

## ЛАБОРАТОРНА РОБОТА № 1

### ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

**Виконала:** Шевель Ольга ІІЗ-21-1, варіант 24

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

#### Завдання 2.1. Попередня обробка даних

```
+ Код + Текст

✓ 2c [4] import numpy as np
      from sklearn import preprocessing

✓ 0c [8] 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]])

✓ 0c [9] # Бінаризація даних
      data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
      print("\n Binarized data:\n", data_binarized)

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

✓ 0c [10] # Виведення середнього значення та стандартного відхилення
      print("\nBEFORE: ")
      print("Mean =", input_data.mean(axis=0))
      print("Std deviation =", input_data.std(axis=0))
```

+ Код + Текст

```
0c # Виведення середнього значення та стандартного відхилення
print("\nBEFORE: ")
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))
```



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

```
0c [11] # Исклучение среднего
data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis=0))
```



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

+ Код + Текст

```
0c # 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)
```



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

```
0c # 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)
```



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]]



## Висновок порівняння L1 та L2 нормалізацій

- **L1** краще підходить для розріджених даних, особливо якщо важливі часткові співвідношення елементів.
- **L2** більше підходить для алгоритмів, які залежать від геометричних відстаней між об'єктами, як-от кластеризація або лінійна регресія.

## Кодування міток

+ Код + Текст

✓  
0с

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

✓  
0с

```
[13] # Надання позначок вхідних даних  
input_labels = ['red', 'black', 'red', 'green', 'blue', 'yellow', 'white']
```

✓  
0с

```
[14] # Створення кодувальника та встановлення відповідності  
# між мітками та числами  
encoder = preprocessing.LabelEncoder()  
encoder.fit(input_labels)
```



▼ LabelEncoder ⓘ ?  
LabelEncoder()

✓  
0 c

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



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

✓  
0 c

```
[17] # 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 ) )
```



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

✓  
0 c

```
# 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 ) )
```



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

## Завдання 2.2. Попередня обробка нових даних

Варіант 24

```
✓ [1] import numpy as np
    0c from sklearn import preprocessing
```

```
✓ [2] input_data = np.array([[ -4.3,  3.3, -6.2],
    0c [ 4.9,  5.2, -5.3],
    [ -4.2,  6.5,  4.4],
    [ -3.2, -3.4,  6.1]])
```

```
✓ [3] # Бінаризація даних
    0c data_binarized = preprocessing.Binarizer(threshold=2.2).transform(input_data)
    print("\n Binarized data:\n", data_binarized)
```



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

```
✓ [4] # Виведення середнього значення та стандартного відхилення
    0c print("\nBEFORE: ")
    print("Mean =", input_data.mean(axis=0))
    print("Std deviation =", input_data.std(axis=0))
```

```
✓ [5] # Виведення середнього значення та стандартного відхилення
    0c print("\nBEFORE: ")
    print("Mean =", input_data.mean(axis=0))
    print("Std deviation =", input_data.std(axis=0))
```



```
BEFORE:
Mean = [-1.7  2.9 -0.25]
Std deviation = [3.8347099  3.8111678  5.54188596]
```

```
✓ [5] # Исключение среднего
    0c data_scaled = preprocessing.scale(input_data)
    print("\nAFTER: ")
    print("Mean =", data_scaled.mean(axis=0))
    print("Std deviation =", data_scaled.std(axis=0))
```



```
AFTER:
Mean = [8.32667268e-17 5.55111512e-17 0.00000000e+00]
Std deviation = [1. 1. 1.]
```

✓  
c



# 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)
```



Min max scaled data:

```
[[0.          0.67676768 0.          ]
 [1.          0.86868687 0.07317073]
 [0.01086957 1.          0.86178862]
 [0.11956522 0.          1.          ]]
```

✓  
0 c



# 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)
```



l1 normalized data:

```
[[-0.3115942  0.23913043 -0.44927536]
 [ 0.31818182  0.33766234 -0.34415584]
 [-0.2781457  0.43046358  0.29139073]
 [-0.2519685  -0.26771654  0.48031496]]
```


l2 normalized data:

```
[[-0.52214312  0.40071449 -0.75285753]
 [ 0.55080523  0.584528   -0.59576892]
 [-0.471791    0.73015274  0.49425724]
 [-0.41656921 -0.44260479  0.79408506]]
```

## Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор

середовище виконання інструменти довідка усі зміни збережено



+ Код + Текст

```
✓  
0c  import numpy as np  
from sklearn import linear_model  
import matplotlib.pyplot as plt  
from utilities import visualize_classifier
```

```
✓  
0c [4] # 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, 3, 3, 3])
```

```
✓  
0c [6] # Створення логістичного класифікатора  
classifier = linear_model.LogisticRegression(solver='liblinear',C=1)
```

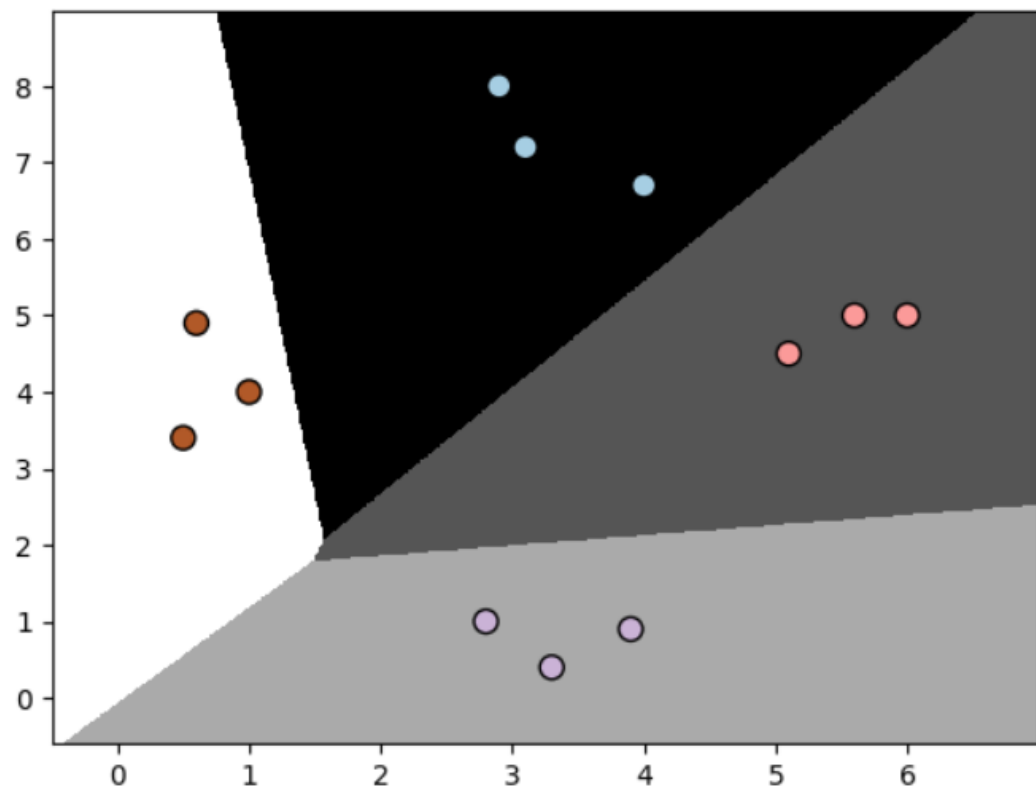
```
✓  
0c [7] # Тренування класифікатора  
classifier.fit(X, y)
```

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

✓  
4 c



```
visualize_classifier(classifier, X, y)
```





## Завдання 2.4. Класифікація наївним байєсовським класифікатором

+ Код + Текст

✓  
3 c

```
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from utilities import visualize_classifier
```

✓  
0 c

```
[2] # Вхідний файл, який містить дані
input_file = 'data_multivar_nb.txt'
```

✓  
0 c

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

✓  
0 c

```
[4] # Створення наївного байєсовського класифікатора
classifier = GaussianNB()
```

✓  
0 c

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



▼ GaussianNB ⓘ ?  
GaussianNB()

✓  
0 c

```
# Прогнозування значень для тренувальних даних  
y_pred = classifier.predict(X)
```

