ЛАБОРАТОРНА РОБОТА № 1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

Хід роботи:

Завдання №1. Попередня обробка даних.

Лістинг програми:

```
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))

# Виключення середнього
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)
```

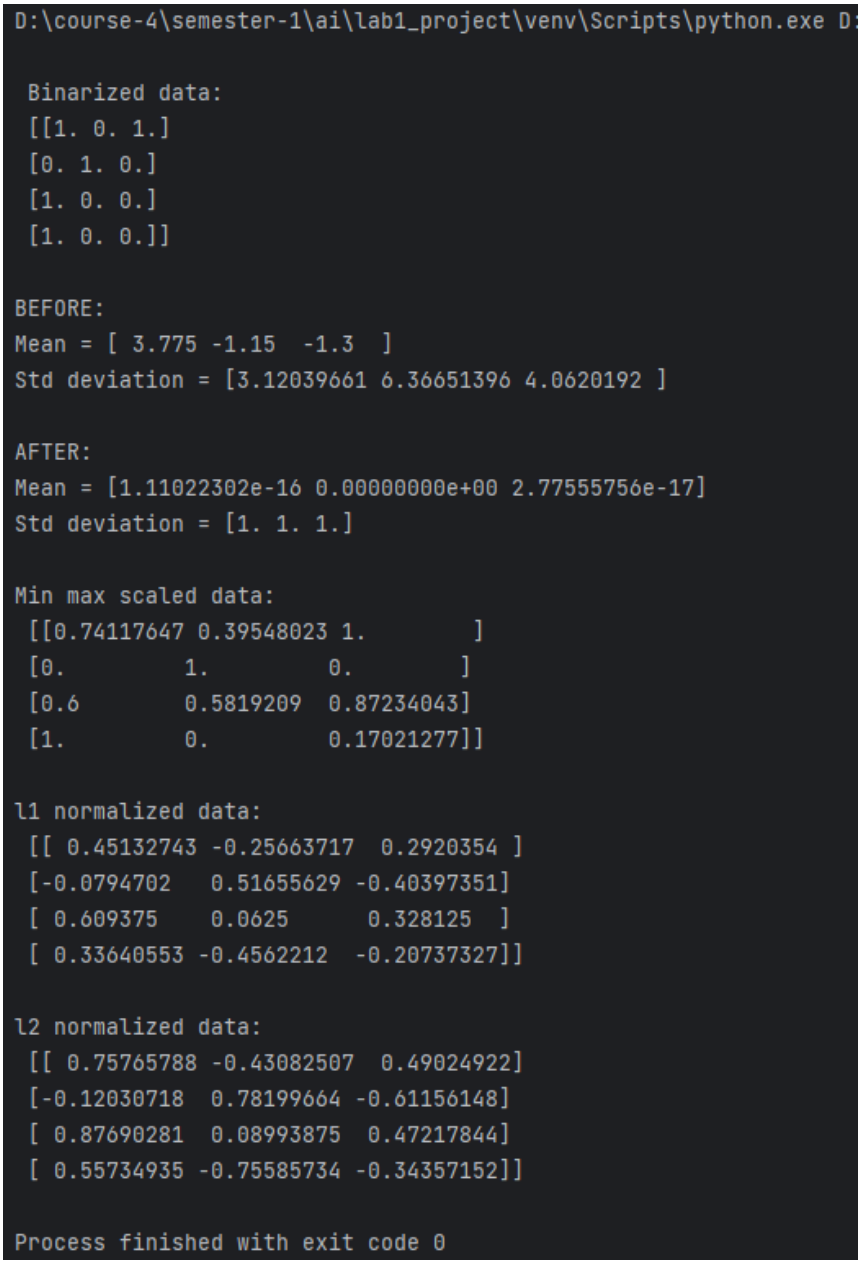


Рис.1.1. Результат виконання коду.

