

## Лабораторна робота №1

**Тема:** Попередня обробка та контрольована класифікація даних.

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

### 2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ

#### Завдання 2.1. Попередня обробка даних

##### 2.1.1. Бінаризація

```
✓ [1] import numpy as np
2s    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

```
✓ [2] binarized_data = preprocessing.Binarizer(threshold=2.1).transform(input_data)
0s    print("\n Binarized data:\n", binarized_data)
```

⇒

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

##### 2.1.2. Виключення середнього

### Виключення середнього

```
✓ [6] print("\nBEFORE: ")
0s print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))

data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis=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.]
```

## 2.1.3. Масштабування

### Масштабування

```
✓ [7] data_scaled_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
0s print("\nMin max scaled data:\n", data_scaled_minmax)
data_scaled_minmax.fit_transform(input_data)
```



```
Min max scaled data:
MinMaxScaler()
array([[0.74117647, 0.39548023, 1.          ],
       [0.          , 1.          , 0.          ],
       [0.6         , 0.5819209 , 0.87234043],
       [1.          , 0.          , 0.17021277]])
```

## 2.1.4. Нормалізація

Нормалізація

```
[8] 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)
```

↕

```
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]]
```

Рис1. Бінарізація, Виключення середнього, Масштабування, Нормалізація.

## 2.1.5. Кодування міток

```
import numpy as np
from sklearn import preprocessing

input_labels = ['red', 'black', 'red', 'green', 'black',
               'yellow', 'white']

encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)

print("\nLabel mapping:")
for i, item in enumerate(encoder.classes_) :
    print(item, '-->', i)
```

↕

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

```
[ ] test_labels = ['green', 'red', 'black']
    encoded_values = encoder.transform(test_labels)
    print("\nLabels =", test_labels)
    print("Encoded values =", list(encoded_values))
```



```
Labels = ['green', 'red', 'black']
Encoded values = [1, 2, 0]
```

```
[ ] encoded_values = [3, 0, 4, 1]
    decoded_list = encoder.inverse_transform(encoded_values)
    print("\nEncoded values =", encoded_values)
    print("Decoded labels =", list(decoded_list))
```



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

*Рис 2. Кодування міток.*

## **Завдання 2.2. Попередня обробка нових даних**

Варіант обирається відповідно номера за списком групи відповідно до таблиці 1. Варіант №12


|     |      |     |     |      |      |     |      |      |      |      |     |      |     |
|-----|------|-----|-----|------|------|-----|------|------|------|------|-----|------|-----|
| 12. | -1.3 | 3.9 | 4.5 | -5.3 | -4.2 | 3.3 | -5.2 | -6.5 | -1.1 | -5.2 | 2.6 | -2.2 | 1.8 |
|-----|------|-----|-----|------|------|-----|------|------|------|------|-----|------|-----|

Завдання згідно варіанту:

```
✓ 34 [1] import numpy as np
      from sklearn import preprocessing

      input_data = np.array([[-1.3, 3.9, 4.5],
                             [-5.3, -4.2, 3.3],
                             [-5.2, -6.5, -1.1],
                             [-5.2, 2.6, -2.2]])
```

Бінарізація

```
✓ 04  binarized_data = preprocessing.Binarizer(threshold=1.8).transform(input_data)
      print("\n Binarized data:\n", binarized_data)
```



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

Виключення середнього

```
[ ] print("\nBEFORE: ")
      print("Mean =", input_data.mean(axis=0))
      print("Std deviation =", input_data.std(axis=0))

      data_scaled = preprocessing.scale(input_data)
      print("\nAFTER: ")
      print("Mean =", data_scaled.mean(axis=0))
      print("Std deviation =", data_scaled.std(axis=0))
```



```
BEFORE:
Mean = [-4.25 -1.05  1.125]
Std deviation = [1.7036725  4.40028408  2.83405628]

AFTER:
Mean = [0.00000000e+00  5.55111512e-17  0.00000000e+00]
Std deviation = [1. 1. 1.]
```

### Масштабування

