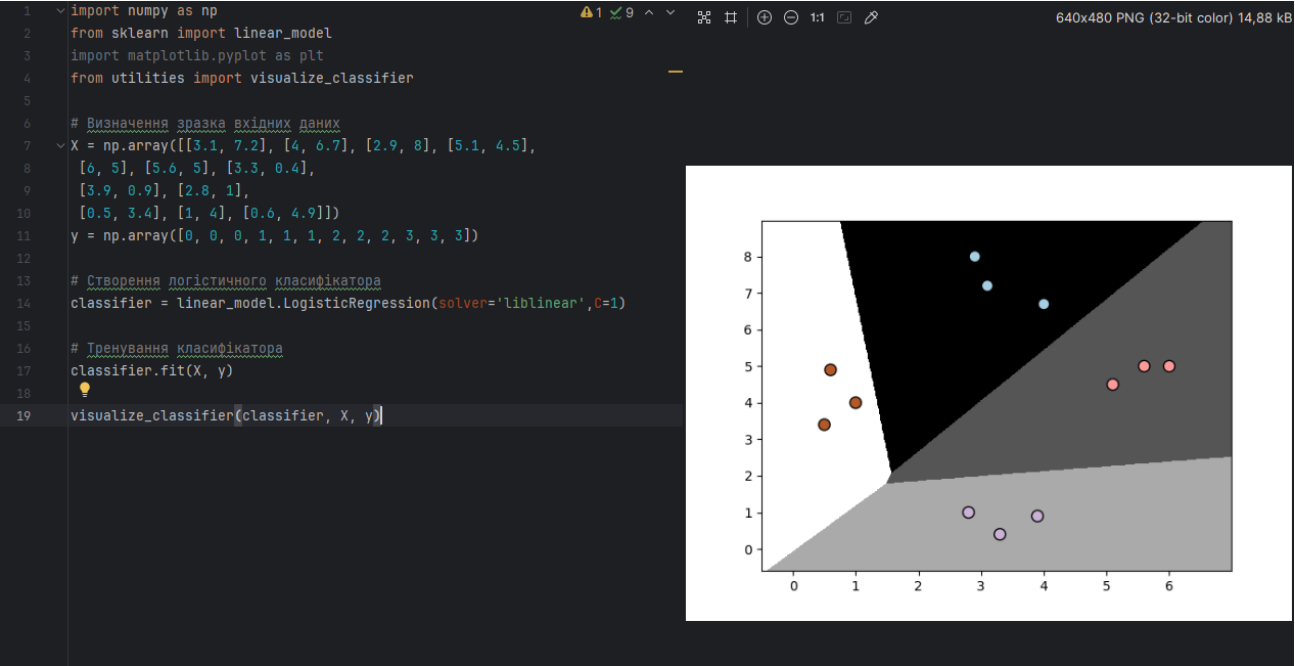


Рис 3.1 Код та результат виконання