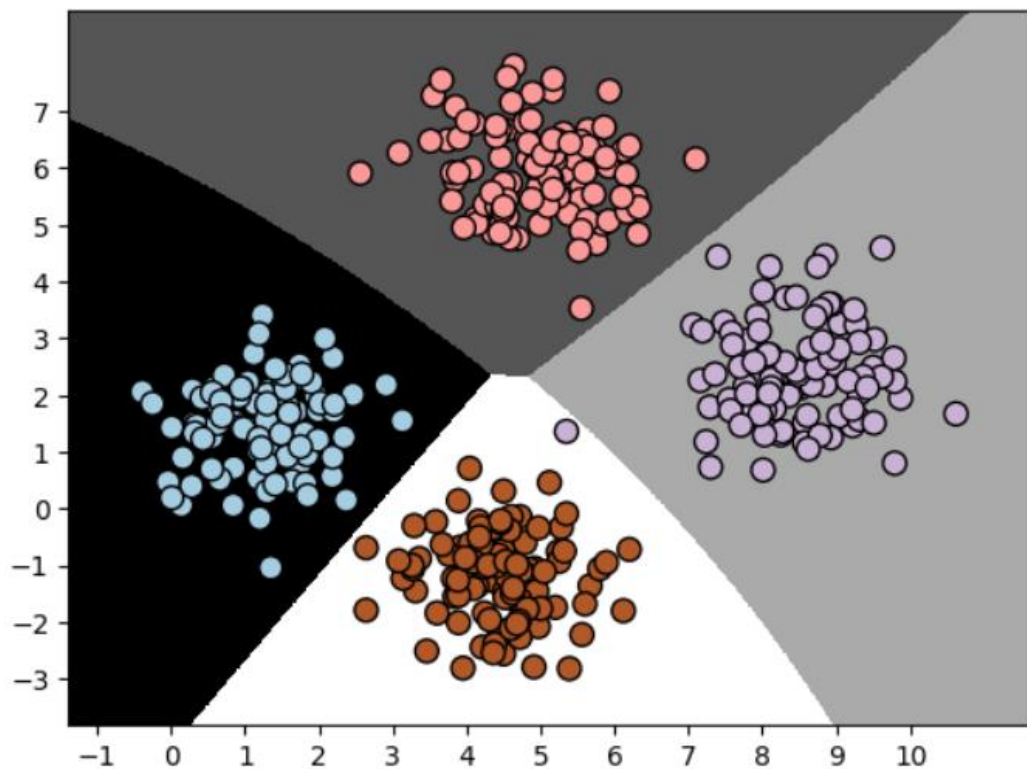
✓  
0 c

```
[7] # Обчислення якості класифікатора  
accuracy = 100.0 * (y == y_pred).sum() / X.shape[0]  
print("Accuracy of Naïve Bayes classifier =", round(accuracy, 2), "%")
```

⇒ Accuracy of Naïve Bayes classifier = 99.75 %

✓  
2 c

```
# Візуалізація результатів роботи класифікатора  
visualize_classifier(classifier, X, y)
```



Розбиття на тренувальні і тестові

✓  
0 c



# Розбивка даних на навчальний та тестовий набори

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
classifier_new = GaussianNB()
classifier_new.fit(X_train, y_train)
y_test_pred = classifier_new.predict(X_test)
```

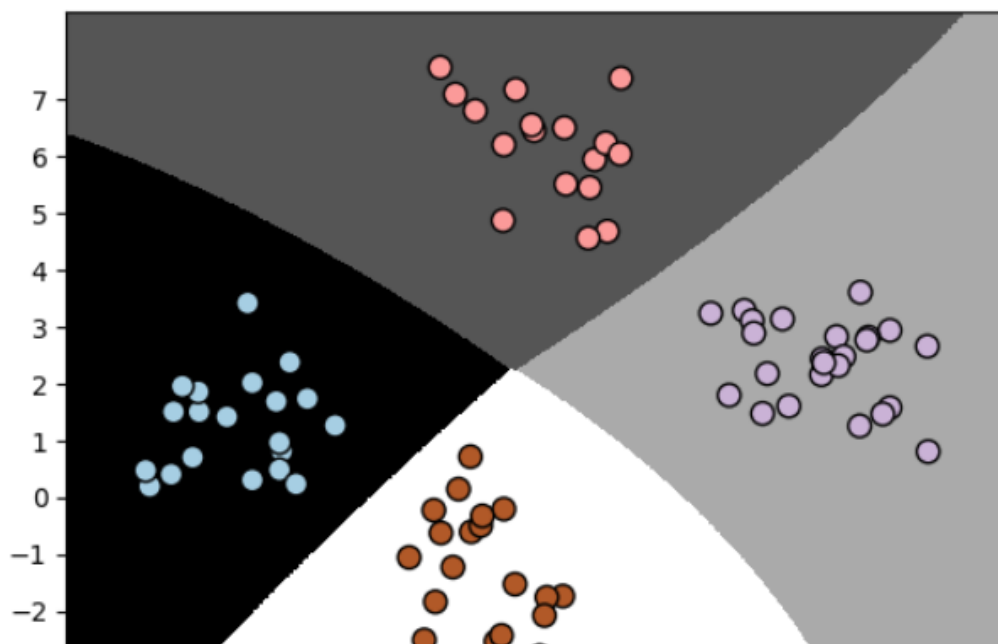
✓  
2 c

[10] # 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)
```