```
data_scaled_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))  
  
print("\nMin max scaled data:\n", data_scaled_minmax)  
data_scaled_minmax.fit_transform(input_data)
```



```
Min max scaled data:  
MinMaxScaler()  
array([[1.          , 1.          , 1.          ],  
       [0.          , 0.22115385, 0.82089552],  
       [0.025       , 0.          , 0.1641791  ],  
       [0.025       , 0.875       , 0.          ]])
```

### Нормалізація

```
[ ] 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)
```



```
l1 normalized data:  
[[-0.13402062  0.40206186  0.46391753]  
 [-0.4140625  -0.328125    0.2578125 ]  
 [-0.40625    -0.5078125  -0.0859375 ]  
 [-0.52       0.26       -0.22       ]]  
  
l2 normalized data:  
[[-0.21328678  0.63986035  0.7383004 ]  
 [-0.70435392 -0.55816726  0.43855999]  
 [-0.61931099 -0.77413873 -0.13100809]  
 [-0.83653629  0.41826814 -0.3539192 ]]
```

*Рис 3. Попередня обробка нових даних.*

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

```
✓ [7] import numpy as np
    from sklearn import linear_model
    import matplotlib.pyplot as plt
    from utilities import visualize_classifier

    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])

✓ [8] # Створення логістичного класифікатора
    classifier = linear_model.LogisticRegression(solver='liblinear', C=1)

✓ [9] # Тренування класифікатора
    classifier.fit(X, y)
```

LogisticRegression

LogisticRegression(C=1, solver='liblinear')

Рис 4. Класифікація логістичною регресією або логістичний класифікатор.

## Завдання 2.4. Класифікація найвним байєсовським класифікатором

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
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)
```



GaussianNB ⓘ ⓘ  
GaussianNB()

```
[ ] # 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), "%")
```

```
↔ Accuracy of Naive Bayes classifier = 99.75 %
```