Чим відрізняється L1 від L2 нормалізації:

- L1-нормалізація використовує метод найменших абсолютних відхилень, при якому сума абсолютних значень в кожному ряду дорівнює 1.
- L2-нормалізація використовує метод найменших квадратів, при якому сума квадратів значень в кожному ряду дорівнює 1
- L1-нормалізація менш чутлива до викидів, того вважається надійнішою за L2-нормалізацію, яка у свою чергу краще підходить для задач, де викиди грають важливу роль.

Завдання №2.

Лістинг файлу LR_1_task_1.py:

```
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))
```

```
D:\course-4\semester-1\ai\lab1_project\venv\Scripts\python

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

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

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

Process finished with exit code 0
```

Рис.2.1. Результат виконання коду.

Завдання №3.

23.	2.5	-1.6	-6.1	-2.4	-1.2	4.3	3.2	3.1	6.1	-4.4	1.4	-1.2	2.5

Лістинг файлу LR_1_task_2.py:

```
import numpy as np
from sklearn import preprocessing

'''
# First task
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]])
'''
input_data = np.array([[2.5, -1.6, -6.1],
                       [-2.4, -1.2, 4.3],
                       [3.2, 3.1, 6.1],
                       [-4.4, 1.4, -1.2]])
# Бінаризація даних
data_binarized =
preprocessing.Binarizer(threshold=2.5).transform(input_data)  #
threshold=2.1 in first task
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))

# Виключення середнього
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)
```

```
D:\course-4\semester-1\ai\lab1_project\venv\Scripts\python.e

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

BEFORE:
Mean = [-0.275  0.425  0.775]
Std deviation = [3.21354555 1.92662269 4.79446295]

AFTER:
Mean = [ 0.00000000e+00 -5.55111512e-17 -4.16333634e-17]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.90789474 0.         0.        ]
 [0.26315789 0.08510638 0.85245902]
 [1.         1.         1.        ]
 [0.         0.63829787 0.40163934]]

l1 normalized data:
 [[ 0.24509804 -0.15686275 -0.59803922]
 [-0.30379747 -0.15189873  0.5443038 ]
 [ 0.25806452  0.25        0.49193548]
 [-0.62857143  0.2        -0.17142857]]

l2 normalized data:
 [[ 0.36852479 -0.23585586 -0.89920048]
 [-0.47351004 -0.23675502  0.84837215]
 [ 0.42362745  0.41038909  0.80753983]
 [-0.92228798  0.29345527 -0.25153308]]

Process finished with exit code 0
```

Рис.3.1. Результат виконання коду.

Завдання №4. Класифікація логістичною регресією або логістичний класифікатор.

Лістинг файлу LR_1_task_3.py:

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

# 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, 2, 3, 3, 3])

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

# Тренування класифікатора
classifier.fit(X, y)

visualize_classifier(classifier, X, y)
```

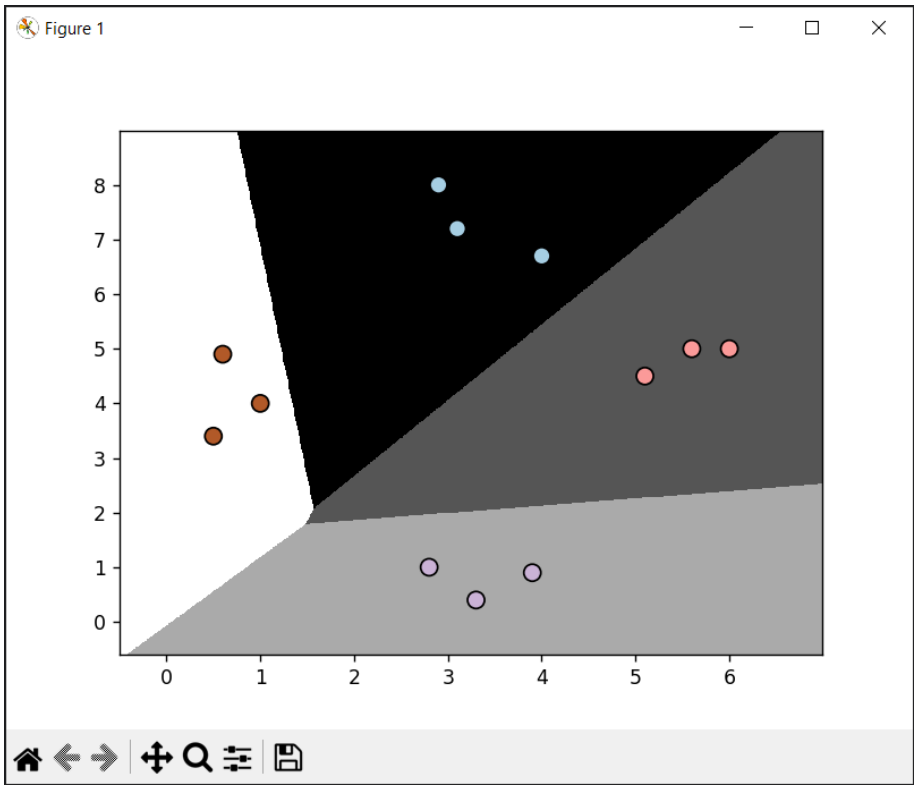


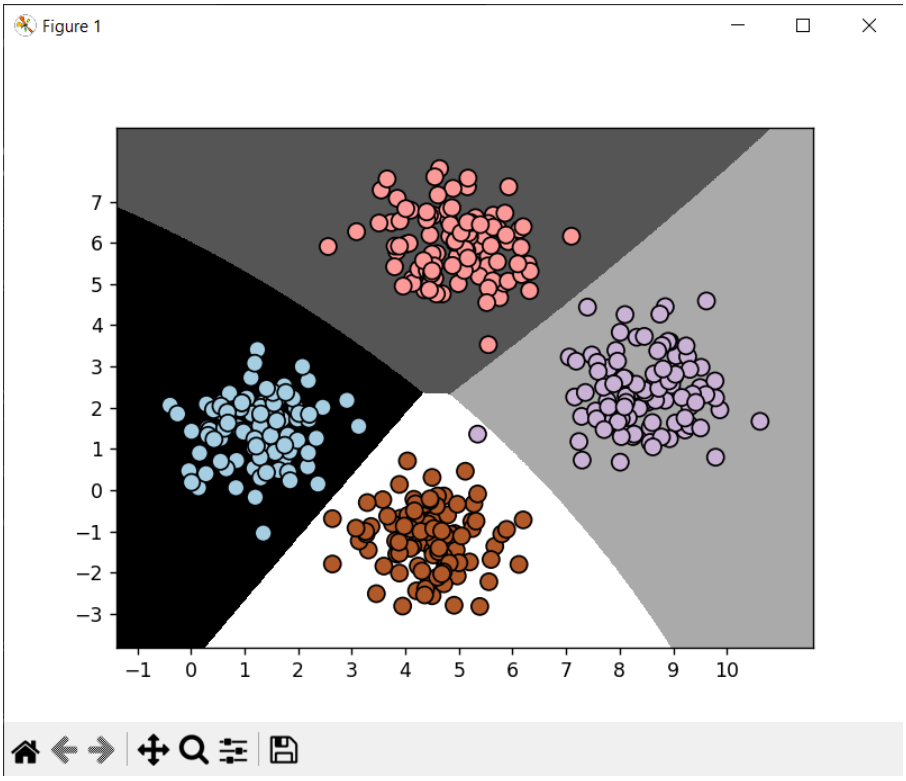
Рис.4.1. Результат виконання коду.

Завдання №5. Класифікація наївним байєсовським класифікатором.

Лістинг файлу LR_1_task_4.py:

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

# 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), "%")

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



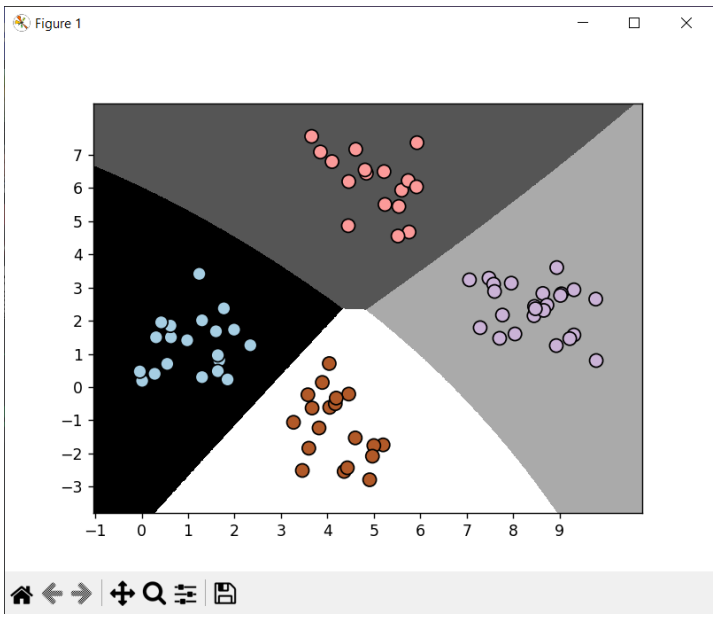

