

# ЛАБОРАТОРНА РОБОТА № 1

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

Github: [link](#)

## Завдання 1. Попередня обробка даних

### Нормалізація:

The screenshot displays the JupyterLab environment. The top bar shows the 'lab1' project and 'Version control' options. The left sidebar contains a file explorer with the following structure:

- lab1 C:\Users\bekke\OneDrive\Desktop\AI\_Labs\lab1
  - .venv library root
  - lab1.docx
  - task1.py (selected)
  - ~\$lab1.docx
- External Libraries
- Scratches and Consoles

The main editor area shows the 'task1.py' file with the following code:

```
33 # 2.1.3
34 """
35
36 # 2.1.4
37 data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
38 data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
39 print("\nl1 normalized data:\n", data_normalized_l1)
40 print("\nl2 normalized data:\n", data_normalized_l2)
41 # 2.1.4
42
43
```

The bottom panel shows the 'Run' button and the 'task1' execution tab. The output of the script is displayed in the console:

```
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 результати нормалізації

**Висновок:** L1 та L2 нормалізації відрізняються методами обчислень за допомогою яких досягається рівність 1:

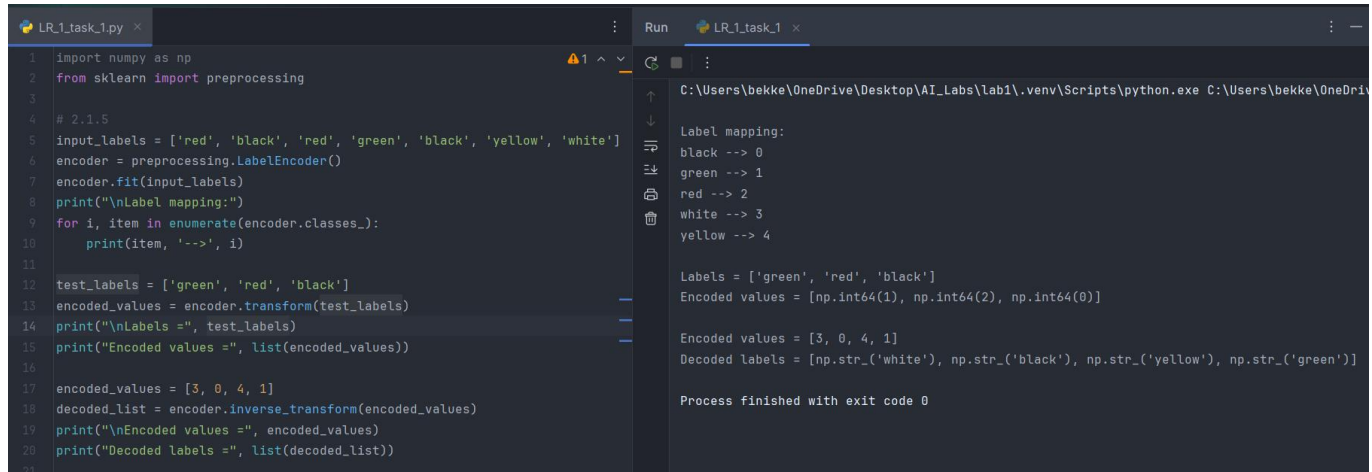
L1 використовує метод найменших абсолютних відхилень

L2 використовує метод найменших квадратів

Також L1 нормалізація вважається більш надійною, тому що вона більш стійка до викидів, але, існують такі обставини, в яких викиди грають велику роль і тому потрібно використовувати L2 нормалізацію.

Якщо порівнювати отримані значення суми рядків зі скріншоту, то у випадку з L2 нормалізацією, є такі рядки, у яких значення рівне не точно 1, а, наприклад, 0.999....

## Кодування міток:



```
1 import numpy as np
2 from sklearn import preprocessing
3
4 # 2.1.5
5 input_labels = ['red', 'black', 'red', 'green', 'black', 'yellow', 'white']
6 encoder = preprocessing.LabelEncoder()
7 encoder.fit(input_labels)
8 print("\nLabel mapping:")
9 for i, item in enumerate(encoder.classes_):
10     print(item, '-->', i)
11
12 test_labels = ['green', 'red', 'black']
13 encoded_values = encoder.transform(test_labels)
14 print("\nLabels =", test_labels)
15 print("Encoded values =", list(encoded_values))
16
17 encoded_values = [3, 0, 4, 1]
18 decoded_list = encoder.inverse_transform(encoded_values)
19 print("\nEncoded values =", encoded_values)
20 print("Decoded labels =", list(decoded_list))
```

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)]

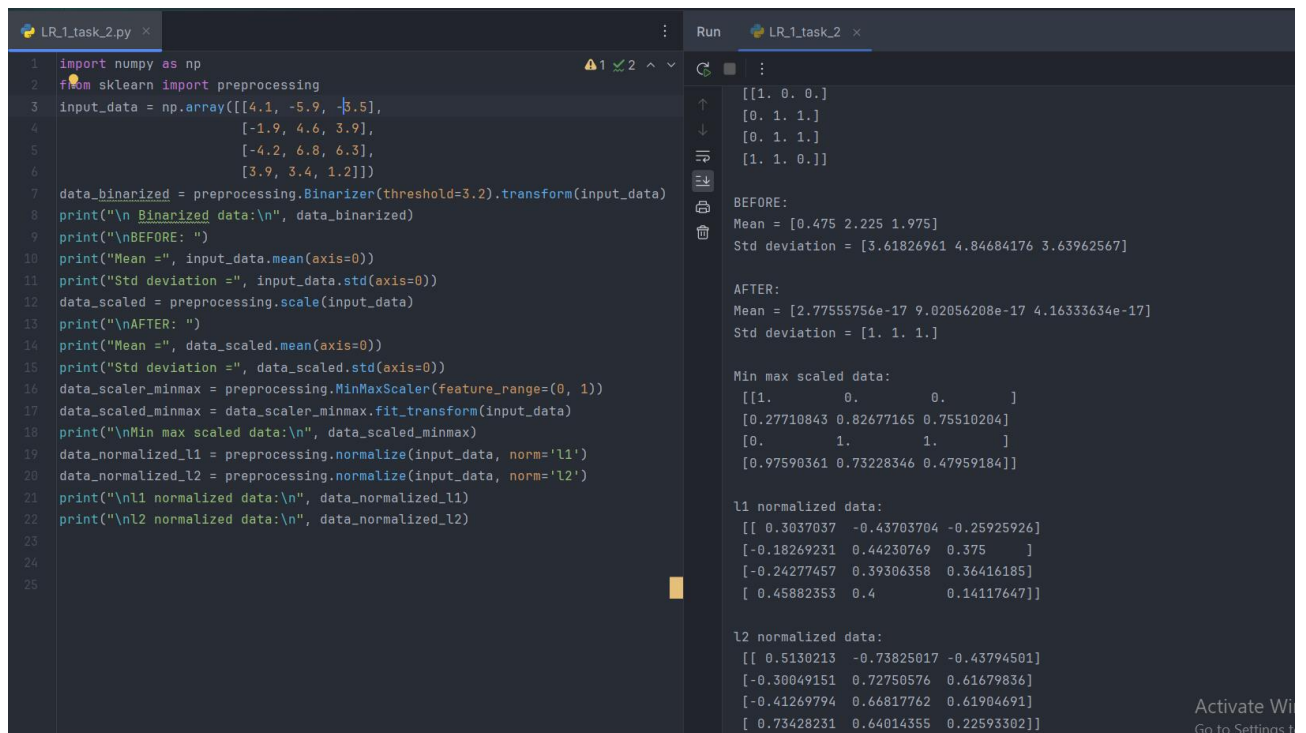
Encoded values = [3, 0, 4, 1]  
Decoded labels = [np.str\_('white'), np.str\_('black'), np.str\_('yellow'), np.str\_('green')]

Process finished with exit code 0

Рис 2. Код та результат кодування міток

## Завдання 2. Попередня обробка нових даних

2.	4.1	-5.9	-3.5	-1.9	4.6	3.9	-4.2	6.8	6.3	3.9	3.4	1.2	3.2
----	-----	------	------	------	-----	-----	------	-----	-----	-----	-----	-----	-----



