ЛАБОРАТОРНА РОБОТА № 1
ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

Хід роботи:

Завдання 2.1-2.1.4. Попередня обробка даних.

LR_1_task_1.py

```
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. Відображення коду програм.

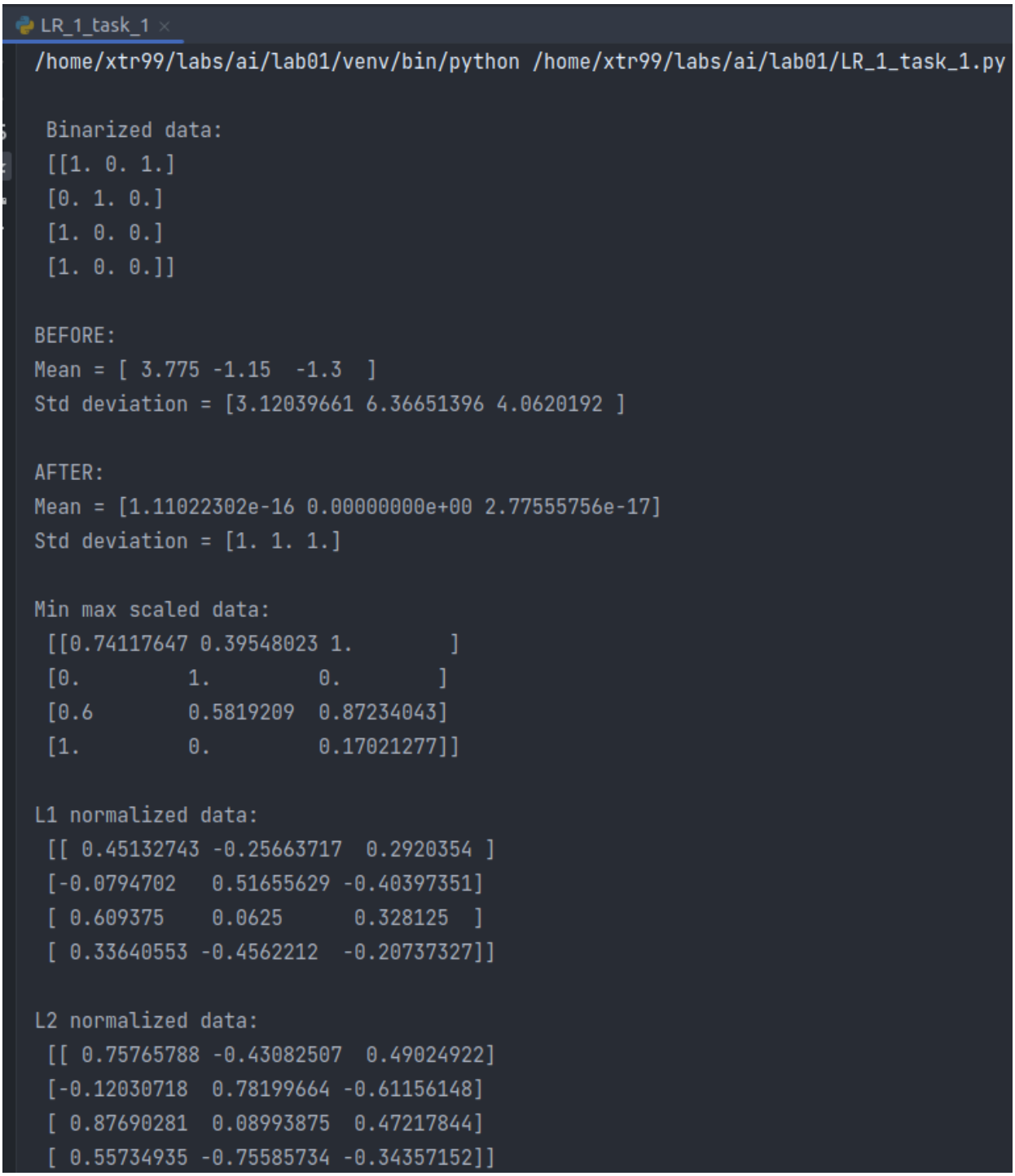

Рис. 2. Виконання коду програм.

L1-нормалізація використовує метод найменших абсолютних відхилень для забезпечення того, що сума абсолютних значень вектора буде дорівнювати 1. За своєю суттю, L1-нормалізація вважається більш надійною порівняно з L2-нормалізацією, оскільки вона менш чутлива до викидів або значень, що відхиляються від середнього значення.

L2-нормалізація, натомість, використовує метод найменших квадратів для забезпечення того, що сума квадратів значень вектора буде дорівнювати 1. Хоча L2-нормалізація також є ефективним методом нормалізації, вона може бути більш чутливою до викидів, що можуть впливати на результуючий вектор.

Отже, L1-нормалізація вважається більш надійною технікою порівняно з L2-нормалізацією через її меншу чутливість до викидів або відхилень від середнього значення.

Для прикладу L1-нормалізації та L2-нормалізації скористаємося даними з рис. 2. рядками 4 масиву.

Результат для L1-нормалізації: |0.33640553|+|-0.4562212| +|-0.20737327| = 1

Результат для L2-нормалізації: 0.55734935^2 + (-0.75585734)^2 + (-0.34357152)^2 = 1

Завдання 2.1.5. Кодування міток.