```
D:\course-4\semester-1\ai\lab1_project\venv\Sc
Accuracy of Naive Bayes classifier = 99.75 %

Process finished with exit code 0
```

Рис.5.1 – 5.2. Результат виконання коду.

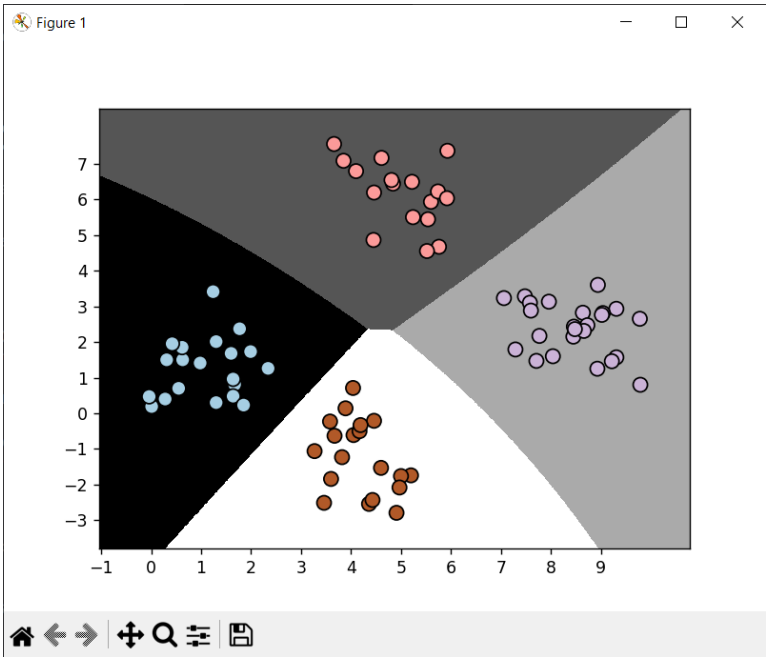
Лістинг файлу LR_1_task_4.py:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split, cross_val_score

from utilities import visualize_classifier

input_file = 'data_multivar_nb.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# 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)
classifier = GaussianNB()
classifier.fit(X, y)
y_test_pred = classifier.predict(X_test)

# Обчислення якості класифікатора
accuracy = 100.0 * (y_test == y_test_pred).sum() / X_test.shape[0]
print("Accuracy of the new Naive Bayes classifier =", round(accuracy, 2),
"%")

# Візуалізація роботи класифікатора
visualize_classifier(classifier, X_test, y_test)

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)) + "%")
```



```
D:\course-4\semester-1\ai\lab1_project\venv\Scripts\p
Accuracy of the new Naive Bayes classifier = 100.0 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%

Process finished with exit code 0
```




```
D:\course-4\semester-1\ai\lab1_project\venv\Scripts\py
Accuracy of the new Naive Bayes classifier = 100.0 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%

Process finished with exit code 0
```

Рис.5.3 – 5.6. Результат виконання коду.

Висновок – точність класифікатора вище у другому методі, оскільки в результаті перевірки було отримано оцінку якості 100%, в той час, як перший метод показав 99,75%. Другий запуск останнього методу дав такі самі результати, як і перший запуск, що підтверджує отриману оцінку.