```
1 import numpy as np
2 from sklearn import preprocessing
3 input_data = np.array([[4.1, -5.9, -3.5],
4                        [-1.9, 4.6, 3.9],
5                        [-4.2, 6.8, 6.3],
6                        [3.9, 3.4, 1.2]])
7 data_binarized = preprocessing.Binarizer(threshold=3.2).transform(input_data)
8 print("\n Binarized data:\n", data_binarized)
9 print("\nBEFORE: ")
10 print("Mean =", input_data.mean(axis=0))
11 print("Std deviation =", input_data.std(axis=0))
12 data_scaled = preprocessing.scale(input_data)
13 print("\nAFTER: ")
14 print("Mean =", data_scaled.mean(axis=0))
15 print("Std deviation =", data_scaled.std(axis=0))
16 data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
17 data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
18 print("\nMin max scaled data:\n", data_scaled_minmax)
19 data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
20 data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
21 print("\nl1 normalized data:\n", data_normalized_l1)
22 print("\nl2 normalized data:\n", data_normalized_l2)
```

[[1. 0. 0.]  
[0. 1. 1.]  
[0. 1. 1.]  
[1. 1. 0.]]

BEFORE:  
Mean = [0.475 2.225 1.975]  
Std deviation = [3.61826961 4.84684176 3.63962567]

AFTER:  
Mean = [2.77555756e-17 9.02056208e-17 4.16333634e-17]  
Std deviation = [1. 1. 1.]

Min max scaled data:  
[[1. 0. 0.]  
[0.27710843 0.82677165 0.75510204]  
[0. 1. 1.]  
[0.97590361 0.73228346 0.47959184]]

l1 normalized data:  
[[ 0.3037037 -0.43703704 -0.25925926]  
[-0.18269231 0.44230769 0.375 ]  
[-0.24277457 0.39306358 0.36416185]  
[ 0.45882353 0.4 0.14117647]]

l2 normalized data:  
[[ 0.5130213 -0.73825017 -0.43794501]  
[-0.30049151 0.72750576 0.61679836]  
[-0.41269794 0.66817762 0.61904691]  
[ 0.73428231 0.64014355 0.22593302]]

Рис 3. Код та результат обробки нових даних

### Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор

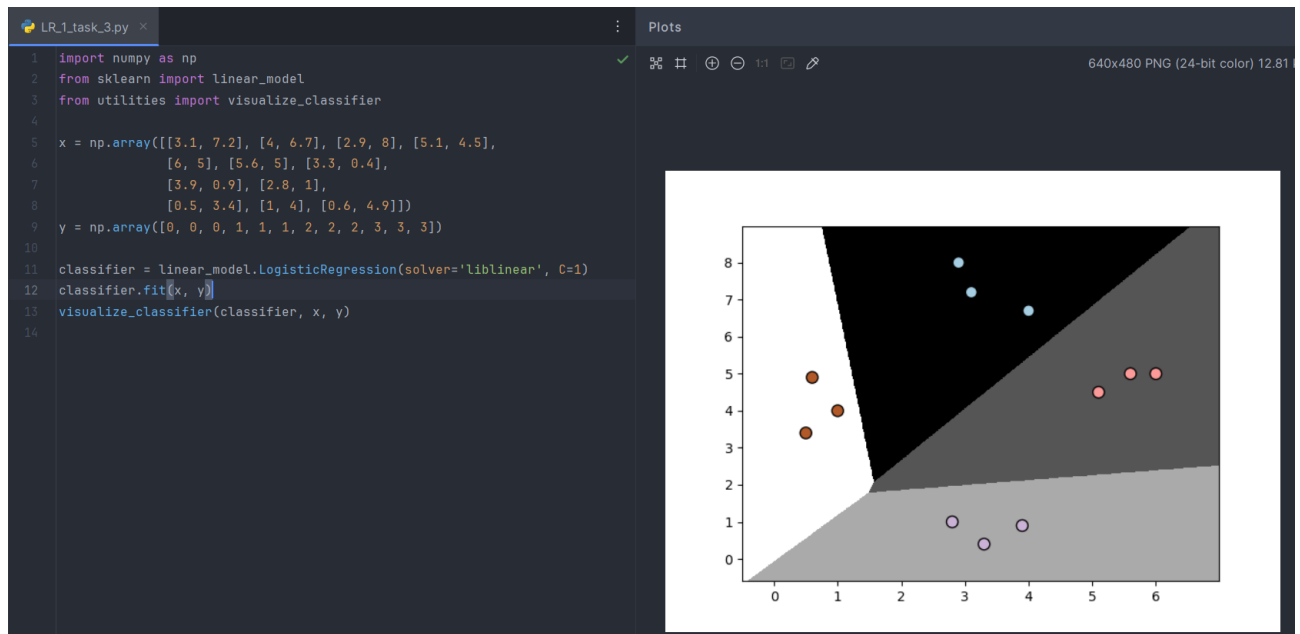


Рис 4. Код та результат класифікації

### Завдання 2.4. Класифікація наївним байєсовським класифікатором

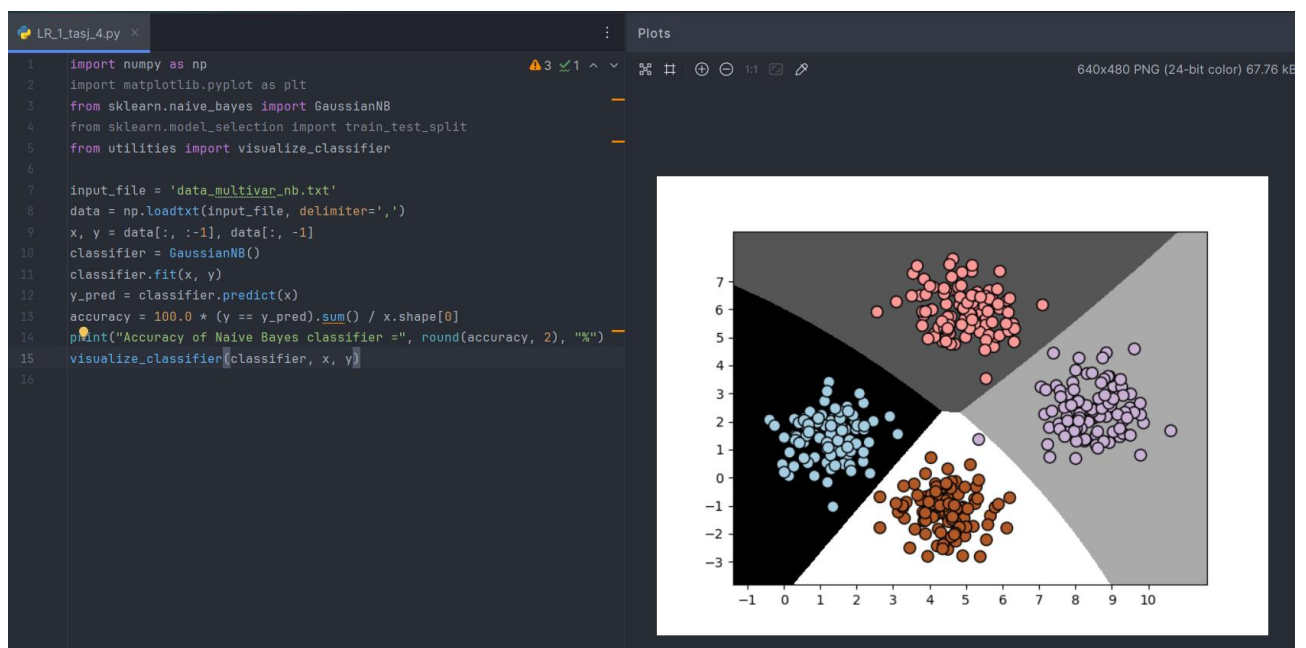


Рис 5. Код та результат класифікації

**Класифікація наївним байєсовським класифікатором з тестовими даними та даними для тренування:**

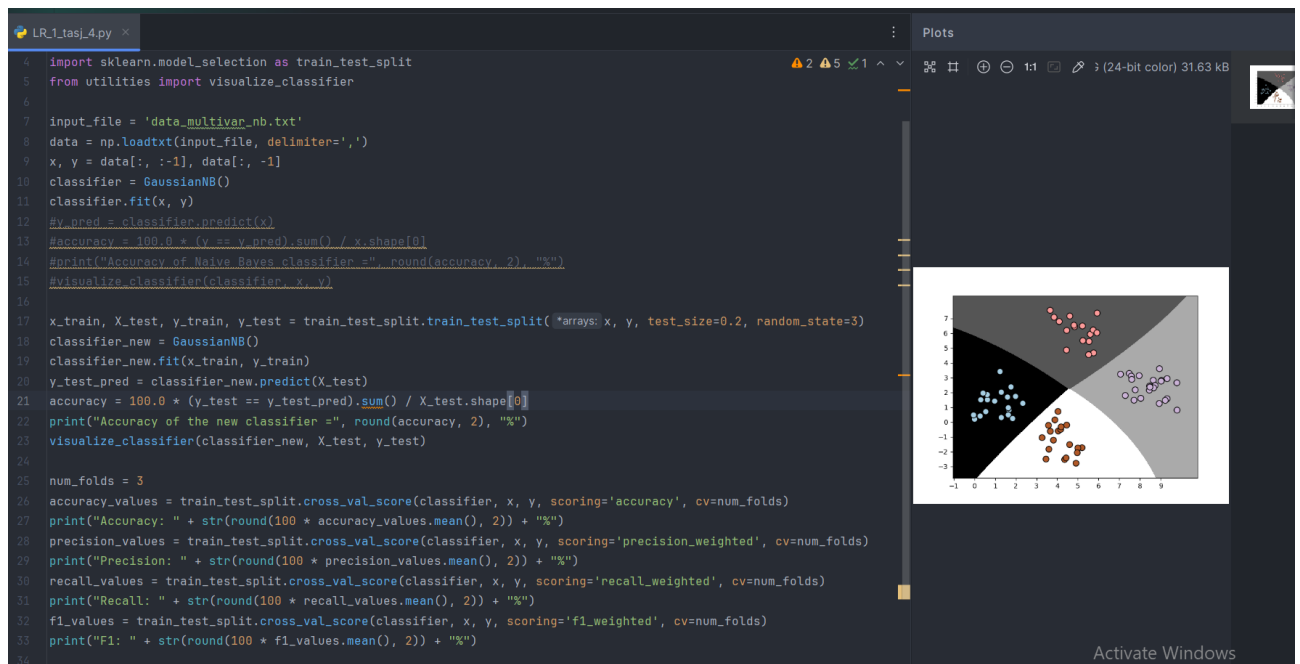


Рис 6. Код та результат класифікації з тестовими даними