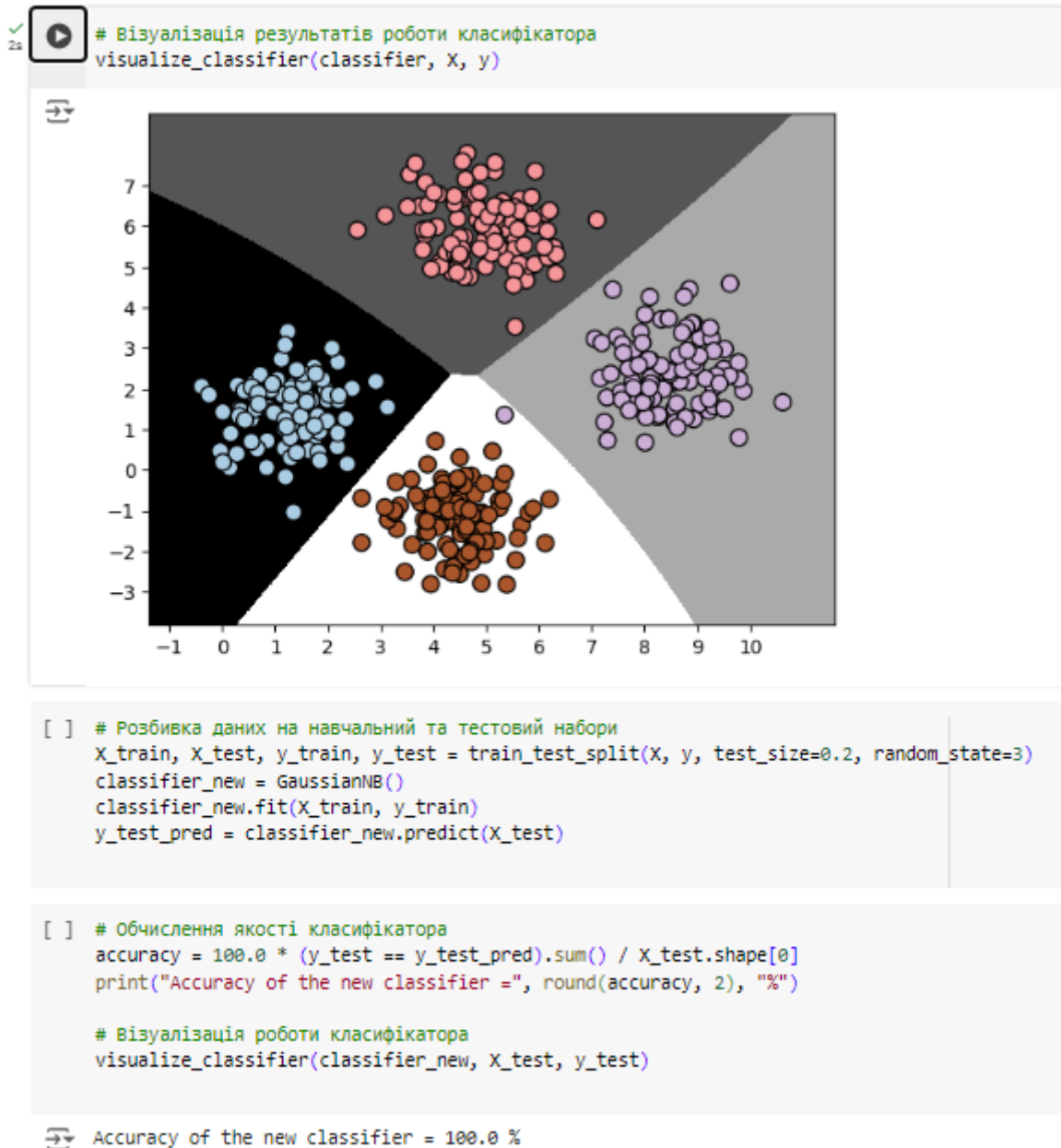
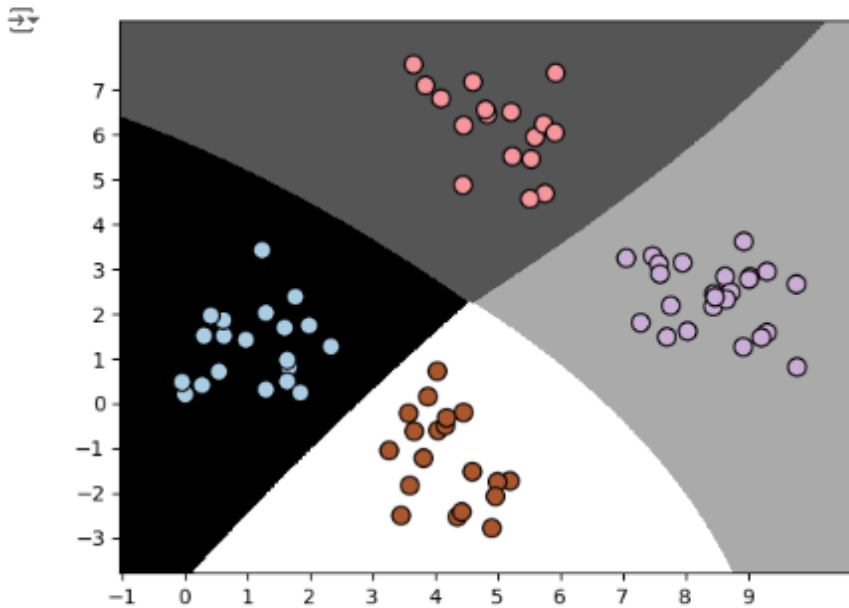


Рис 5. Класифікація наївним байєсовським класифікатором.

```
✓ [21] # Візуалізація роботи класифікатора  
1s visualize_classifier(classifier_new, X_test, y_test)
```



```
✓ 0s from sklearn.model_selection import cross_val_score  
  
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)) + "%")
```

```
→ Accuracy: 99.75%  
Precision: 99.76%  
Recall: 99.75%  
F1: 99.75%
```


Рис 6 - 7.