Завдання №6. Вивчити метрики якості класифікації.

Лістинг файлу LR_1_task_5.py:

```
import pandas as pd
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, \
    recall_score, f1_score, roc_curve, roc_auc_score
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('data_metrics.csv')
df.head()

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

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


def find_TP(y_true, y_pred):
    return sum((y_true == 1) & (y_pred == 1))


def find_FN(y_true, y_pred):
    return sum((y_true == 1) & (y_pred == 0))


def find_FP(y_true, y_pred):
    return sum((y_true == 0) & (y_pred == 1))


def find_TN(y_true, y_pred):
    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))


def find_conf_matrix_values(y_true, y_pred):
    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 ryabova_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]])


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

assert np.array_equal(ryabova_confusion_matrix(df.actual_label.values,
df.predicted_RF.values),
                      confusion_matrix(df.actual_label.values,
df.predicted_RF.values)
                      ), 'ryabova_confusion_matrix() is not correct for RF'
```

```
assert np.array_equal(ryabova_confusion_matrix(df.actual_label.values,
                                               df.predicted_LR.values),
confusion_matrix(df.actual_label.values,

df.predicted_LR.values)), 'ryabova_confusion_matrix() is not correct for LR'

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


def ryabova_accuracy_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return (TP + TN) / (TP + FP + TN + FN)


assert ryabova_accuracy_score(df.actual_label.values, df.predicted_RF.values) ==
accuracy_score(df.actual_label.values,

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

df.predicted_LR.values), 'ryabova_accuracy_score failed on LR'
print('Accuracy RF: %.3f' % (ryabova_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Accuracy LR: %.3f' % (ryabova_accuracy_score(df.actual_label.values,
df.predicted_LR.values)))

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


def ryabova_recall_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FN)


assert ryabova_recall_score(df.actual_label.values, df.predicted_RF.values) ==
recall_score(df.actual_label.values,

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

df.predicted_LR.values), 'ryabova_recall_score failed on LR'
print('Recall RF: %.3f' % (ryabova_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall LR: %.3f' % (ryabova_recall_score(df.actual_label.values,
df.predicted_LR.values)))

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


def ryabova_precision_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FP)


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

```
    df.actual_label.values, df.predicted_LR.values), 'ryabova_precision_score
failed on LR'
print('Precision RF:%.3f' % (ryabova_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision LR: %.3f' % (ryabova_precision_score(df.actual_label.values,
df.predicted_LR.values)))

f1_score(df.actual_label.values, df.predicted_RF.values)


def ryabova_f1_score(y_true, y_pred):
    recall = ryabova_recall_score(y_true, y_pred)
    precision = ryabova_precision_score(y_true, y_pred)
    return 2 * precision * recall / (precision + recall)


assert ryabova_f1_score(df.actual_label.values, df.predicted_RF.values) ==
f1_score(df.actual_label.values,

df.predicted_RF.values), 'ryabova_f1_score failed on RF'
assert ryabova_f1_score(df.actual_label.values, df.predicted_LR.values) ==
f1_score(df.actual_label.values,

df.predicted_LR.values), 'ryabova_f1_score failed on LR'
print('F1 RF: %.3f' % (ryabova_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 LR: %.3f' % (ryabova_f1_score(df.actual_label.values,
df.predicted_LR.values)))

print('\nscores with threshold = 0.5')
print('Accuracy RF: %.3f' % (ryabova_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall RF: %.3f' % (ryabova_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF: %.3f' % (ryabova_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 RF: %.3f' % (ryabova_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('Accuracy LR: %.3f' % (ryabova_accuracy_score(df.actual_label.values,
df.predicted_LR.values)))
print('Recall LR: %.3f' % (ryabova_recall_score(df.actual_label.values,
df.predicted_LR.values)))
print('Precision LR: %.3f' % (ryabova_precision_score(df.actual_label.values,
df.predicted_LR.values)))
print('F1 LR: %.3f' % (ryabova_f1_score(df.actual_label.values,
df.predicted_LR.values)))