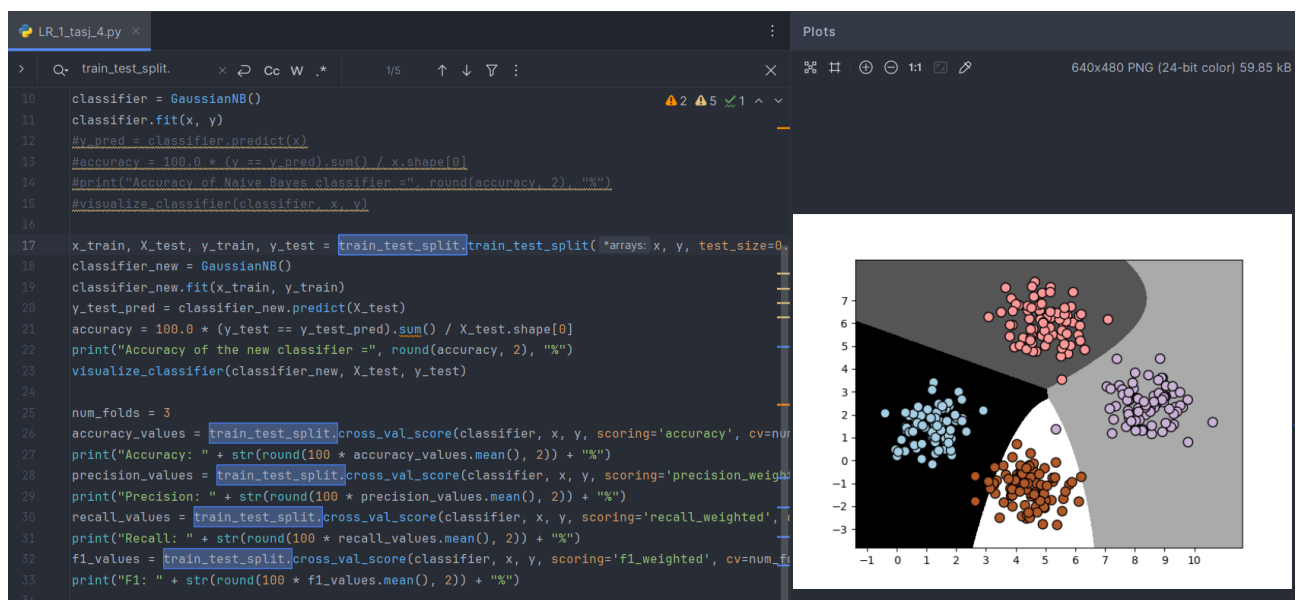


Рис 7. Код та результат класифікації з іншими параметрами

**Висновок:** відповідно до результатів, я зрозумів, що класифікатор, на основі тренувальних даних, поділив область на сектори до яких заніс точки. Сектори – це класи, а точки - це дані з певними ознаками, що відносяться до певного класу.

**Зробіть ще один прогін та зображення результатів класифікації занесіть у звіт**

1. Перезапустивши код – класифікація працює без змін.
2. Якщо для тренування залишити тільки 20%, змінивши `test_size = 0.8`, то можна помітити, що класифікатор часто невірно відносить дані/точки до класів, мабуть, це відноситься до пояснень з лабораторної:  
«Наприклад, ми можемо вважати тварину гепардом, якщо воно має плямисту шкіру, чотири лапи та хвіст і розвиває швидкість, рівну приблизно 70 миль на годину.»  
Скоріше за все, точка одного кольору, потрапляє до точок зовсім іншого кольору, по тій причині, що вона має багато спільних ознак з цими точками/даними, тому класифікатор помилково відносить її до класу, якому вона насправді не належить.  
Також межі класів не такі чіткі, як у випадку з `test_size = 0.2`, що, мабуть, також в якійсь мірі відноситься до наївності класифікації.