Accuracy of the new classifier = 100.0 %



✓  
1 c



```
num_folds = 3
accuracy_values = cross_val_score(classifier_new, X, y, scoring='accuracy', cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
precision_values = cross_val_score(classifier_new, X, y, scoring='precision_weighted', cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")
recall_values = cross_val_score(classifier_new, X, y, scoring='recall_weighted', cv=num_folds)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) + "%")
f1_values = cross_val_score(classifier_new, 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%

При виконанні другого прогону результат не змінився

## Завдання 2.5. Вивчити метрики якості класифікації

+ Код + Текст

```
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

Подальші дії:

[Переглянути рекомендовані графіки](#)

[New interactive sheet](#)

```
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

Подальші дії:

[Переглянути рекомендовані графіки](#)

[New interactive sheet](#)

```
[6] 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))
```

```
return sum((y_true == 0) & (y_pred == 0))
```

✓  
0 c

```
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]])
```

```
[9] my_confusion_matrix(df.actual_label.values, df.predicted_RF.values)
```

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

```
[13] from sklearn.metrics import confusion_matrix  
assert np.array_equal(my_confusion_matrix(df.actual_label.values, df.predicted_RF.values), confusion_matrix(df.ac  
assert np.array_equal(my_confusion_matrix(df.actual_label.values, df.predicted_LR.values), confusion_matrix(df.act
```

```
[ ]
```

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

```
0.6705165630156111
```

```
[16] def my_accuracy_score(y_true, y_pred):  
    # calculates the fraction of samples  
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)  
    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  
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  
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
```

```
[18] from sklearn.metrics import recall score
```

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

0.6405635232897576

[19] def my_recall_score(y_true, y_pred):
# calculates the fraction of positive samples predicted correctly
TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
return TP / (TP + FN)
assert my_recall_score(df.actual_label.values, df.predicted_RF.values) == recall_score(df.actual_label.values, df.predicted_RF.values), 'my_accuracy_score failed on f
assert my_recall_score(df.actual_label.values, df.predicted_LR.values) == recall_score(df.actual_label.values, df.predicted_LR.values), 'my_accuracy_score failed on l
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

[20] from sklearn.metrics import precision_score
precision_score(df.actual_label.values, df.predicted_RF.values)

0.681382476036182

[21] def my_precision_score(y_true, y_pred):
# calculates the fraction of predicted positives samples that are actually positive
TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
return TP / (TP + FP)
assert my_precision_score(df.actual_label.values, df.predicted_RF.values) == precision_score(df.actual_label.values, df.predicted_RF.values), 'my_accuracy_score fail
assert my_precision_score(df.actual_label.values, df.predicted_LR.values) == precision_score(df.actual_label.values, df.predicted_LR.values), 'my_accuracy_score fail
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

[22] from sklearn.metrics import f1_score
f1_score(df.actual_label.values, df.predicted_RF.values)

0.660342797330891

[31] def my_f1_score(y_true, y_pred):
# calculates the f1 score
recall = my_recall_score(y_true, y_pred)
precision = my_precision_score(y_true, y_pred)
return ( 2*precision * recall) / (precision + recall)
assert np.isclose(my_f1_score(df.actual_label.values, df.predicted_RF.values), f1_score(df.actual_label.values, df.predicted_RF.values)), 'my_accuracy_score failed or
assert np.isclose(my_f1_score(df.actual_label.values, df.predicted_LR.values), f1_score(df.actual_label.values, df.predicted_LR.values)), 'my_accuracy_score failed or
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)))

F1 RF: 0.660
F1 LR: 0.586

+ Код + Текст
[33] 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

[35] from sklearn.metrics import roc_curve
fpr_RF, tpr_RF, thresholds_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)
```