print('\nscores with threshold = 0.75')
print('Accuracy RF: %.3f' % (ryabova_accuracy_score(df.actual_label.values,
(df.model_RF >= 0.75).astype('int').values)))
print('Recall RF: %.3f' % (ryabova_recall_score(df.actual_label.values,
(df.model_RF >= 0.75).astype('int').values)))
print('Precision RF: %.3f' % (ryabova_precision_score(df.actual_label.values,
(df.model_RF >= 0.75).astype('int').values)))
print('F1 RF: %.3f' % (ryabova_f1_score(df.actual_label.values, (df.model_RF >=
0.75).astype('int').values)))
print('Accuracy LR: %.3f' % (ryabova_accuracy_score(df.actual_label.values,
(df.model_LR >= 0.75).astype('int').values)))
print('Recall LR: %.3f' % (ryabova_recall_score(df.actual_label.values,
(df.model_LR >= 0.75).astype('int').values)))
print('Precision LR: %.3f' % (ryabova_precision_score(df.actual_label.values,
(df.model_LR >= 0.75).astype('int').values)))
print('F1 LR: %.3f' % (ryabova_f1_score(df.actual_label.values, (df.model_LR >=
```

```
0.75).astype('int').values)))

print('\nscores with threshold = 0.25')
print('Accuracy RF: %.3f' % (ryabova_accuracy_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Recall RF: %.3f' % (ryabova_recall_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f' % (ryabova_precision_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('F1 RF: %.3f' % (ryabova_f1_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('Accuracy LR: %.3f' % (ryabova_accuracy_score(df.actual_label.values,
(df.model_LR >= 0.25).astype('int').values)))
print('Recall LR: %.3f' % (ryabova_recall_score(df.actual_label.values,
(df.model_LR >= 0.25).astype('int').values)))
print('Precision LR: %.3f' % (ryabova_precision_score(df.actual_label.values,
(df.model_LR >= 0.25).astype('int').values)))
print('F1 LR: %.3f' % (ryabova_f1_score(df.actual_label.values, (df.model_LR >=
0.25).astype('int').values)))

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)

auc_RF = roc_auc_score(df.actual_label.values, df.model_RF.values)
auc_LR = roc_auc_score(df.actual_label.values, df.model_LR.values)
print('AUC RF:%.3f' % auc_RF)
print('AUC LR:%.3f' % auc_LR)