```python
# 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))
```

Рис. 3. Відображення коду програми.

```
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']
```

Рис. 4. Результат виконання програми.

Як можна побачити на рис.4 кожній мітці присвоєне число за допомогою якого можна закодувати та декодувати значення мітки.

Завдання 2.2. Попередня обробка нових даних.

Таблиця 1

№ варіанту	Значення змінної input_data												Поріг бінаризації
8.	4.6	9.9	-3.5	-2.9	4.1	3.3	-2.2	8.8	-6.1	3.9	1.4	2.2	2.2

LR_1_task_2.py

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

input_data = np.array([[4.6, 9.9, -3.5],
                       [-2.9, 4.1, 3.3],
                       [-2.2, 8.8, -6.1],
                       [3.9, 1.4, 2.2]])
# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
print("\n Binarized data:\n", data_binarized)
# Виключення середньогоI
data_scaled = preprocessing.scale(input_data)
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. 5. Відображення коду програм.

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

Рис. 6. Результат бінарізації даних.

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

Рис. 7. Результат виключення середнього.

```
Min max scaled data:
 [[1.         1.         0.27659574]
 [0.         0.31764706 1.        ]
 [0.09333333 0.87058824 0.        ]
 [0.90666667 0.         0.88297872]]
```

Рис. 8. Результат масштабування MinMax.

```
L1 normalized data:
 [[ 0.25555556  0.55       -0.19444444]
 [-0.2815534   0.39805825  0.32038835]
 [-0.12865497  0.51461988 -0.35672515]
 [ 0.52        0.18666667  0.29333333]]

L2 normalized data:
 [[ 0.40126114  0.86358375 -0.30530739]
 [-0.4825966   0.68229174  0.54916164]
 [-0.20125974  0.80503895 -0.55803836]
 [ 0.83129388  0.29841319  0.46893501]]
```

Рис. 9. Результат нормалізації L1 та L2.

Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор

LR_1_task_3.py

