ЛАБОРАТОРНА РОБОТА № 1

Тема: **«ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ»**

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

Хід роботи

Посилання на програмнй код на Github: https://github.com/NightSDay/AI-2023/tree/main/AI-subject/L-1_Classification/lab1

Завдання 2.1. **Попередня обробка даних**

Як правило, при обробці ми маємо справу з великими обсягами необроблених вихідних даних. Алгоритми машинного навчання розраховані на те, що, перш ніж вони зможуть розпочати процес тренування, отримані дані будуть відформатовані певним чином. Щоб привести дані до форми, що прийнятна для алгоритмів машинного навчання, ми повинні попередньо підготувати їх і перетворити на потрібний формат.

Розглянемо декілька різних методів попередньої обробки даних

Бінарізація — процес застосовується в тих випадках, коли ми хочемо перетворити наші числові значення на булеві значення (0, 1). Скористаємося вбудованим методом для бінаризації вхідних даних, встановивши значення 2.1 як порогове. Всі значення понад 2.1 примусово встановлюються рівними 1. Інші значення стають рівними 0.

Виключення середнього — методика попередньої обробки даних, що зазвичай використовується в машинному навчанні. Як правило, із векторів ознак (feature vectors) доцільно виключати середні значення, щоб кожна ознака (feature) центрувалася на нулі. Це робиться з метою, виключити з розгляду зміщення значень у векторах ознак.

Масштабування — у нашому векторі ознак кожне значення може змінюватись у деяких випадкових межах. Тому дуже важливо масштабувати ознаки, щоб вони були рівним ігровим полем для тренування алгоритму машинного навчання. Ми не хочемо, щоб будь-яка з ознак могла набувати штучно великого або малого значення лише через природу вимірів. Кожен рядок відмасштабований таким чином, щоб максимальним значенням була б 1, а всі решта значень визначалися відносно неї.

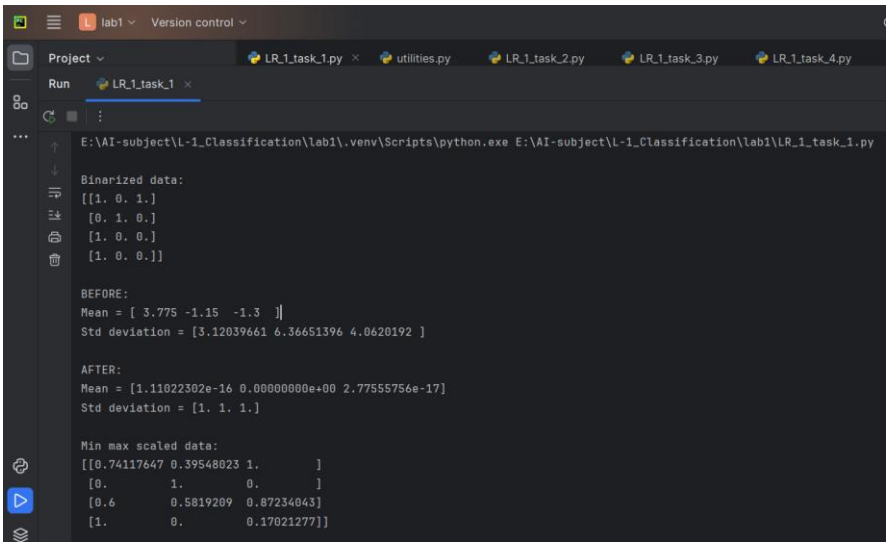
Нормалізація — Процес нормалізації полягає у зміні значень у векторі ознак таким чином, щоб для їх вимірювання можна було використовувати одну загальну шкалу. У машинному навчанні використовують різні форми нормалізації. У найбільш поширених з них, значення змінюються так, щоб їх сума дорівнювала 1. **L1-нормалізація** використовує метод найменших абсолютних відхилень (Least Absolute Deviations), що забезпечує рівність 1 суми абсолютних значень в кожному ряду. **L2-нормалізація** використовує метод найменших квадратів, що забезпечує рівність 1 суми квадратів значень. Взагалі, техніка *L1-нормалізації* вважається більш надійною по порівняно з *L2-нормалізацією*, оскільки вона менш чутлива до викидів. Дуже часто дані містять викиди, і з цим нічого не вдієш. Ми хочемо використовувати безпечні методики, що дозволяють ігнорувати викиди у процесі обчислень. Якби ми вирішували завдання, в якому викиди грають важливу роль, то, ймовірно, найкращим вибором була б *L2-нормалізація*.