3. Я не впевнений, щодо попередніх висновків, тому що, можливо, тренувальних даних замало, щоб класифікувати точки правильно, а можливо обидва фактори впливають на невірну класифікацію: не досконалість класифікатора та замала кількість тестових даних.
4. Якщо прибрати `random_state` і перезапустити програму, то також виникають попередньо описані девіації, можливо вони також пов'язані з наївною роботою класифікатора і отриманий результат також репрезентує приклад з гепардом.

**Завдання 2.5.** Вивчити метрики якості класифікації

**F1\_score.** Порівняйте результати для різних порогів та зробіть висновки

**Висновок:** зменшення значення `threshold` певною мірою впливає на результат розрахунків:

Значення `assurasy` зменшилось, що свідчить про те, що частина правильно спрогнозованих вибірок також зменшилась. Аналогічно можна зробити висновки і про `precision` значення, воно також зменшилось, що означає, що

частина очікуваних, що є позитивними - зменшилась.

F1\_score значення в обох випадках однакове, тому можна сказати, що воно майже ніяк не змінилось.

Найбільші зміни помітні в recall значенні, тепер воно стало рівним 100%, що було очікувано, тому що, відповідно до методичних рекомендацій:

«Один із способів підвищити повноту – збільшити кількість вибірок, які ви визначаєте як прогнозовані позитивні, шляхом зниження порога для прогнозованих позитивних результатів. На жаль, це також збільшить кількість хибних спрацьовувань. Інший показник продуктивності, названий точністю, враховує це»

**Отже, значення threshold суттєво пливає на розрахунки. Зменшення цього значення, призвело до збільшення recall і тим самим до зменшення ассигасу та precision, тобто модель виявляє забагато позитивних прикладів, багато з яких є помилковими.**

**Побудуйте криву ROC для кожної моделі:**

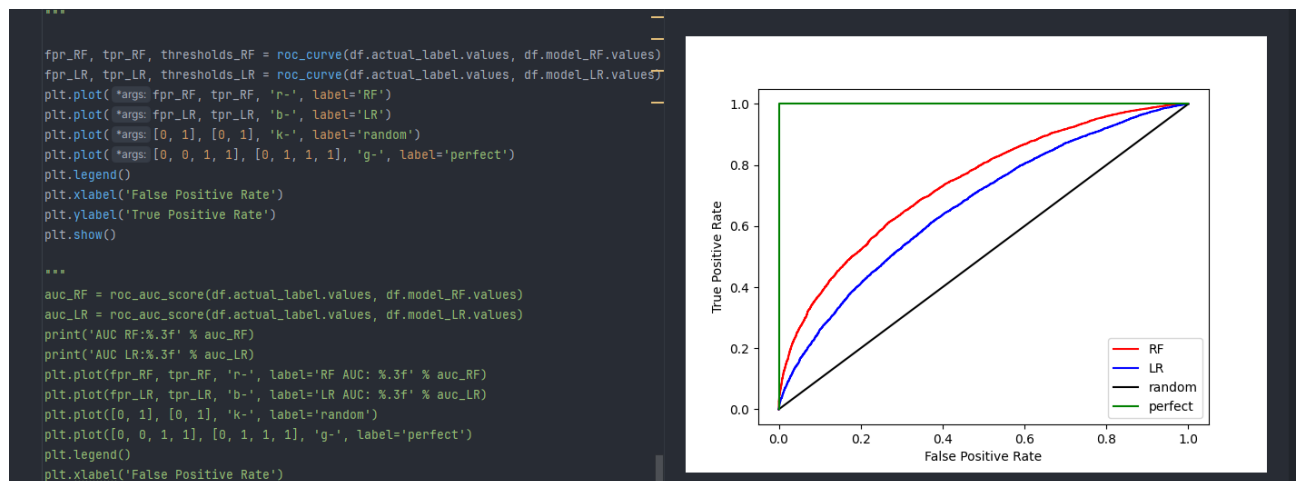


Рис 8. Крива ROC

Використання метрики площі під кривою

**Рисунок занесіть у звіт. Зробіть висновки яка з двох моделей (RF та LR) краща і чому. Висновки занесіть у звіт:**

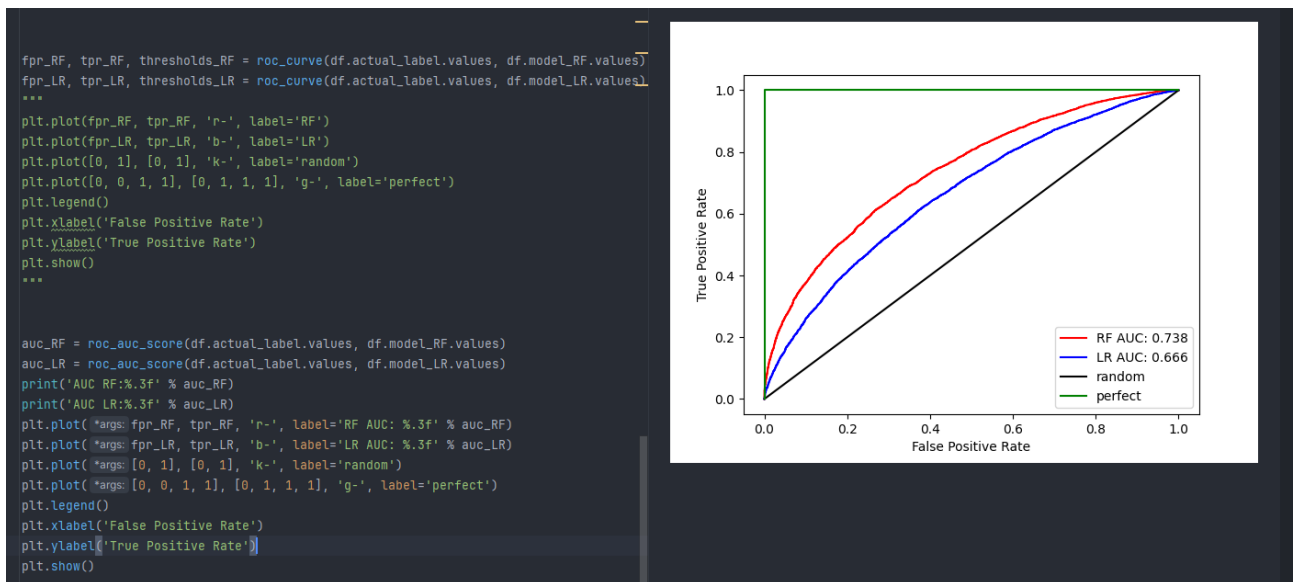


Рис 9. Використання метрики площі під кривою

**Висновок:** враховуючи результат, що ми бачимо на графіку, RF модель працює краще, тому що її AUC значення більше.

**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.