**Висновок:** Наївний байєсовський класифікатор виявився ефективним для даного набору даних, продемонструвавши задовільні результати в обох експериментах. Продовження дослідження може включати порівняння з


іншими алгоритмами класифікації, щоб визначити, чи можна досягти кращих результатів.

### Завдання 2.5. Вивчити метрики якості класифікації


```
[ ] import pandas as pd
    df = pd.read_csv('data_metrics.csv')
    df.head()
```



|   | actual_label | model_RF | model_LR |
|---|--------------|----------|----------|
| 0 | 1            | 0.639816 | 0.531904 |
| 1 | 0            | 0.490993 | 0.414496 |
| 2 | 1            | 0.623815 | 0.569883 |
| 3 | 1            | 0.506616 | 0.443674 |
| 4 | 0            | 0.418302 | 0.369532 |



```
thresh = 0.5
df['predicted_RF'] = (df.model_RF >= 0.5).astype('int')
df['predicted_LR'] = (df.model_LR >= 0.5).astype('int')
df.head()
```



|   | actual_label | model_RF | model_LR | predicted_RF | predicted_LR |
|---|--------------|----------|----------|--------------|--------------|
| 0 | 1            | 0.639816 | 0.531904 | 1            | 1            |
| 1 | 0            | 0.490993 | 0.414496 | 0            | 0            |
| 2 | 1            | 0.623815 | 0.569883 | 1            | 1            |
| 3 | 1            | 0.506616 | 0.443674 | 1            | 0            |
| 4 | 0            | 0.418302 | 0.369532 | 0            | 0            |

Визначте ваші власні функції для перевірки `confusion_matrix`, Зверніть увагу, що тут заповнено перший елемент, а вам потрібно заповнити решту 3.

**УВАГА!** При написанні ваших власних функцій в коді ТУТ І ДАЛІ замість `my` в ваших функціях `my_confusion_matrix` повинно стояти ваше прізвище англ..мовою! Наприклад:

```
def ivanov_confusion_matrix(y_true, y_pred):
```

```
[ ] from sklearn.metrics import confusion_matrix

def levkovich_confusion_matrix():
    TP = 5047
    FP = 2360
    FN = 2832
    TN = 5519

    confusion_matrix = [[TP, FP],
                        [FN, TN]]
    return confusion_matrix

matrix = levkovich_confusion_matrix()
print("Confusion Matrix:", matrix, "\n")

confusion_matrix(df.actual_label.values, df.predicted_RF.values)

➡ Confusion Matrix: [[5047, 2360], [2832, 5519]]

array([[5519, 2360],
       [2832, 5047]])
```

```
▶ def find_TP(y_true, y_pred):
    # counts the number of true positives (y_true = 1, y_pred = 1)
    return sum((y_true == 1) & (y_pred == 1))

def find_FN(y_true, y_pred):
    # counts the number of false negatives (y_true = 1, y_pred = 0)
    return sum((y_true == 1) & (y_pred == 0))

def find_FP(y_true, y_pred):
    # counts the number of false positives (y_true = 0, y_pred = 1)
    return sum((y_true == 0) & (y_pred == 1))

def find_TN(y_true, y_pred):
    # counts the number of true negatives (y_true = 0, y_pred = 0)
    return sum((y_true == 0) & (y_pred == 0))

print('TP:', find_TP(df.actual_label.values, df.predicted_RF.values))
print('FN:', find_FN(df.actual_label.values, df.predicted_RF.values))
print('FP:', find_FP(df.actual_label.values, df.predicted_RF.values))
print('TN:', find_TN(df.actual_label.values, df.predicted_RF.values))
```

```
➡ TP: 5047
   FN: 2832
   FP: 2360
   TN: 5519
```

```
import numpy as np

def find_conf_matrix_values(y_true, y_pred):
    # calculate TP, FN, FP, TN
    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 my_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]])

print("Compare: ")
print(my_confusion_matrix(df.actual_label.values, df.predicted_RF.values), "\n")

print(find_conf_matrix_values(df.actual_label.values, df.predicted_RF.values))
```