Лістинг програмного коду до завдань 2.1.1 – 2.1.4.:

```
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
data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
print(f"\nBinarized data:\n{data_binarized}")

# Середнє значення та стандартне відхилення
print("\nBEFORE: ")
print(f"Mean = {input_data.mean(axis=0)}")
print(f"Std deviation = {input_data.std(axis=0)}")
```

```
# виключення середнього
data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print(f"Mean = {data_scaled.mean(axis=0)}")
print(f"Std deviation = {data_scaled.std(axis=0)}")

# Масштабування
data_scaled_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled_minmax = data_scaled_minmax.fit_transform(input_data)
print(f"\nMin max scaled data:\n{data_scaled_minmax}")

# Нормалізація
data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
print(f"\nL1 normalized data:\n{data_normalized_l1}")
print(f"\nL2 normalized data:\n{data_normalized_l2}")
```

Результат виконання програми:




```
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. 1.1. – 1.2. Результат виконання програми

Висновок чим відрізняються L1-нормалізація від L2-нормалізації

Основна різниця полягає у відношенні до значень. L1-нормалізація використовує абсолютні значення, які забезпечують меншу чутливість до великих відхилень. З іншого боку, L2-нормалізація оперує квадратичними значеннями, що робить її більш чутливою до великих відхилень, оскільки вони впливають на суму квадратів значень значно сильніше. Отже, L1-нормалізація менш реагує на великі відхилень, але може втратити частину інформації про розподіл значень, тоді як L2-нормалізація точніше враховує великі відхилення, але стає більш чутливою до них.

Кодування міток — це процес перетворення категоріальних міток або текстових даних у числові значення. Це часто використовується в машинному навчанні, оскільки багато алгоритмів працюють тільки з числовими даними. Наприклад, якщо у вас є стовпець з категоріями "Червоний", "Синій" та "Зелений", то після кодування міток може з'явитися стовпець з числовими значеннями, наприклад "0", "1" та "2", де кожна цифра відповідає відповідній категорії. Це важливий крок у підготовці даних для багатьох моделей машинного навчання, оскільки дозволяє включити категоріальні дані в процес навчання.

Лістинг програмного коду до завдання 2.1.5:

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

# Кодувальник 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 = ['red', 'green', 'black']
encoded_values = encoder.transform(test_labels)
print(f"\nLabels = {test_labels}")
print(f"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(f"\nEncoded values = {encoded_values}")
print(f"Decoded labels = {list(decoded_list)}")
```

Результат виконання програми:

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

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

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

Process finished with exit code 0
```

Рис. 2. Результат виконання програми

Отже, в результаті бачимо масиви: початковий масив числових значень та масив, де ці числа були декодовані назад у текстові мітки. Наприклад, "3" може бути декодовано назад у "green", і так далі. Як бачимо алгоритм працює коректно навіть при дублюванні міток (деякі значення зустрічаються більше 1 разу у початковому масиві).

Завдання 2.2. **Попередня обробка нових даних**

У коді програми попереднього завдання поміняйте дані по рядках (значення змінної input_data) на значення відповідно варіанту таблиці 1 та виконайте операції: Бінарізації, Виключення середнього, Масштабування, Нормалізації.

Варіант обирається відповідно номера за списком групи відповідно до таблиці 1.

Таблиця 1

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

Лістнг програми:

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


input_data = np.array([[-3.3, -1.6, 6.1],
                       [2.4, -1.2, 4.3],
                       [-3.2, 5.5, -6.1],
                       [-4.4, 1.4, -1.2]])

troubleshold = 2.1

# Бінаризація даних
data_binarized =
preprocessing.Binarizer(threshold=troubleshold).transform(input_data)
print(f"\nBinarized data:\n{data_binarized}")

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

# Виключення середнього
data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print(f"Mean = {data_scaled.mean(axis=0)}")
print(f"Std deviation = {data_scaled.std(axis=0)}")

# Масштабування MinMax
data_scaled_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled_minmax = data_scaled_minmax.fit_transform(input_data)
print(f"\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(f"\nL1 normalized data:\n{data_normalized_l1}")
print(f"\nL2 normalized data:\n{data_normalized_l2}")
```

Результат виконання програми:

```
E:\AI-subject\L-1_Classification\lab1\.venv\Scripts\python.exe E:\AI-subject\L-1_Classification\lab1\LR_1_task_2.py

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]]
```