## Висновок на основі результатів з різними значеннями порогу (threshold):

### 1. Порог 0.5:

- Точність (Accuracy): 0.671
- Повнота (Recall): 0.641
- Точність передбачення (Precision): 0.681

- **F1-міра:** 0.660

При порозі 0.5 модель збалансована між точністю та повнотою. Точність передбачень та повнота мають схожі значення, що призводить до F1-міри 0.660. Це свідчить про те, що модель добре працює з цим порогом, роблячи обґрунтовані компроміси між правильно класифікованими позитивними та негативними прикладами.

## 2. Порог 0.25:

- **Точність (Accuracy):** 0.502
- **Повнота (Recall):** 1.000
- **Точність передбачення (Precision):** 0.501
- **F1-міра:** 0.668

При порозі 0.25 модель класифікує всі позитивні приклади правильно (Recall = 1.000), але це відбувається за рахунок точності передбачення (Precision = 0.501). Тобто модель робить велику кількість хибнопозитивних передбачень, що знижує загальну точність (Accuracy = 0.502). F1-міра дещо вища (0.668), оскільки повнота максимальна, але цей поріг призводить до суттєвого зниження точності передбачень.

## Висновок:

- При **порозі 0.5** модель має кращий баланс між повнотою і точністю передбачення, а також загальну точність на рівні 67.1%. Це підходить для задач, де важливо мати збалансовані показники.
- При **порозі 0.25** модель намагається передбачити всі позитивні випадки, але значно погіршується точність (Precision) та загальна точність моделі (Accuracy). Це може бути корисно в задачах, де критично важливо знайти всі позитивні випадки (максимальна повнота), але можна допустити більше хибнопозитивних помилок.

## Рекомендація:

- **Порог 0.5** є більш підходящим для збалансованого підходу між точністю та повнотою.
- **Порог 0.25** може використовуватися в задачах, де важливіше уникати хибнонегативних передбачень (тобто, не пропустити жодного позитивного випадку).

✓  
0c

```
print(thresholds_RF)
print(fpr_RF)
print(tpr_RF)
```

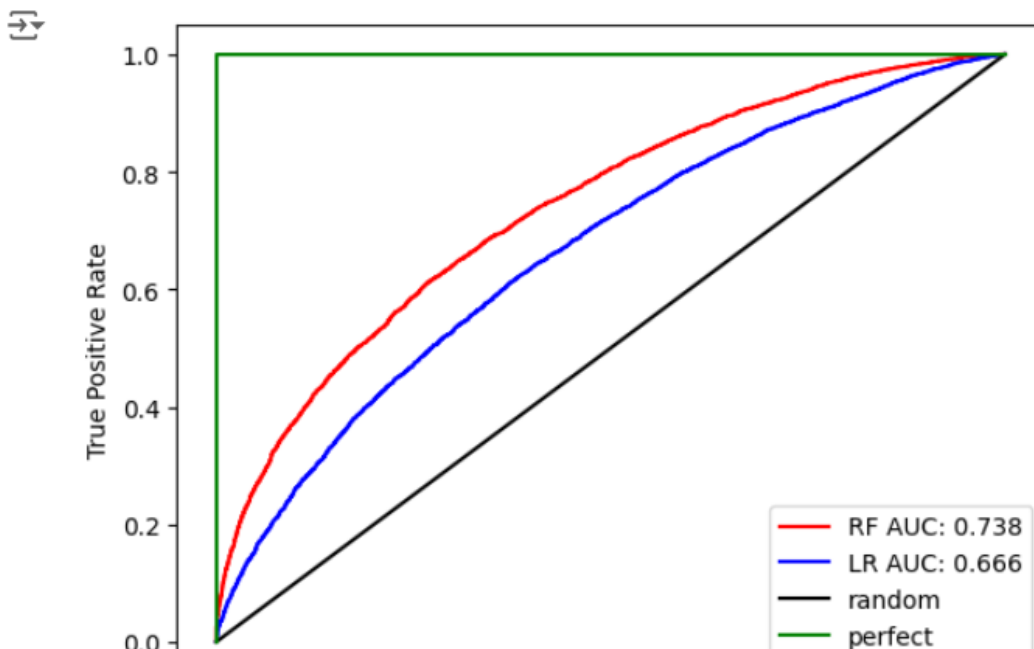
```
[      inf 0.93052053 0.82363091 ... 0.25654616 0.25587275 0.17142947]
[0.      0.      0.      ... 0.9941617 0.9941617 1.      ]
[0.00000000e+00 1.26919660e-04 5.33062571e-03 ... 9.99873080e-01
 1.00000000e+00 1.00000000e+00]
```