```python
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)
```

Рис. 10. Відображення коду програми.

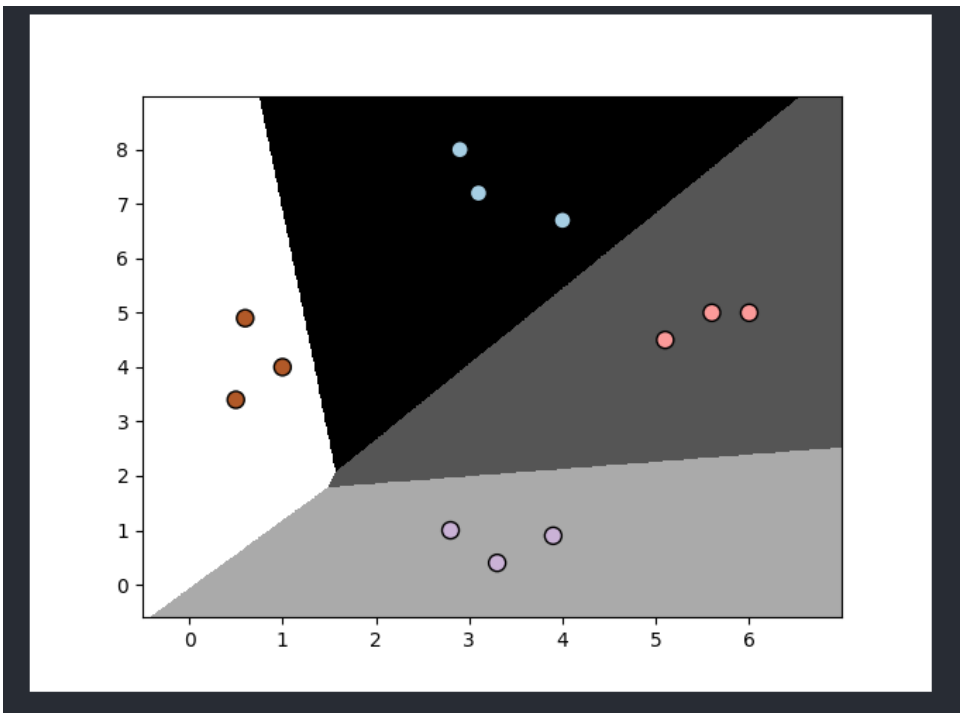

Рис. 11. Результат виконання програми.

Завдання 2.4. Класифікація наївним байєсовським класифікатором

LR_1_task_4.py

```python
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