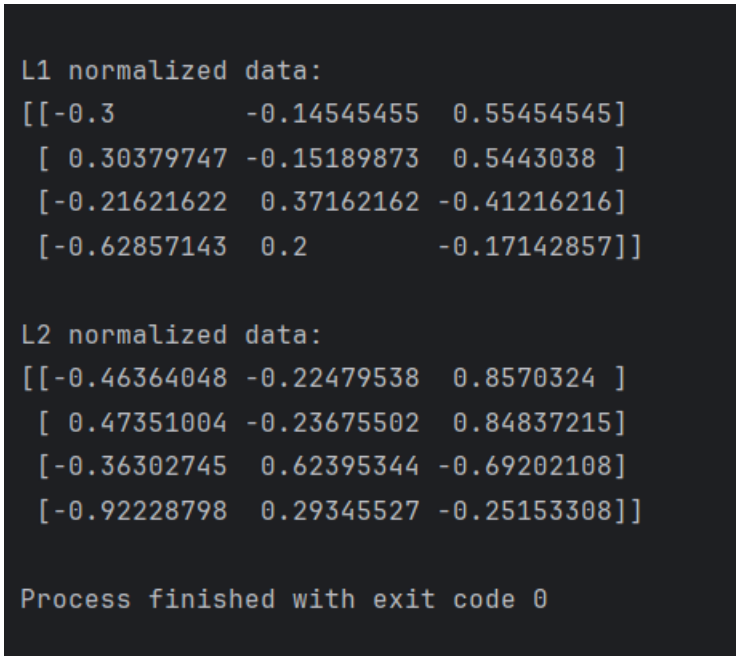


Рис. 3. Результат виконання програми

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

Логістична регресія (logistic regression) - це методика, що використовується для пояснення відносин між вхідними та вихідними змінними. Вхідні змінні вважаються незалежними, вихідні – залежними. Залежна змінна може мати лише фіксований набір значень. Ці значення відповідають класам завдання класифікації. Метою є ідентифікація відносин між незалежними та залежними змінними за допомогою оцінки ймовірностей того, що та або інша залежна змінна відноситься до того чи іншого класу. За своєю природою логістична функція є сигмоїдою, що використовується для створення функцій з різними параметрами. Вона тісно пов’язана з аналізом даних на основі узагальненої лінійної моделі, у відповідності до якої робиться спроба підігнати пряму лінію до групи точок таким чином, щоб мінімізувати помилку. Замість лінійної регресії ми застосовуємо логістичну регресію. В дійсності сама по собі логістична регресія призначена не для класифікації даних, проте вона дозволяє спростити вирішення цього завдання.

Лістинг файлу *utilities.py*:

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


def visualize_classifier(classifier, X, y):
    # Define the minimum and maximum values for X and Y
    # that will be used in the mesh grid
    min_x, max_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
    min_y, max_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0

    # Define the step size to use in plotting the mesh grid
    mesh_step_size = 0.01

    # Define the mesh grid of X and Y values
    x_vals, y_vals = np.meshgrid(np.arange(min_x, max_x, mesh_step_size),
np.arange(min_y, max_y, mesh_step_size))

    # Run the classifier on the mesh grid
    output = classifier.predict(np.c_[x_vals.ravel(), y_vals.ravel()])

    # Reshape the output array
    output = output.reshape(x_vals.shape)

    # Create a plot
    plt.figure()

    # Choose a color scheme for the plot
    plt.pcolormesh(x_vals, y_vals, output, cmap=plt.cm.gray)

    # Overlay the training points on the plot
    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='black', linewidth=1,
cmap=plt.cm.Paired)

    # Specify the boundaries of the plot
    plt.xlim(x_vals.min(), x_vals.max())
    plt.ylim(y_vals.min(), y_vals.max())

    # Specify the ticks on the X and Y axes
    plt.xticks((np.arange(int(X[:, 0].min() - 1), int(X[:, 0].max() + 1), 1.0)))
    plt.yticks((np.arange(int(X[:, 1].min() - 1), int(X[:, 1].max() + 1), 1.0)))

    plt.show()
```

Лістинг власне програми:

```
import numpy as np
from sklearn import linear_model
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)
```

Результат виконання програми:

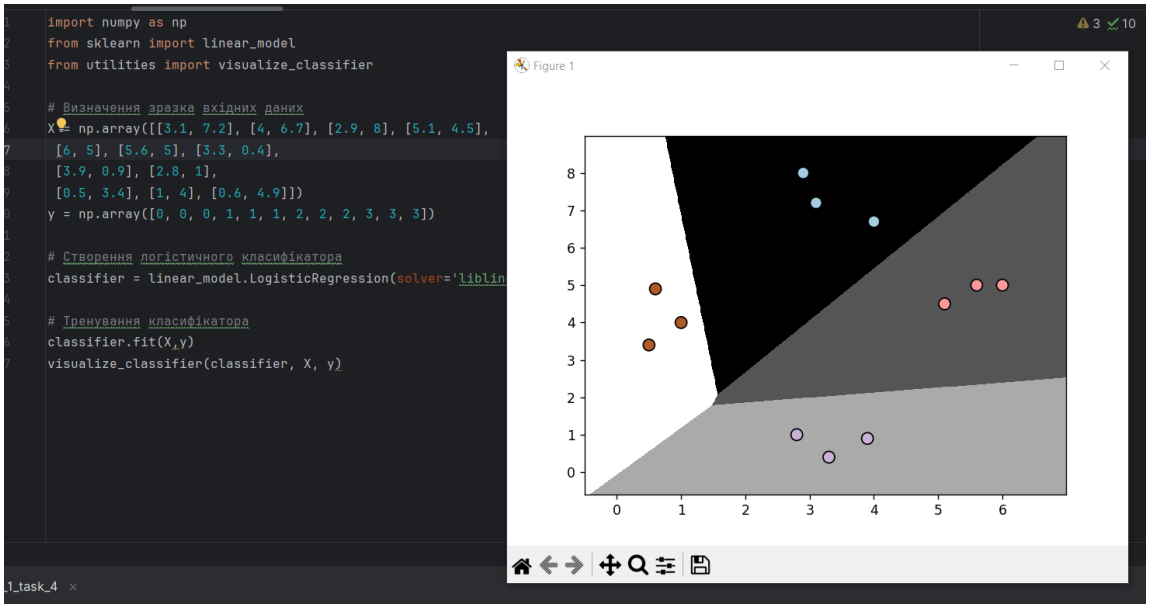


Рис. 4. Результат виконання програми

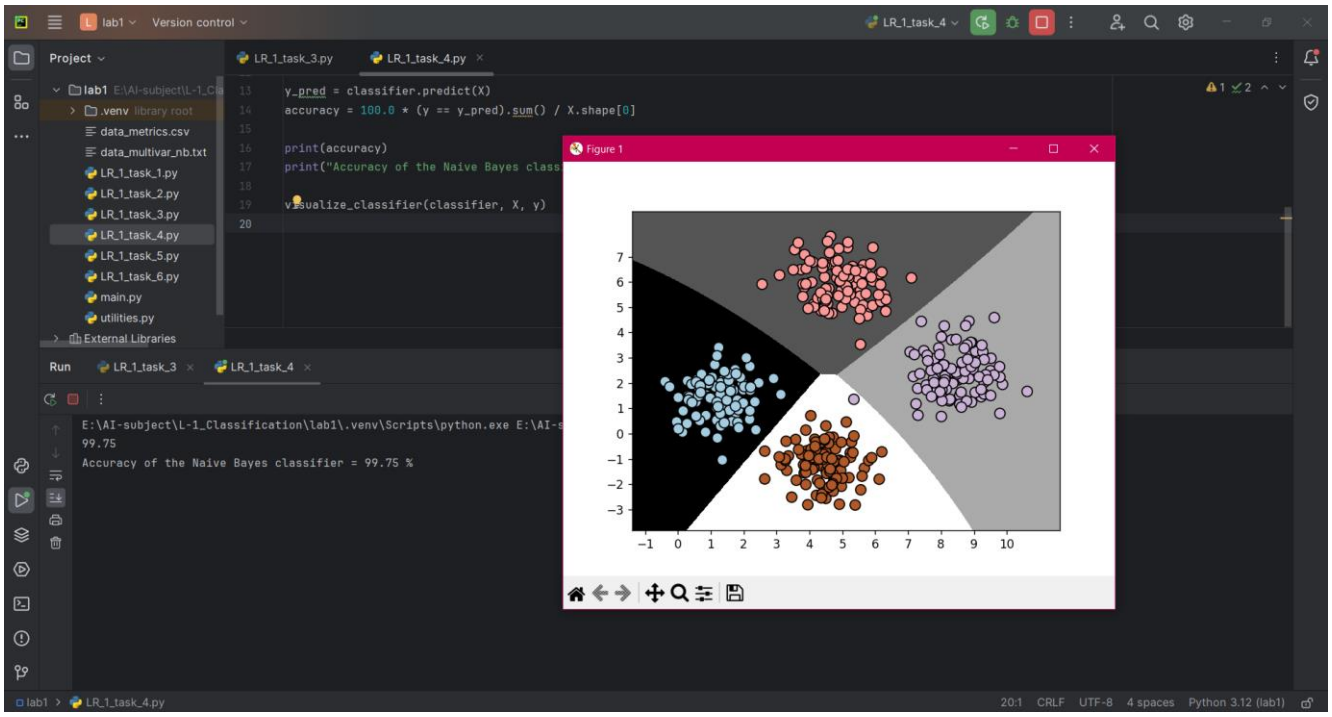
Завдання 2.4. Класифікація наївним байєсовським класифікатором

Наївний байєсовський класифікатор (Naive Bayes classifier) — це простий класифікатор, заснований на використанні теореми Байєса, яка описує ймовірність події з урахуванням пов'язаних з нею умов. Такий класифікатор створюється за допомогою привласнення позначок класів екземплярам завдання. Останні представляються як векторів значень ознак. При цьому передбачається, що значення будь-якої заданої ознаки не залежить від значень інших ознак. Його припущення про незалежність ознак і становить наївну частину байєсовського класифікатора. Ми можемо оцінювати вплив будь-якої ознаки змінної класу незалежно від впливу інших ознак.

Лістинг програми:

```
import numpy as np
from sklearn.naive_bayes import GaussianNB
from utilities import visualize_classifier

input_file = 'data_multivar_nb.txt'

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

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

y_pred = classifier.predict(X)
accuracy = 100.0 * (y == y_pred).sum() / X.shape[0]

print(accuracy)
print("Accuracy of the Naive Bayes classifier =", round(accuracy, 2), "%")

visualize_classifier(classifier, X, y)
```



Лістинг модифікованої програми:

```
import numpy as np
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 = GaussianNB()
classifier.fit(X_train, y_train)

y_test_pred = classifier.predict(X_test)
accuracy = 100.0 * (y_test == y_test_pred).sum() / X_test.shape[0]

print(accuracy)
print("Accuracy of the new Naive Bayes classifier =", round(accuracy, 2), "%")

visualize_classifier(classifier, X_test, y_test)
```

```
num_folds = 3
accuracy_values = cross_val_score(classifier, X, y, scoring = 'accuracy', cv =
num_folds)
print("Acuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")

precision_values = cross_val_score(classifier, X, y, scoring='precision_weighted',
cv=num_folds)
print("Preccision: " + 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)) + "%")
```

Результат виконання програми:

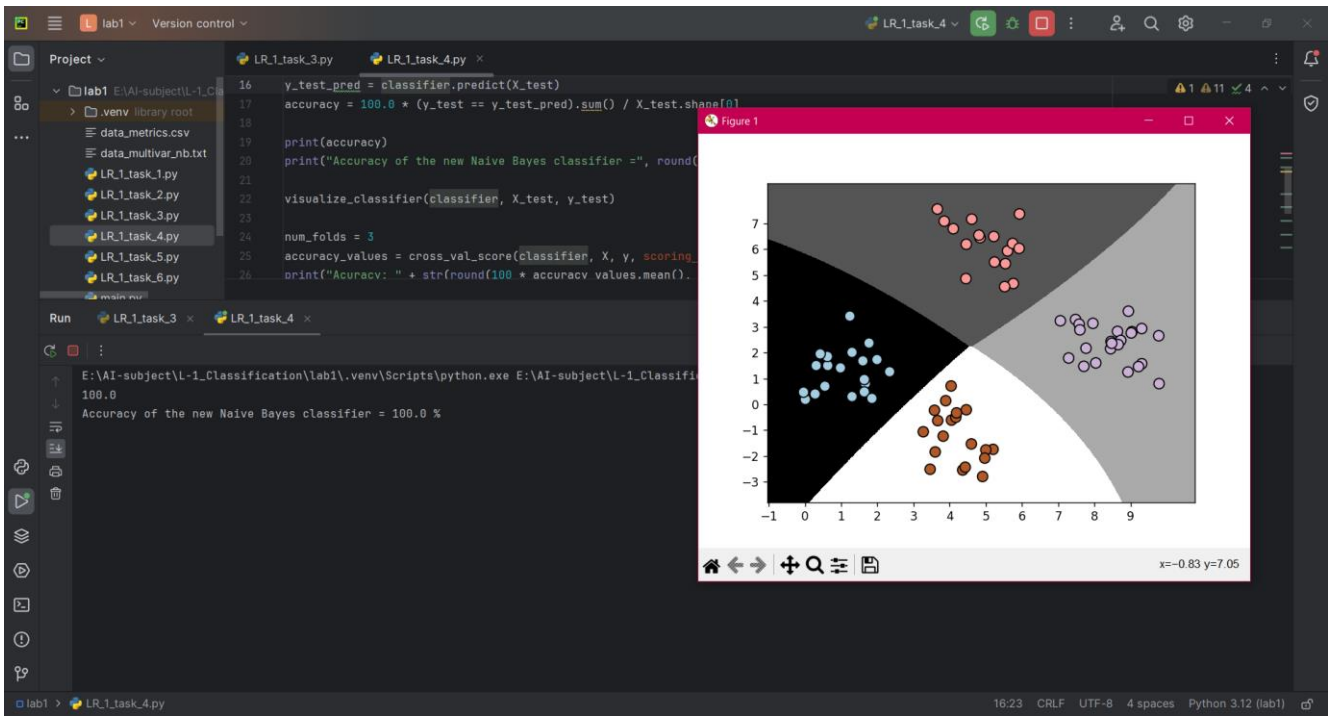


Рис. 6. Результат виконання програми

Порівняння результатів

У першому випадку була досягнута точність на рівні 99.75%. У другому випадку, після застосування перехресної перевірки та поділу даних на тестові та тренувальні, точність підвищилася до 100%. Отже, можна зробити висновок, що перший метод є менш надійним.

Завдання 2.5. **Вивчити метрики якості класифікації**

Класифікація полягає у спробі передбачити, з якого класу надходить конкретна вибірка із популяції.

У цьому завданні ми розглянемо деякі з цих метрик і напишемо власні функції з нуля, щоб зрозуміти математику, що лежить в основі деяких з них.

Запрограмуємо наступні показники зі sklearn.metrics:

confusion_matrix – матриця помилок (або матриця неточностей чи плутанини);

accuracy_score – акуратність (з англ. може перекласти як точність, але не плутайте бо то інший показник)

recall_score – повнота

precision_score – точність

f1_score – F-міра

roc_curve – ROC-крива, крива робочих характеристик (англ. Receiver Operating Characteristics curve).

roc_auc_score – вимір площі під ROC-кривою (англ. Area Under the Curve - AUC). (ROC-AUC)

Лістинг програми:

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

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

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 radchenko_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]])


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

assert np.array_equal(radchenko_confusion_matrix(df.actual_label.values,
df.predicted_RF.values),
                      confusion_matrix(df.actual_label.values,
df.predicted_RF.values))

def radchenko_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 radchenko_accuracy_score(df.actual_label.values, df.predicted_RF.values) ==
accuracy_score(df.actual_label.values,

df.predicted_RF.values), 'radchenko_accuracy_score failed on RF'

assert radchenko_accuracy_score(df.actual_label.values, df.predicted_LR.values) ==
accuracy_score(df.actual_label.values,

df.predicted_LR.values), 'radchenko_accuracy_score failed on LR'

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

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


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