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()
```

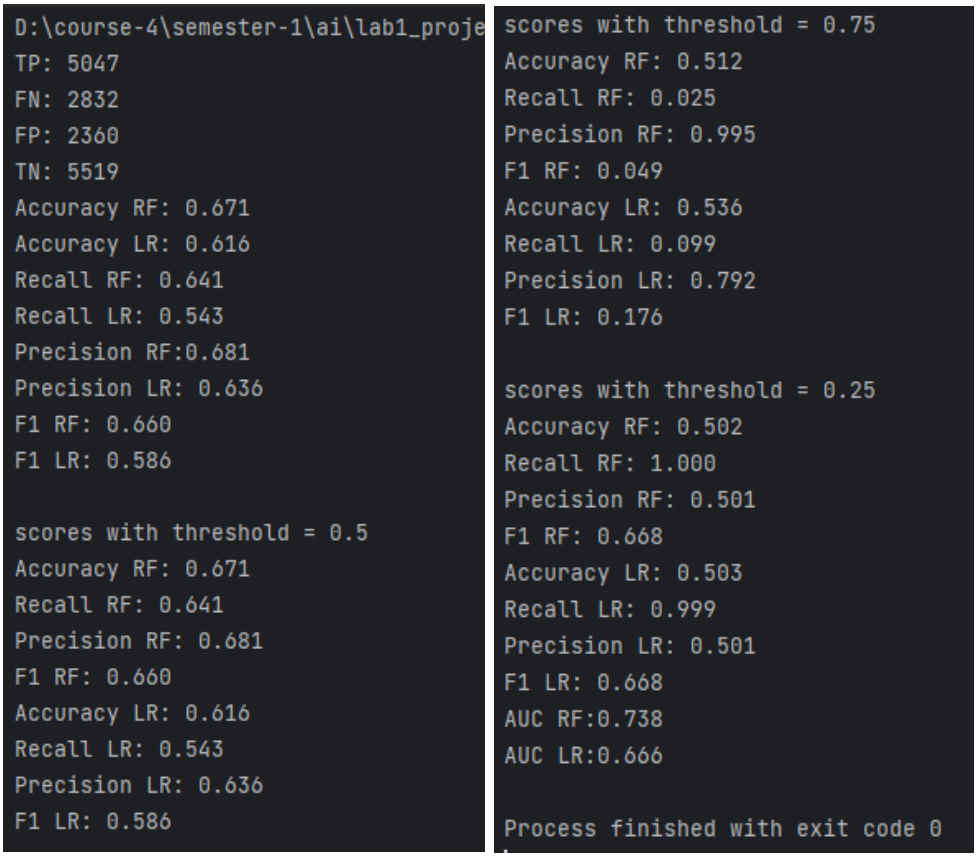


Рис.6.1 – 6.2. Результат виконання коду.

При зміні порогу відбуваються такі зміни у показниках:

збільшення порогу – для обох моделей зменшується частка правильно спрогнозованих вибірок (accuracy), правильно спрогнозованих позитивних подій (recall), середнє значення повноти і точності (f1), в той час збільшується частка спрогнозованих позитивних подій, які насправді є позитивними (precision). Тобто, більше позитивних подій правильно класифікуються.

зменшення порогу – як і при збільшенні, для обох моделей зменшуються accuracy, також precision, але збільшуються recall і доволі незначно f1, що значить, що більше подій класифікуються як позитивні, які насправді такими не є.

За графіком можна зробити висновок, що модель RF краще за LR, тому що ROC-крива RF знаходиться далі від чорної лінії, ніж ROC-крива LR, та площа під кривою для моделі RF більша.

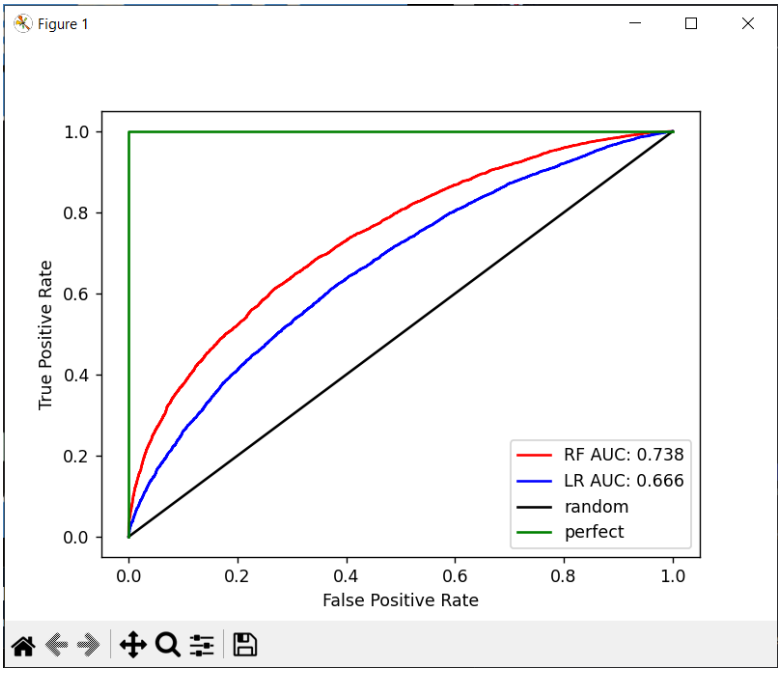


Рис.6.3. Результат виконання коду.

Завдання №7. Розробіть програму класифікації даних в файлі ata_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.

Лістинг файлу LR_1_task_6.py:

```
import numpy as np
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split, cross_val_score

from utilities import visualize_classifier

input_file = 'data_multivar_nb.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=3)

classifier_svm = SVC()
classifier_svm.fit(X, y)

classifier_g = GaussianNB()
classifier_g.fit(X, y)
```

```
# виведення показників
def print_stats(classifier, X, y):
    num_folds = 3
    stats = ["accuracy", "precision_weighted", "recall_weighted",
"f1_weighted"]

    print(classifier.__class__.__name__)

    for item in stats:
        item_val = cross_val_score(classifier, X, y, scoring=item,
cv=num_folds)
        print(item + ": " + str(round(100 * item_val.mean(), 2)) + "%")

    print()


print_stats(classifier_svm, X, y)
print_stats(classifier_g, X, y)

visualize_classifier(classifier_svm, X_test, y_test)
visualize_classifier(classifier_g, X_test, y_test)
```

Висновок: для початкових вхідних даних методи дали однаковий результат. Тому для вибору класифікатора варто звернути увагу на саме завдання та характеристики даних – SVM слід вибирати для складних задач класифікації з складними границями рішення та невеликими наборами даних з великою кількістю ознак, де важлива максимальна роздільність між класами, а наївний метод Баєса для простих даних, де ознаки вважаються незалежними, і задач, де вимагається швидка обробка даних.

```
D:\course-4\semester-1\ai\lab1_proj
SVC
accuracy: 99.75%
precision_weighted: 99.76%
recall_weighted: 99.75%
f1_weighted: 99.75%

GaussianNB
accuracy: 99.75%
precision_weighted: 99.76%
recall_weighted: 99.75%
f1_weighted: 99.75%


Process finished with exit code 0
```

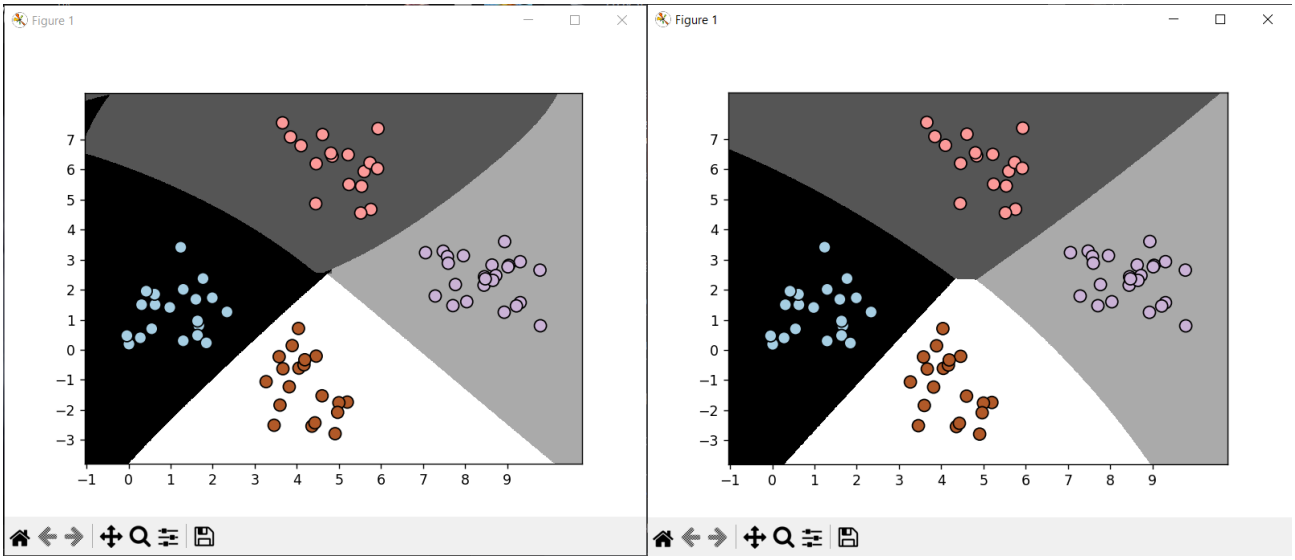


Рис.7.1 – 7.3. Результат виконання коду.

Висновки: в ході виконання лабораторної роботи було досліджено попередню обробку та класифікацію даних, використовуючи спеціалізовані бібліотеки та мову програмування Python.