# 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)
```

Рис. 12. Відображення коду програми.

LR_1_task_4

```
/home/xtr99/labs/ai/lab01/venv/bin/python /home/xtr99/labs/ai/lab01/LR_1_task_4.py
Accuracy of Naive Bayes classifier = 99.75 %
```

Рис. 13. Результат якості класифікатора.

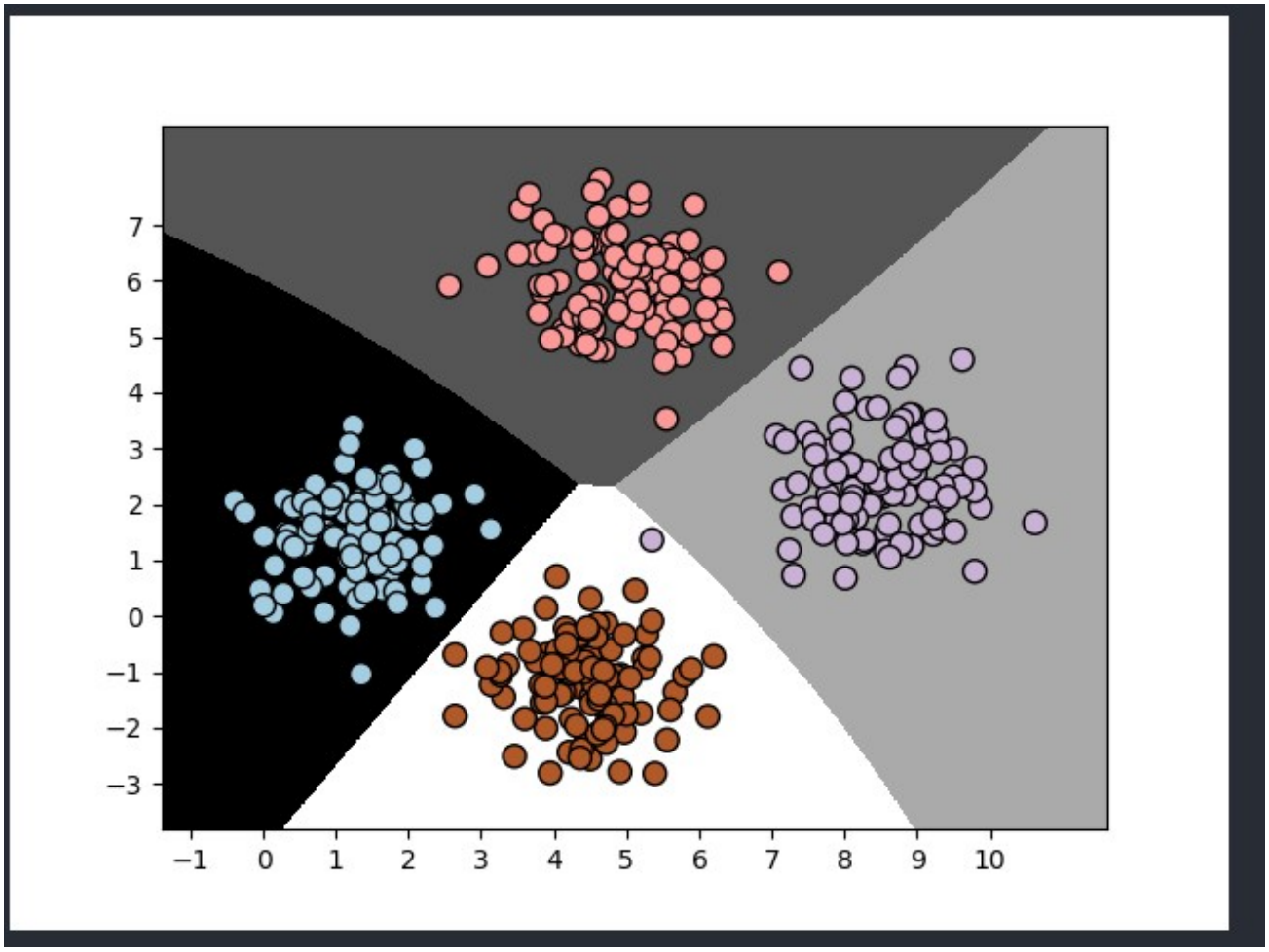

Рис. 14. Результат виконання програми.