```

LR_1_task_4.py  LR_1_task_5.py  LR_1_task_6.py  data_multivar_nb.txt  Run  LR_1_task_6.py
10
11
12 1 usage
13 def get_data():
14     data = read_file()
15     return data[:, :-1], data[:, -1]
16
17 1 usage
18 def report(a, b, avg):
19     texts = ["Precision", "Recall", "F1"]
20     funcs = [precision_score, recall_score, f1_score]
21     for i in range(len(texts)):
22         print(texts[i], funcs[i](a, b, average=avg))
23     print('')
24
25 x, y = get_data()
26 X_train, X_test, y_train, y_test = train_test_split(*arrays: x, y)
27 svm_classifier = SVC(kernel='linear', probability=True)
28 svm_classifier.fit(X_train, y_train)
29 y_pred = svm_classifier.predict(X_test)
30 print("Accuracy:", accuracy_score(y_test, y_pred))
31 report(y_test, y_pred, avg: "weighted")
32 y_prob = svm_classifier.predict_proba(X_test)
33 print("Roc auc score", roc_auc_score(y_test, y_prob, multi_class="ovr"))

```

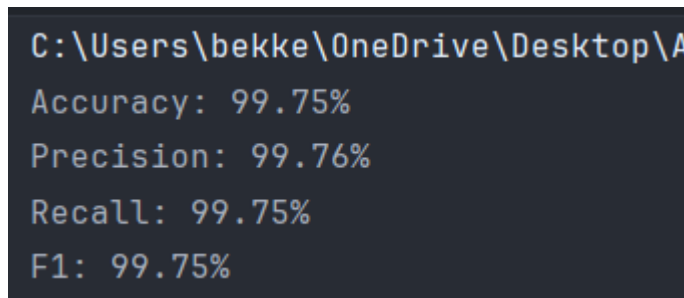
```

C:\Users\bekke\OneDrive\Desktop\AI_La
Accuracy: 1.0
Precision 1.0
Recall 1.0
F1 1.0
Roc auc score 1.0
Process finished with exit code 0

```

Рис 10. Код програми та результати метрик

**Висновок:** оскільки наївна модель, показує майже 100%, які ми маємо у SVM моделі, можна зробити висновок, що, в даному випадку, можна обрати будь-яку з цих моделей. Звісно, SVM модель має кращі показники, ніж наївна, тому краще обрати її, але різниця не суттєва, тому, на мою думку, потрібно обирати модель в залежності від обставин та поставлених задач. Але, якщо робити загальний висновок, порівнюючи ці моделі, то, в більшості випадків, краще обирати SVM модель, тому що вона більш досконала у порівнянні з наивною.

A screenshot of a terminal window with a dark background and light-colored text. The text displays the file path 'C:\Users\bekke\OneDrive\Desktop\A' followed by four performance metrics: 'Accuracy: 99.75%', 'Precision: 99.76%', 'Recall: 99.75%', and 'F1: 99.75%'.

```
C:\Users\bekke\OneDrive\Desktop\A
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

Рис 11. Попередні показники наївної моделі