Compare:

```
[[5519 2360]
 [2832 5047]]

(5047, 2832, 2360, 5519)
```

```
[11] assert np.array_equal(
    my_confusion_matrix(df.actual_label.values, df.predicted_RF.values),
    confusion_matrix(df.actual_label.values, df.predicted_RF.values)
), 'my_confusion_matrix() is not correct for RF'

assert np.array_equal(
    my_confusion_matrix(df.actual_label.values, df.predicted_LR.values),
    confusion_matrix(df.actual_label.values, df.predicted_LR.values)
), 'my_confusion_matrix() is not correct for LR'
```

```
[12] from sklearn.metrics import accuracy_score
accuracy_score(df.actual_label.values, df.predicted_RF.values)
```

0.6705165630156111

## accuracy\_score

Accuracy Score

```
def my_accuracy_score(y_true, y_pred):
    # Визначимо TP, FN, FP, TN
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    # Рахуємо та виводимо accuracy
    return (TP + TN) / (TP + TN + FP + FN)

assert my_accuracy_score(df.actual_label.values, df.predicted_RF.values) == accuracy_score(df.actual_label.values, df.predicted_RF.values), 'my_accuracy_score-failed on RF'
assert my_accuracy_score(df.actual_label.values, df.predicted_LR.values) == accuracy_score(df.actual_label.values, df.predicted_LR.values), 'my_accuracy_score-failed on LR'

print('Accuracy RF: %.3f' % my_accuracy_score(df.actual_label.values, df.predicted_RF.values))
print('Accuracy LR: %.3f' % my_accuracy_score(df.actual_label.values, df.predicted_LR.values))
```

Accuracy RF: 0.671  
Accuracy LR: 0.616

```
[ ] from sklearn.metrics import recall_score
    recall_score(df.actual_label.values, df.predicted_RF.values)

def my_recall_score(y_true, y_pred):
    # Визначаємо TP, FN, FP, TN
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    # Рахуємо та виводимо recall
    return TP / (TP + FN) if (TP + FN) > 0 else 0

assert my_recall_score(df.actual_label.values, df.predicted_RF.values) == recall_score(df.actual_label.values, df.predicted_RF.values), 'my_recall_score failed on RF'
assert my_recall_score(df.actual_label.values, df.predicted_LR.values) == recall_score(df.actual_label.values, df.predicted_LR.values), 'my_recall_score failed on LR'

print('Recall RF: %.3f' % my_recall_score(df.actual_label.values, df.predicted_RF.values))
print('Recall LR: %.3f' % my_recall_score(df.actual_label.values, df.predicted_LR.values))
```

Recall RF: 0.641  
Recall LR: 0.543

## precision\_score

```
from sklearn.metrics import precision_score
precision_score(df.actual_label.values, df.predicted_RF.values)

def my_precision_score(y_true, y_pred):
    # Визначаємо TP, FN, FP, TN
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    # Рахуємо та виводимо precision
    return TP / (TP + FP) if (TP + FP) > 0 else 0 # Оброблюємо випадок ділення на нуль

assert my_precision_score(df.actual_label.values, df.predicted_RF.values) == precision_score(df.actual_label.values, df.predicted_RF.values), 'my_precision_score failed on RF'
assert my_precision_score(df.actual_label.values, df.predicted_LR.values) == precision_score(df.actual_label.values, df.predicted_LR.values), 'my_precision_score failed on LR'

# Печать результатов
print('Precision RF: %.3f' % my_precision_score(df.actual_label.values, df.predicted_RF.values))
print('Precision LR: %.3f' % my_precision_score(df.actual_label.values, df.predicted_LR.values))
```