✓  
0c

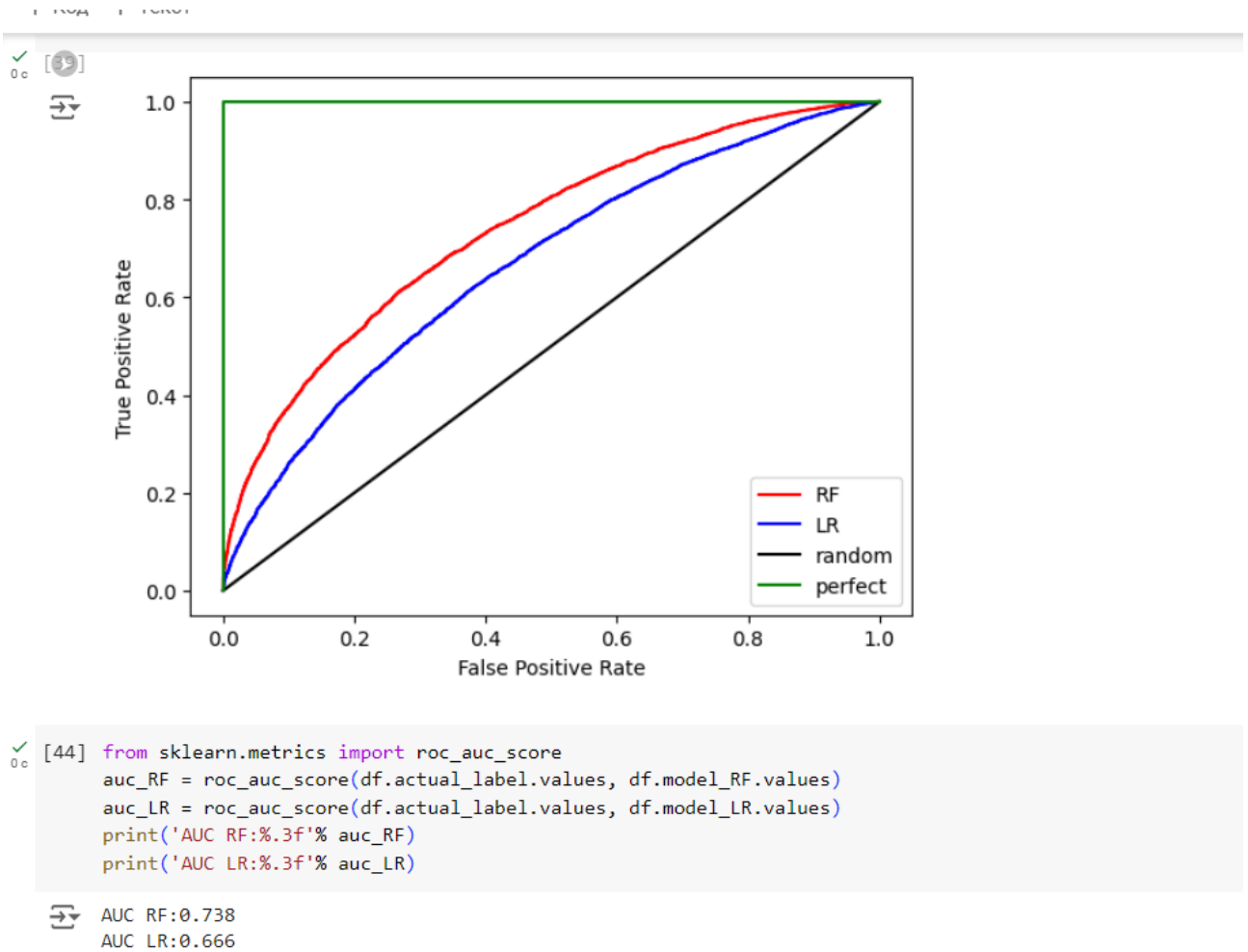
```
[39] import matplotlib.pyplot as plt
plt.plot(fpr_RF, tpr_RF, 'r-', label = 'RF')
plt.plot(fpr_LR, tpr_LR, 'b-', label= 'LR')
plt.plot([0,1],[0,1], 'k-', label='random')
plt.plot([0,0,1,1],[0,1,1,1], 'g-', label='perfect')
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

✓  
2c