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

df.predicted_RF.values), 'radchenko_recall_score failed on RF'
```

```
assert radchenko_recall_score(df.actual_label.values, df.predicted_LR.values) ==
recall_score(df.actual_label.values,

df.predicted_LR.values), 'radchenko_recall_score failed on LR'

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

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


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


assert radchenko_precision_score(df.actual_label.values, df.predicted_RF.values)
== precision_score(
    df.actual_label.values, df.predicted_RF.values), 'bubenko_precision_score
failed on RF'

assert radchenko_precision_score(df.actual_label.values, df.predicted_LR.values)
== precision_score(
    df.actual_label.values, df.predicted_LR.values), 'bubenko_precision_score
failed on LR'

print('Precision RF: %.3f' % (radchenko precision score(df.actual label.values,
df.predicted_RF.values)))
print('Precision LR: %.3f' % (radchenko_precision_score(df.actual_label.values,
df.predicted_LR.values)))

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


def radchenko_f1_score(y_true, y_pred):
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    recall = radchenko_recall_score(y_true, y_pred)
    precision = radchenko_precision_score(y_true, y_pred)
    return 2 * (recall * precision) / (recall + precision)


assert radchenko_f1_score(df.actual_label.values, df.predicted_RF.values) ==
f1_score(df.actual_label.values,

df.predicted_RF.values), 'radchenko_f1_score failed on RF'
assert radchenko_f1_score(df.actual_label.values, df.predicted_LR.values) ==
f1_score(df.actual_label.values,

df.predicted_LR.values), 'radchenko_f1_score failed on LR'

print('F1 RF: %.3f' % (radchenko_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 LR: %.3f' % (radchenko_f1_score(df.actual_label.values,
df.predicted_LR.values)))

print('\nscores with threshold = 0.5')
print('Accuracy RF: %.3f' % (radchenko_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall RF: %.3f' % (radchenko_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF: %.3f' % (radchenko_precision_score(df.actual_label.values,
```

```
df.predicted_RF.values)))
print('F1 RF: %.3f' % (radchenko_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('Accuracy LR: %.3f' % (radchenko_accuracy_score(df.actual_label.values,
df.predicted_LR.values)))
print('Recall LR: %.3f' % (radchenko_recall_score(df.actual_label.values,
df.predicted_LR.values)))
print('Precision LR: %.3f' % (radchenko_precision_score(df.actual_label.values,
df.predicted_LR.values)))
print('F1 LR: %.3f' % (radchenko_f1_score(df.actual_label.values,
df.predicted_LR.values)))
print('')

print('scores with threshold = 0.25')
print(
    'Accuracy RF: %.3f' % (radchenko_accuracy_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Recall RF: %.3f' % (radchenko_recall_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f' % (
    radchenko_precision_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print('F1 RF: %.3f' % (radchenko_f1_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))
print(
    'Accuracy LR: %.3f' % (radchenko_accuracy_score(df.actual_label.values,
(df.model_LR >= 0.25).astype('int').values)))
print('Recall LR: %.3f' % (radchenko_recall_score(df.actual_label.values,
(df.model_LR >= 0.25).astype('int').values)))
print('Precision LR: %.3f' % (
    radchenko_precision_score(df.actual_label.values, (df.model_LR >=
0.25).astype('int').values)))
print('F1 LR: %.3f' % (radchenko_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()
```