Завдання 2.4

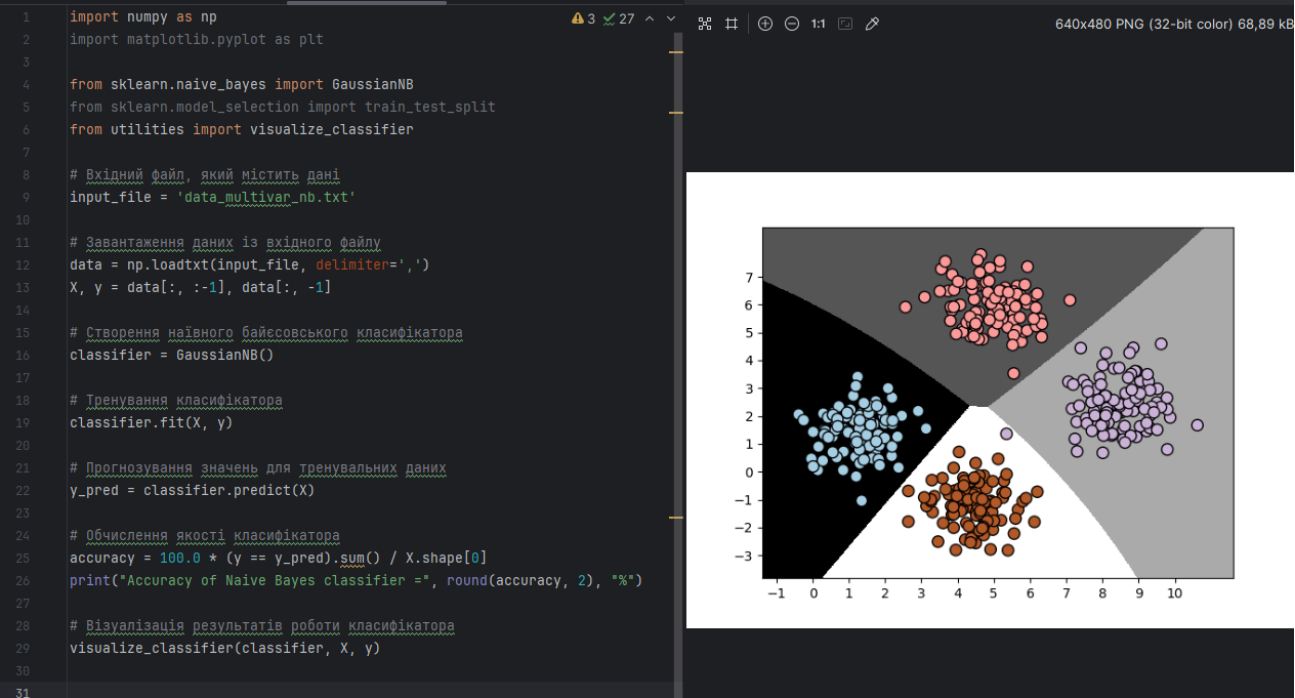