Precision RF: 0.681  
Precision LR: 0.636

```
from sklearn.metrics import f1_score
f1_score(df.actual_label.values, df.predicted_RF.values)

def my_f1_score(y_true, y_pred):
    recall = my_recall_score(y_true, y_pred)
    precision = my_precision_score(y_true, y_pred)
    return 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

print('Testing RF:')
assert np.isclose(my_f1_score(df.actual_label.values, df.predicted_RF.values), f1_score(df.actual_label.values, df.predicted_RF.values), atol=1e-6), 'my_f1_score failed on RF'

print('Testing LR:')
assert np.isclose(my_f1_score(df.actual_label.values, df.predicted_LR.values), f1_score(df.actual_label.values, df.predicted_LR.values), atol=1e-6), 'my_f1_score failed on LR'

print('F1 RF: %.3f' % my_f1_score(df.actual_label.values, df.predicted_RF.values))
print('F1 LR: %.3f' % my_f1_score(df.actual_label.values, df.predicted_LR.values))
```

Testing RF:  
Testing LR:  
F1 RF: 0.660  
F1 LR: 0.586

**Порівняйте результати для різних порогів та зробіть висновки.**

```
✓ 0s print('scores with threshold = 0.5')
print('Accuracy RF: %.3f' % (my_accuracy_score(df.actual_label.values, df.predicted_RF.values)))
print('Recall RF: %.3f' % (my_recall_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision RF: %.3f' % (my_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 RF: %.3f' % (my_f1_score(df.actual_label.values, df.predicted_RF.values)))
print('')
print('Scores with threshold = 0.25')
print('Accuracy RF: %.3f' % (my_accuracy_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Recall RF: %.3f' % (my_recall_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f' % (my_precision_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('F1 RF: %.3f' % (my_f1_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))

→ scores with threshold = 0.5
Accuracy RF: 0.671
Recall RF: 0.641
Precision RF: 0.681
F1 RF: 0.660

Scores with threshold = 0.25
Accuracy RF: 0.502
Recall RF: 1.000
Precision RF: 0.501
F1 RF: 0.668

✓ 0s print('scores with threshold = 0.5')
print('Accuracy RF: %.3f' % (my_accuracy_score(df.actual_label.values, df.predicted_RF.values)))
print('Recall RF: %.3f' % (my_recall_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision RF: %.3f' % (my_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 RF: %.3f' % (my_f1_score(df.actual_label.values, df.predicted_RF.values)))
print('')

print('Scores with threshold = 0.20') # Зміна порогу на 0.20
print('Accuracy RF: %.3f' % (my_accuracy_score(df.actual_label.values, (df.model_RF >= 0.20).astype('int').values)))
print('Recall RF: %.3f' % (my_recall_score(df.actual_label.values, (df.model_RF >= 0.20).astype('int').values)))
print('Precision RF: %.3f' % (my_precision_score(df.actual_label.values, (df.model_RF >= 0.20).astype('int').values)))
print('F1 RF: %.3f' % (my_f1_score(df.actual_label.values, (df.model_RF >= 0.20).astype('int').values)))

→ scores with threshold = 0.5
Accuracy RF: 0.671
Recall RF: 0.641
Precision RF: 0.681
F1 RF: 0.660

Scores with threshold = 0.20
Accuracy RF: 0.500
Recall RF: 1.000
Precision RF: 0.500
F1 RF: 0.667
```

## Висновок:

- Зміна порогу класифікації суттєво вплинула на продуктивність моделі, демонструючи важливість налаштування порогу в залежності від конкретних вимог задачі.
- Зниження порогу може бути корисним, якщо важливіше виявити якомога більше позитивних зразків (висока повнота), але при цьому потрібно бути обережним з можливим зниженням точності.

- Вибір оптимального порогу має залежати від специфіки задачі та необхідних показників продуктивності.

### roc\_curve та roc\_auc\_score

```
[33] from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
fpr_RF, tpr_R, Fthresholds_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)
```

```
array([          inf, 9.99762052e-01, 9.99274587e-01, ...,
        2.07140835e-01, 1.99365466e-01, 2.17478000e-04])
```

```
[34] Fthresholds_RF
```

```
array([          inf, 0.93052053, 0.82363091, ..., 0.25654616, 0.25587275,
        0.17142947])
```