Результат виконання програми:

```
E:\AI-subject\L-1_Classification\lab1\.venv\Scripts\python.exe E:\AI-subject\L-1_Classification\lab1\LR_1_task_5.py
TP: 5047
FN: 2832
FP: 2360
TN: 5519
Accuracy RF: 0.671
Accuracy LR: 0.616
Recall RF: 0.641
Recall LR: 0.543
Precision RF: 0.681
Precision LR: 0.636
F1 RF: 0.660
F1 LR: 0.586

scores with threshold = 0.5
Accuracy RF: 0.671
Recall RF: 0.641
Precision RF: 0.681
F1 RF: 0.660
Accuracy LR: 0.616
Recall LR: 0.543
Precision LR: 0.636
F1 LR: 0.586
```

Рис. 8. Результат виконання програми

Висновок щодо результатів для різних порогів (0.5 та 0.25)

Отже, з підвищенням порогу значення міри F1 зменшується. При зниженні порогу до 0.25 спостерігається збільшення кількості виявлених позитивних випадків, що впливає на підвищення значення Recall. Однак це призводить до зниження точності моделі, що відображається у вищих значеннях False Positives та зменшенні Precision.

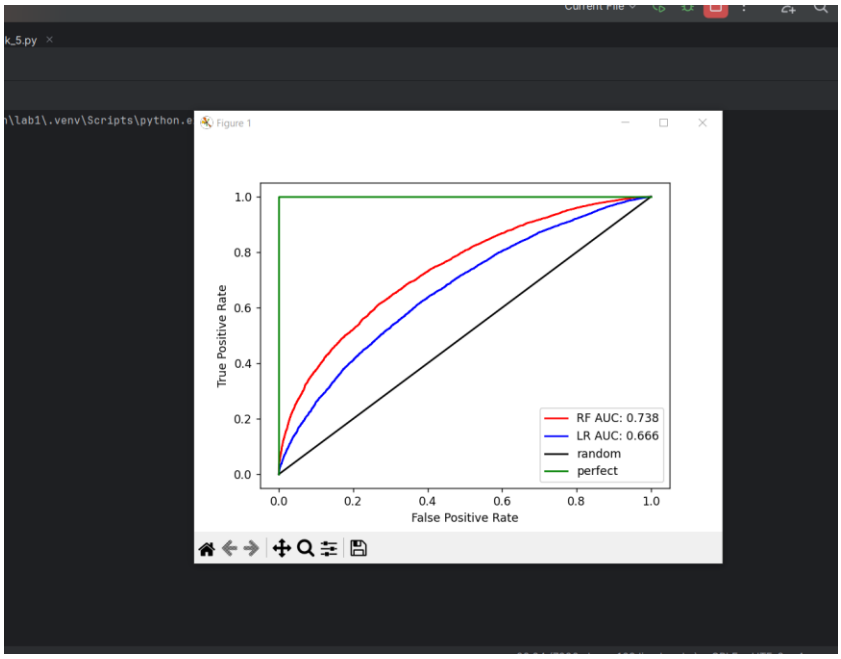


Рис. 9. Результат виконання програми (ROC-крива)

Висновок щодо того, яка з двох моделей (LR чи RF) краща і чому

Отже, аналізуючи графік, можна відзначити, що RF модель виявляється більш точною, ніж LR модель. RF має кращі можливості у відокремленні класів та забезпечує більш ефективну роботу в цій задачі класифікації. Проте, можуть виникнути ситуації, коли LR матиме переваги перед RF, тому важливо враховувати складність моделі.