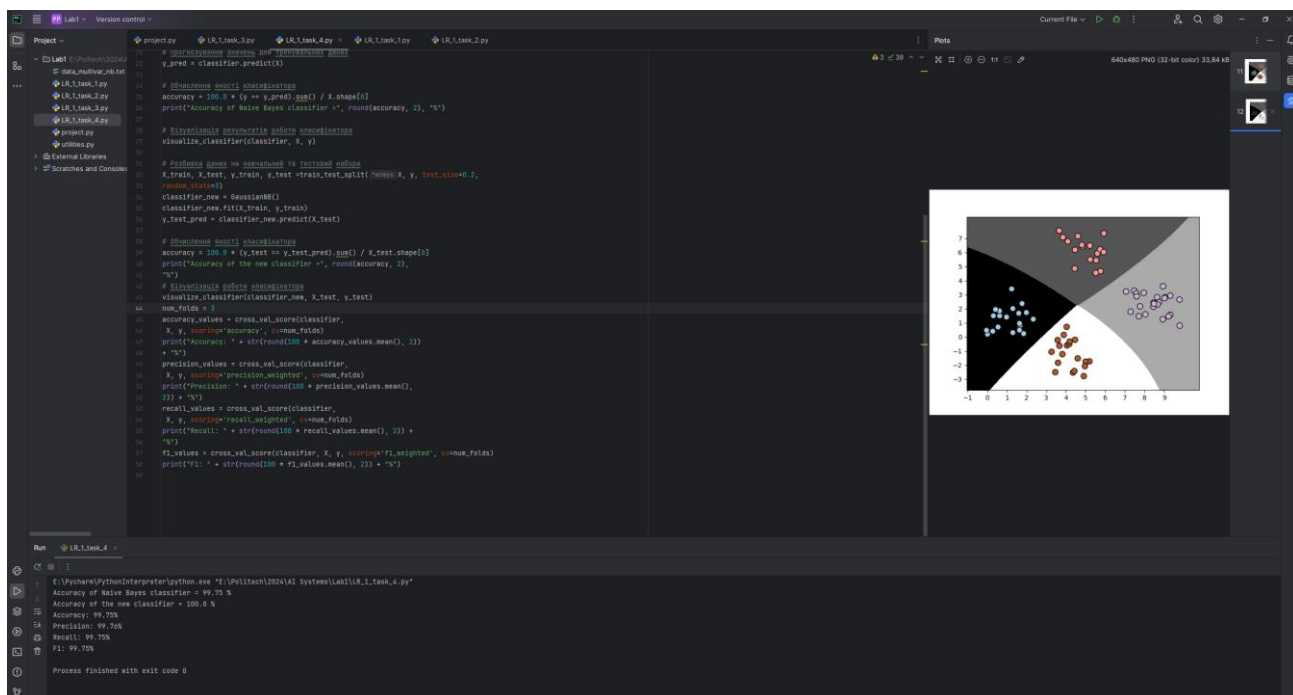
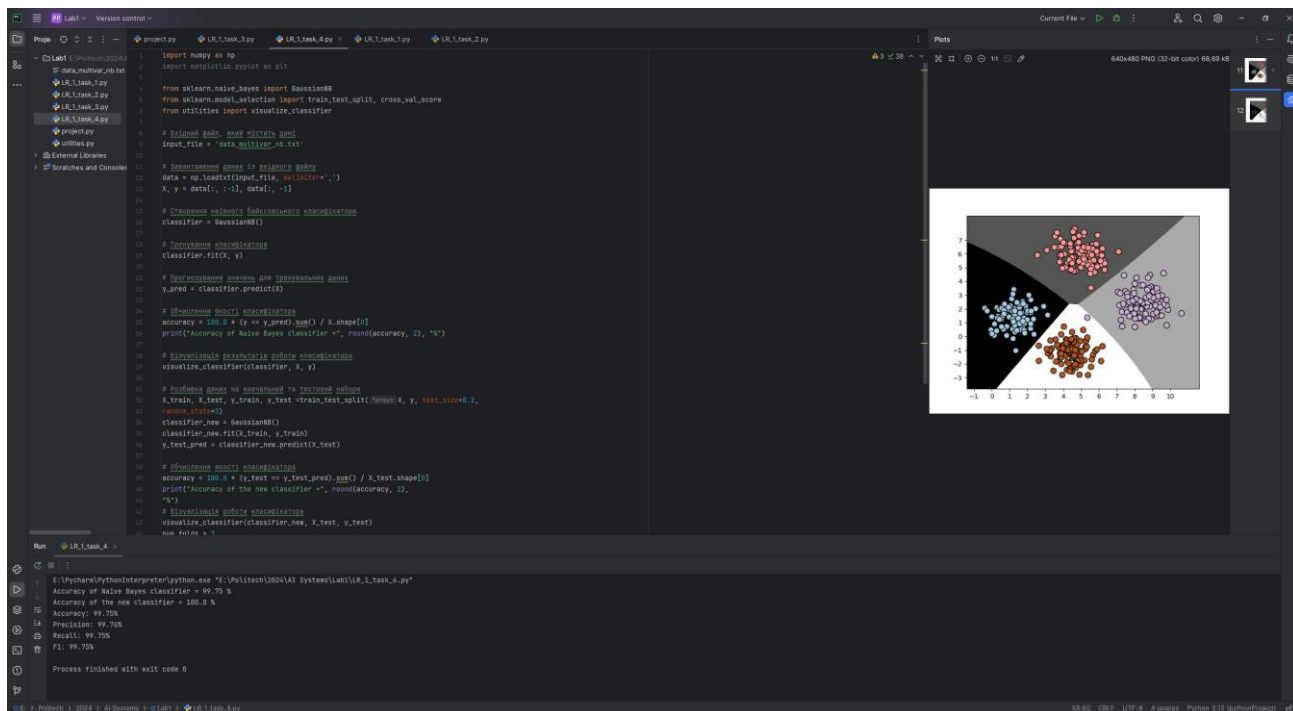