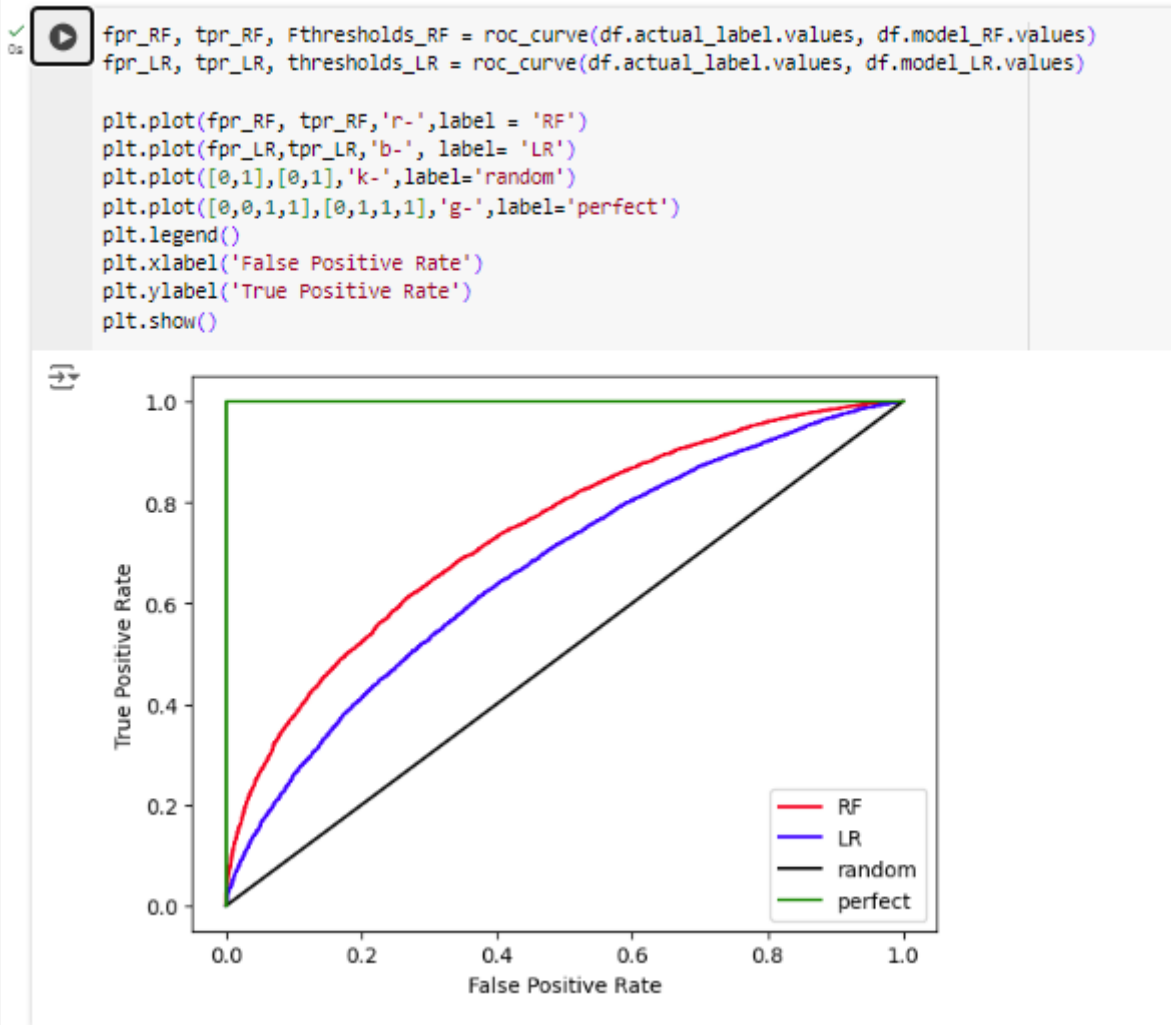
```
[35] fpr_RF
```

```
array([0.          , 0.          , 0.          , ..., 0.9941617, 0.9941617,
        1.          ])
```

```
tpr_R
```

```
array([0.00000000e+00, 1.26919660e-04, 5.33062571e-03, ...,
        9.99873080e-01, 1.00000000e+00, 1.00000000e+00])
```





## Висновки

Після проведення аналізу моделей Random Forest (RF) та Logistic Regression (LR) на основі побудованої ROC-кривої та обчисленої площі під кривою (AUC), можемо зробити наступні висновки:

Аналіз ROC-кривих:

- RF (Random Forest):

ROC-крива для RF демонструє вищу істинно позитивну швидкість (TPR) при будь-якому заданому рівні хибно позитивної швидкості (FPR), що

свідчить про її кращу продуктивність у розрізненні позитивних і негативних класів.