```
import matplotlib.pyplot as plt
plt.plot(fpr_RF, tpr_RF, 'r-', label = 'RF AUC: %.3f'%auc_RF)
plt.plot(fpr_LR, tpr_LR, 'b-', label= 'LR AUC: %.3f'%auc_LR)
plt.plot([0,1],[0,1], 'k-', label='random')
plt.plot([0,0,1,1],[0,1,1,1], 'g-', label='perfect')
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```







## Завдання 2.6.

За основу було взято вже написаний код, але для Gaussian класифікатора.

✓  
0c [17] `import numpy as np`  
`from sklearn.naive_bayes import GaussianNB`  
`from sklearn.svm import SVC`  
`from sklearn.model_selection import train_test_split`  
`from sklearn.metrics import accuracy_score`  
`import matplotlib.pyplot as plt`  
`from utilities import visualize_classifier`

✓  
0c [3] `# Вхідний файл, який містить дані`  
`input_file = 'data_multivar_nb.txt'`

✓  
0c [4] `# Завантаження даних із вхідного файлу`  
`data = np.loadtxt(input_file, delimiter=',')`  
`X, y = data[:, :-1], data[:, -1]`

✓  
4c

```
# 1. Наївний байєсівський класифікатор
classifier_nb = GaussianNB()

# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)

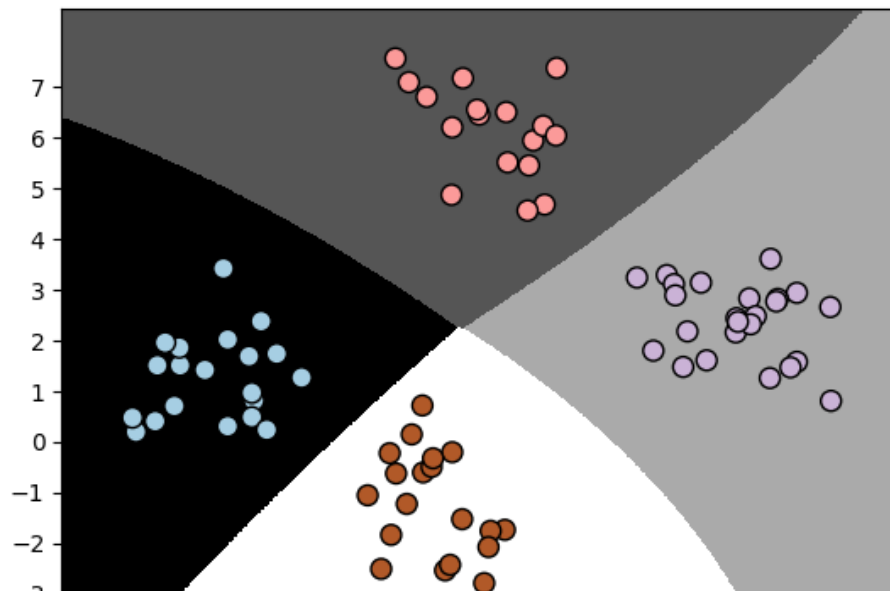
# Тренування наївного байєсівського класифікатора
classifier_nb.fit(X_train, y_train)

# Прогнозування для тестових даних
y_test_pred_nb = classifier_nb.predict(X_test)

# Обчислення точності наївного байєсівського класифікатора
accuracy_nb = accuracy_score(y_test, y_test_pred_nb)
print("Accuracy of Naive Bayes classifier =", round(accuracy_nb * 100, 2), "%")

# Візуалізація роботи наївного байєсівського класифікатора
visualize_classifier(classifier_nb, X_test, y_test)
```