```
# 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)
```

Рис. 15. Додавання до програми обчислення якості та візуалізації робити класифікатора.

```
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)) + "%")
```

Рис. 16. Додавання до програми вбудовані функції для обчислення якості (accuracy), точності (precision) 2 та повноти (recall) 3 класифікатора на підставі потрійний перехрестної перевірки.

```
Accuracy of Naive Bayes classifier = 99.75 %
Accuracy of the new classifier = 100.0 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

Рис. 17. Результат обчислення якості (accuracy), точності (precision) 2 та повноти (recall) 3 класифікатора на підставі потрійний перехрестної перевірки.

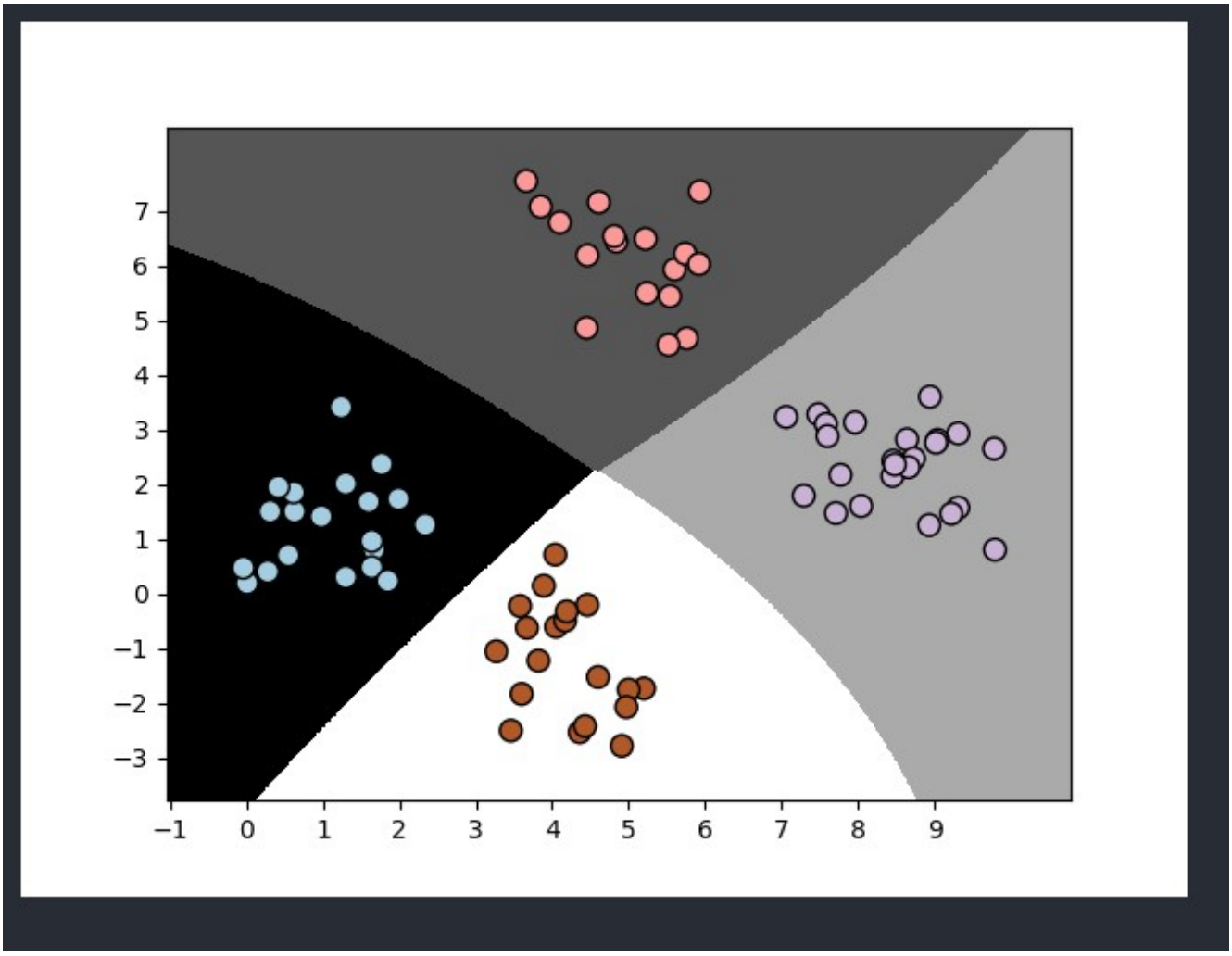

Рис. 18. Результат виконання програми.

```
LR_1_task_4
/home/xtr99/labs/ai/lab01/venv/bin/python /home/
Accuracy of Naive Bayes classifier = 99.75 %
Accuracy of the new classifier = 100.0 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%

Process finished with exit code 0
```

Рис. 19. Повторний результат обчислення якості (accuracy), точності (precision) 2 та повноти (recall) 3 класифікатора на підставі потрійний перехрестної перевірки.