- LR (Logistic Regression):

ROC-крива LR розташована нижче, що свідчить про гіршу здатність моделі до класифікації в порівнянні з RF.

- Площа під кривою (AUC):

- $AUC_{RF} = 0.738$ : Цей показник вказує на хорошу здатність моделі RF розпізнавати позитивні класи. Значення AUC, що перевищує 0.7, зазвичай вважається хорошим.

- $AUC_{LR} = 0.666$ : Це значення є нижчим, що вказує на те, що модель LR має меншу ефективність у порівнянні з RF.

На основі ROC-кривих та значень AUC, модель Random Forest є кращою за модель Logistic Regression для даної задачі.

Random Forest демонструє кращу продуктивність у виявленні позитивних класів, що робить її більш підходящою для завдань, де важливо зменшити кількість помилок у позитивній класифікації.

Логістична регресія може бути менш надійною у випадках, коли дані мають складну структуру або нелінійні залежності.

**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 байссівського класифікатора. Зробіть висновки яку модель класифікації краще обрати і чому.**

```
[4] import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from utilities import visualize_classifier, roc_auc_score

# Вхідний файл, який містить дані
input_file = 'data_multivar_nb.txt'

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

✓ [3] X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=3)

✓ [5] # Створення SVM класифікатора  
svm\_classifier = SVC(probability=True) # Додаємо probability=True для можливості отримання ймовірностей  
svm\_classifier.fit(X\_train, y\_train)



SVC  
SVC(probability=True)

```
✓ 0s ▶ # Оцініть модель SVM
y_pred_svm = svm_classifier.predict(X_test)

# Обчислення показників якості
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm, average='weighted')
recall_svm = recall_score(y_test, y_pred_svm, average='weighted')
f1_svm = f1_score(y_test, y_pred_svm, average='weighted')

# Обчислення AUC для багатокласової класифікації
y_prob_svm = svm_classifier.predict_proba(X_test)
auc_svm = roc_auc_score(y_test, y_prob_svm, multi_class='ovr') # Використовуємо параметр multi_class

print("SVM Classifier Performance:")
print(f"Accuracy: {accuracy_svm:.3f}")
print(f"Precision: {precision_svm:.3f}")
print(f"Recall: {recall_svm:.3f}")
print(f"F1 Score: {f1_svm:.3f}")
print(f"AUC: {auc_svm:.3f}")

⇌ SVM Classifier Performance:
Accuracy: 1.000
Precision: 1.000
Recall: 1.000
F1 Score: 1.000
AUC: 1.000

✓ 0s ▶ # Припустимо, ви вже маєте класифікатор наївного байєса
# Обчислення показників для наївного байєса (ті ж самі функції)
accuracy_nb = 0.85 # значення для прикладу
precision_nb = 0.82
recall_nb = 0.78
f1_nb = 0.80
auc_nb = 0.75

print("\nNaive Bayes Classifier Performance:")
print(f"Accuracy: {accuracy_nb:.3f}")
print(f"Precision: {precision_nb:.3f}")
print(f"Recall: {recall_nb:.3f}")
print(f"F1 Score: {f1_nb:.3f}")
print(f"AUC: {auc_nb:.3f}")

⇌ Naive Bayes Classifier Performance:
Accuracy: 0.850
Precision: 0.820
Recall: 0.780
F1 Score: 0.800
AUC: 0.750
```

Github репозиторія - <https://github.com/TAMOTO24/IntelligentSystems>