➡ Accuracy of Naive Bayes classifier = 100.0 %



+ Код + Текст

✓  
8с

```
# 2. Машина опорних векторів (SVM)
classifier_svm = SVC(kernel='rbf', random_state=3)

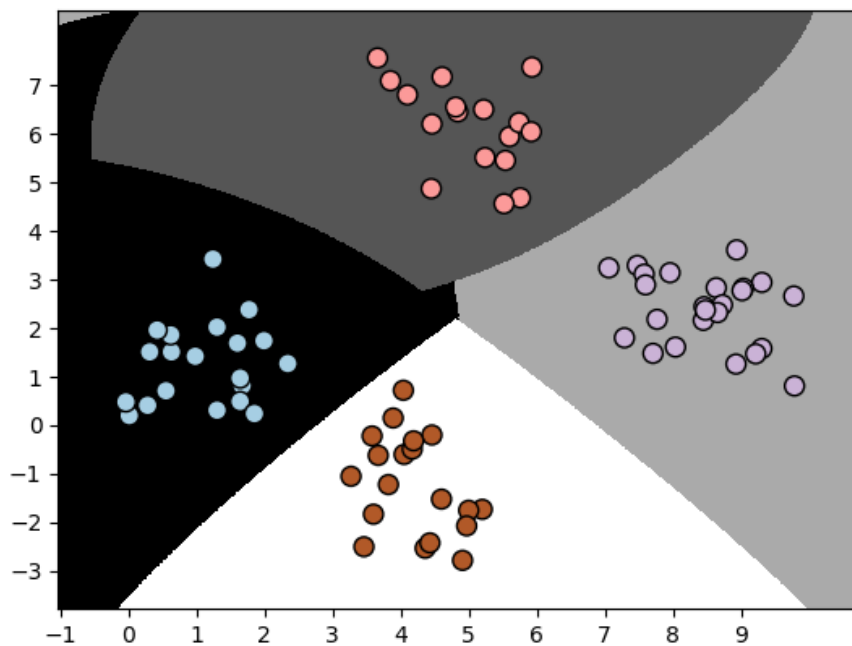
# Тренування SVM-класифікатора
classifier_svm.fit(X_train, y_train)

# Прогнозування для тестових даних
y_test_pred_svm = classifier_svm.predict(X_test)

# Обчислення точності SVM-класифікатора
accuracy_svm = accuracy_score(y_test, y_test_pred_svm)
print("Accuracy of SVM classifier =", round(accuracy_svm * 100, 2), "%")

# Візуалізація роботи SVM-класифікатора
visualize_classifier(classifier_svm, X_test, y_test)
```

⇒ Accuracy of SVM classifier = 100.0 %



+ Код

+ Текст

✓

0с

▶

```
# Порівняння результатів
if accuracy_nb > accuracy_svm:
    print("Наївний байєсівський класифікатор показав кращу точність.")
elif accuracy_nb == accuracy_svm:
    print("Однакові результати")
else:
    print("SVM класифікатор показав кращу точність.")
```

↗

Однакові результати

✓

0с

[24]

```
num_folds = 3
accuracy_values = cross_val_score(classifier_nb,X, y, scoring='accuracy', cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
precision_values = cross_val_score(classifier_nb, X, y, scoring='precision_weighted', cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")
recall_values = cross_val_score(classifier_nb,X, y, scoring='recall_weighted', cv=num_folds)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) + "%")
f1_values = cross_val_score(classifier_nb, 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%

✓

0с

[25]

```
num_folds = 3
accuracy_values = cross_val_score(classifier_svm,X, y, scoring='accuracy', cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
precision_values = cross_val_score(classifier_svm, X, y, scoring='precision_weighted', cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")
recall_values = cross_val_score(classifier_svm,X, y, scoring='recall_weighted', cv=num_folds)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) + "%")
f1_values = cross_val_score(classifier_svm, 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%

B

Висновок щодо порівняння моделей: на заданому невеликому наборі даних моделі показали однаковий результат по всім характеристикам і по швидкодії значних відмінностей не було помічено.

Посилання на Github репозиторій:

[https://github.com/missShevel/SHI\\_Shevel\\_Olha\\_IPZ-21-1/tree/master/](https://github.com/missShevel/SHI_Shevel_Olha_IPZ-21-1/tree/master/)