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.**

Лістинг програми:

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


def evaluate_classifier(classifier, X, y, num_folds=3):
    accuracy_values = cross_val_score(classifier, X, y, scoring='accuracy',
cv=num_folds)
    precision_values = cross_val_score(classifier, X, y,
scoring='precision_weighted', cv=num_folds)
    recall_values = cross_val_score(classifier, X, y, scoring='recall_weighted',
cv=num_folds)
    f1_values = cross_val_score(classifier, X, y, scoring='f1_weighted',
cv=num_folds)

    accuracy_mean = round(100 * accuracy_values.mean(), 2)
    precision_mean = round(100 * precision_values.mean(), 2)
    recall_mean = round(100 * recall_values.mean(), 2)
    f1_mean = round(100 * f1_values.mean(), 2)

    return accuracy_mean, precision_mean, recall_mean, f1_mean


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_train, y_train)

classifier_nb = GaussianNB()
classifier_nb.fit(X_train, y_train)

num_folds = 3

print('Naive Bayes')
nb_accuracy, nb_precision, nb_recall, nb_f1 = evaluate_classifier(classifier_nb,
X, y, num_folds)
print("Accuracy:", nb_accuracy, "%")
print("Precision:", nb_precision, "%")
print("Recall:", nb_recall, "%")
print("F1:", nb_f1, "%")

print('\nSVM')
svm_accuracy, svm_precision, svm_recall, svm_f1 =
evaluate_classifier(classifier_svm, X, y, num_folds)
print("Accuracy:", svm_accuracy, "%")
print("Precision:", svm_precision, "%")
print("Recall:", svm_recall, "%")
print("F1:", svm_f1, "%")

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

Результат виконання програми:

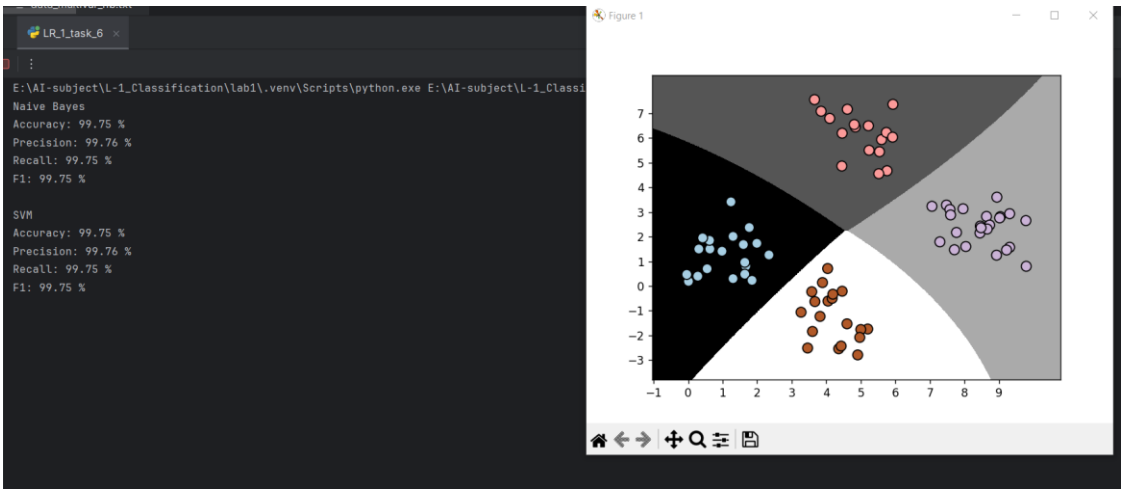


Рис. 10. - 12. Результат виконання програми

Висновки про те, яку модель класифікації краще обрати і чому

У нашому прикладі показники SVM та наївного байєсівського класифікатора виявилися подібними, тому в даному випадку можна обирати як першу, так і другу модель класифікації. Проте, вибір конкретної моделі залежить від багатьох факторів, таких як обсяг даних, складність задачі та метрики успішності. SVM зазвичай ефективний в задачах з великою кількістю ознак, де присутня чітка границя між класами. Однак його головний недолік полягає у тому, що він застосову-

ється лише до задач з двома класами. Наївний баєсівський класифікатор може бути дуже ефективним у простих класифікаційних завданнях та у випадках, коли взаємозв'язки між ознаками є простими..

Висновок: у ході виконання лабораторної роботи я, використовуючи спеціалізовані бібліотеки та мову програмування Python, дослідив попередню обробку та класифікацію даних, зокрема, L1-нормалізацію та L2-нормалізацію, методи класифікації та різницю між моделями RF і LR.