Рис 4.1 Код та результат виконання



		Панчук П.С			ДУ «Житомирська політехніка».24.121.02.000 – ІПЗ	Арк.
		.Голенко М.Ю				6
Змн.	Арк.	№ докум.	Підпис	Дата		

Завдання 2.5

Порівняння результатів для різних порогів:

accuracy(точність): при 0.5 accuracy більша ніж 0.25. Це значить що модель з порогом 0.5 правильно класифікує більше даних ніж модель з 0.25

recall(чутливість): при 0.5 recall 64.1%, при 0.25 – 100%. Модель при 0.25 порозі виявляє всі позитивні приклади, але це йде за рахунок точності

precision(Точність для позитивного) при 0.5 precision 68.1%. при 0.25 precision падає до 50.1% через зниження порогу.

f1: при 0.5 = 66%, при 0.25 = 66.8%. Незважаючи на зниження точності, F1 на приблизно такому ж рівні

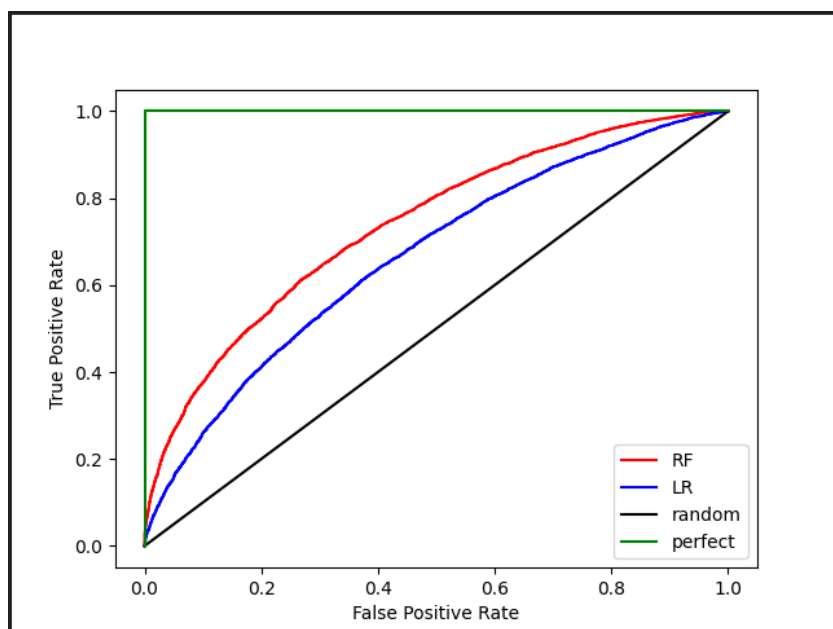


Рис 5.1 ROC крива

		Панчук П.С			ДУ «Житомирська політехніка».24.121.02.000 – ІПЗ	Арк.
		Голенко М.Ю				7
Змн.	Арк.	№ докум.	Підпис	Дата		

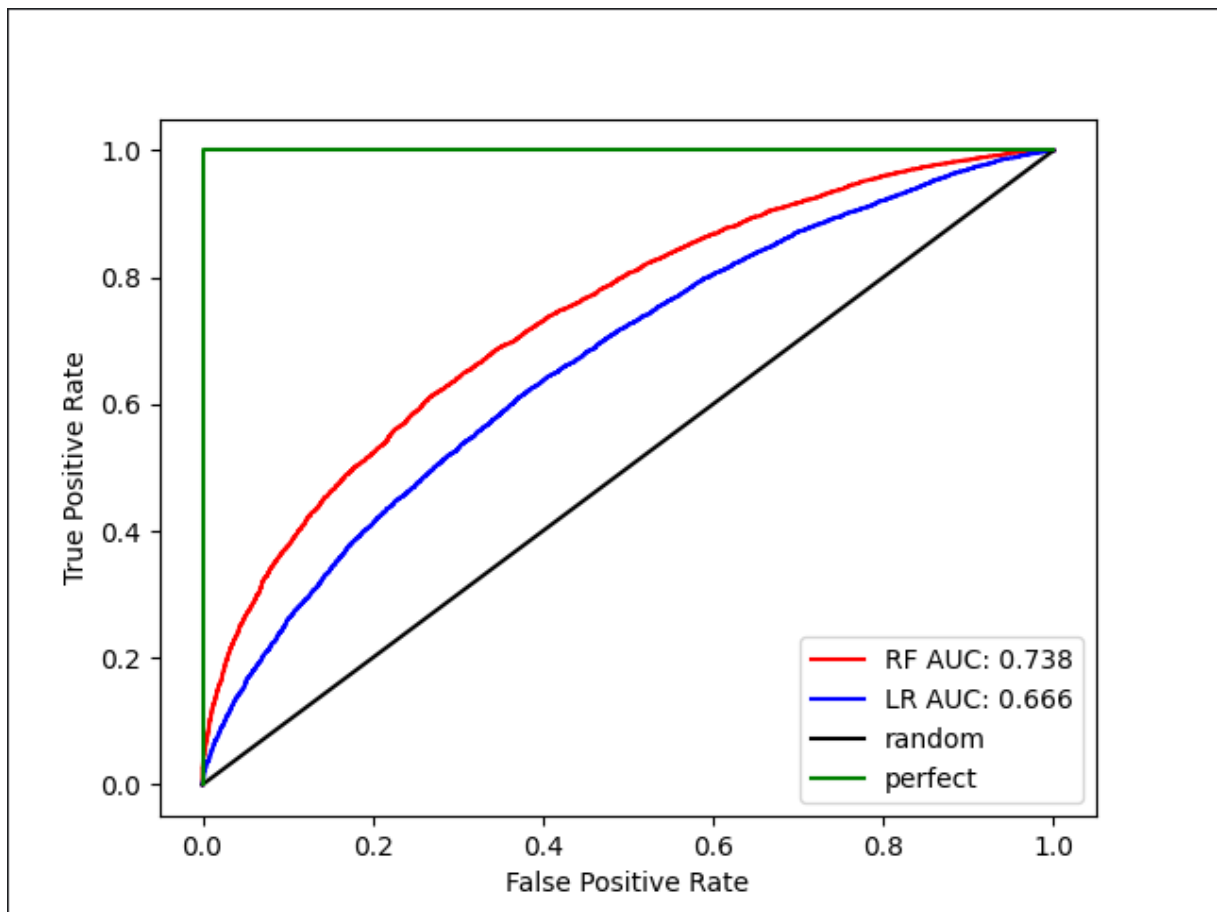


Рис 5.2 ROC крива з додаванням аус до легенди

AUC є одним з основних показників якості класифікаційної моделі, він показує здатність моделі відрізняти позитивні та негативні класи. Вищий AUC означає кращу здатність моделі до розрізнення. Тому RF краще 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.

Код програми:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score

# Завантажуємо дані з файлу
data = pd.read_csv('data_multivar_nb.txt', header=None)
X = data.iloc[:, :-1].values # ознаки (фічі)
y = data.iloc[:, -1].values # цільові мітки

# Розділяємо дані на тренувальні та тестові набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 1. Класифікація за допомогою машини опорних векторів (SVM)
svm_model = SVC()
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)

# Оцінка якості моделі SVM
accuracy_svm = accuracy_score(y_test, y_pred_svm)
recall_svm = recall_score(y_test, y_pred_svm, average='weighted')
precision_svm = precision_score(y_test, y_pred_svm, average='weighted')
f1_svm = f1_score(y_test, y_pred_svm, average='weighted')

# 2. Класифікація за допомогою наївного байєсівського класифікатора
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)

# Оцінка якості моделі Наївного Байєса
accuracy_nb = accuracy_score(y_test, y_pred_nb)
recall_nb = recall_score(y_test, y_pred_nb, average='weighted')
precision_nb = precision_score(y_test, y_pred_nb, average='weighted')
f1_nb = f1_score(y_test, y_pred_nb, average='weighted')

# Виведення результатів
print("SVM Classifier Results:")
print(f"Accuracy: {accuracy_svm:.3f}")
print(f"Recall: {recall_svm:.3f}")
print(f"Precision: {precision_svm:.3f}")
print(f"F1 Score: {f1_svm:.3f}")
print("")

print("Naive Bayes Classifier Results:")
print(f"Accuracy: {accuracy_nb:.3f}")
print(f"Recall: {recall_nb:.3f}")
print(f"Precision: {precision_nb:.3f}")
print(f"F1 Score: {f1_nb:.3f}")

# Порівняння результатів та висновки
if f1_svm > f1_nb:
    print("\nМодель SVM краще підходить для цієї задачі класифікації.")
elif f1_svm == f1_nb:
    print("\nМоделі показують однакову продуктивність")
else:
    print("\nМодель наївного байєсівського класифікатора краще підходить для цієї задачі класифікації.")
```

		Панчук П.С			ДУ «Житомирська політехніка».24.121.02.000 – ІПЗ	Арк.
		.Голенко М.Ю				9
Змн.	Арк.	№ докум.	Підпис	Дата		

```
E:\Pycharm\PythonInterpreter\python.exe "E:\Politech\2024\AI Systems\Lab1\LR_1_task_6.py"
SVM Classifier Results:
Accuracy: 0.992
Recall: 0.992
Precision: 0.992
F1 Score: 0.992

Naive Bayes Classifier Results:
Accuracy: 0.992
Recall: 0.992
Precision: 0.992
F1 Score: 0.992

Моделі показують однакову продуктивність
```

Рис 6.1 Результат виконання програми

Висновки:

Обидві моделі показали однакові результати. Це значить що дані мають бути легко відокремлюваними, тому обидві моделі справляються добре.

Модель SVM є більш складнішою, предназначена для більш складних завдань. Наївний байєсівський класифікатор більш простіший та швидший для тренування та застосування, також в ньому менші обчислювальні витрати.

Для даної задачі краще використати наївний бейєсівський класифікатор, він простіший і швидший.

		Панчук П.С			ДУ «Житомирська політехніка».24.121.02.000 – ІПЗ	Арк.
		.Голенко М.Ю				10
Змн.	Арк.	№ докум.	Підпис	Дата		