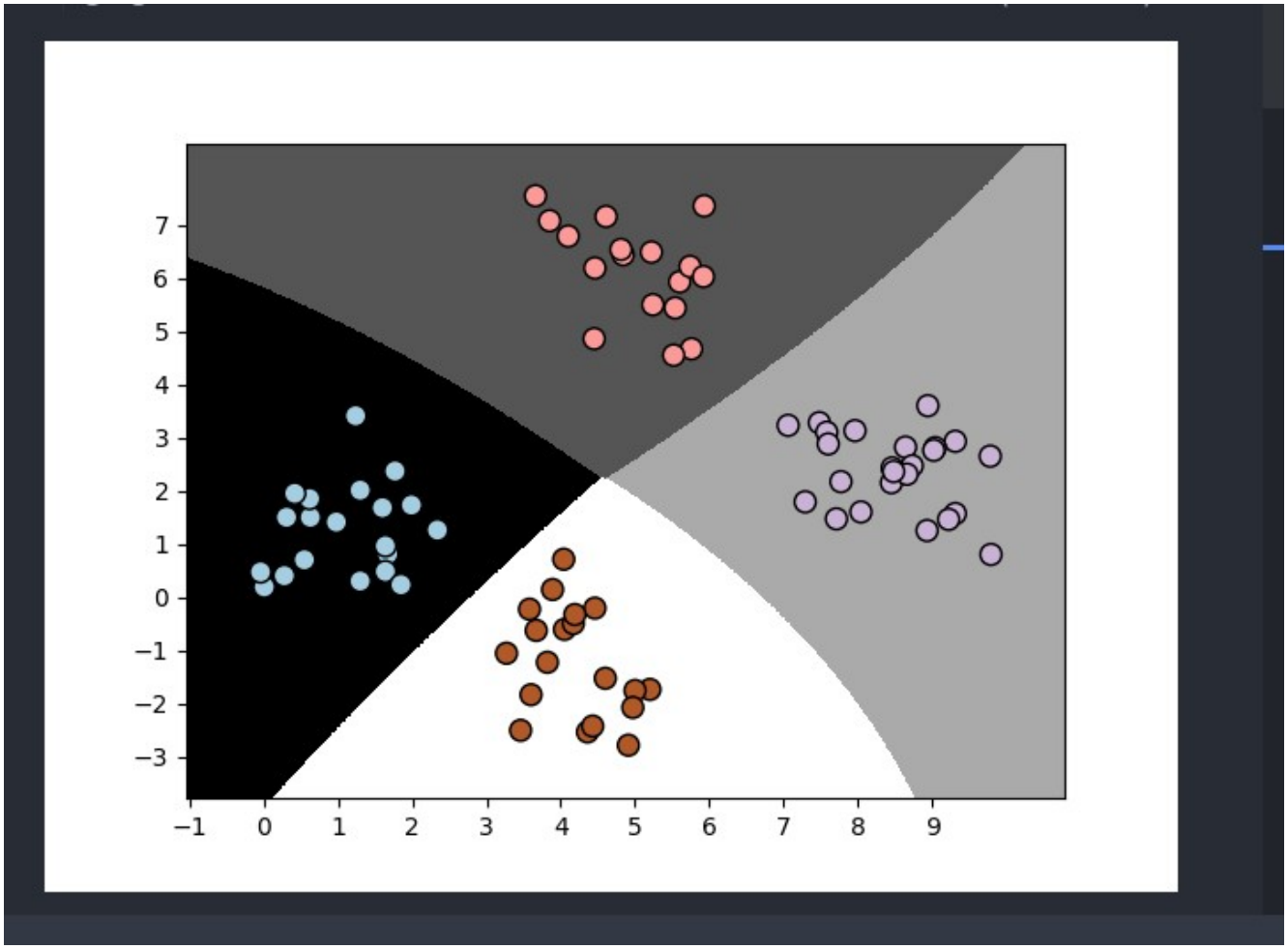

Рис. 20. Повторний результат виконання програми.

Отже, як можна побачити з рис.17-20 результати виконань повністю співпадають, тобто класифікаціє точна.

Завдання 2.5. Вивчити метрики якості класифікації

LR_1_task_5.py

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

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

threshold = 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)


2 usages
def kulish_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))


2 usages
def kulish_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))
```

Рис. 21. Відображення частини коду програми.

```
/home/xtr99/labs/ai/lab01/venv/bin/python /home/xtr99/labs/ai/lab01/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
```

Рис. 22. Результати виконання програми.

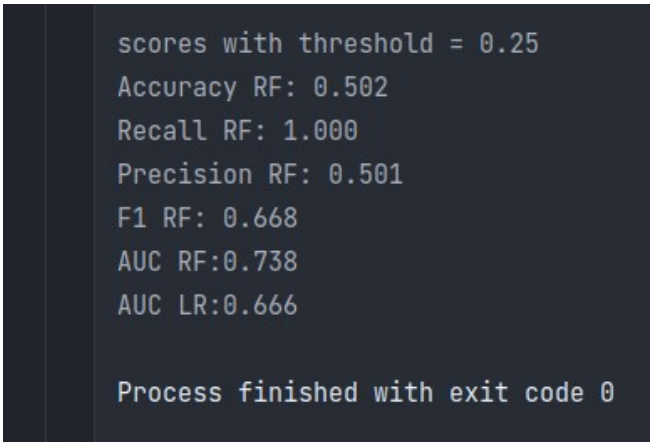

Рис. 23. Результат виконання для порогу 0.25

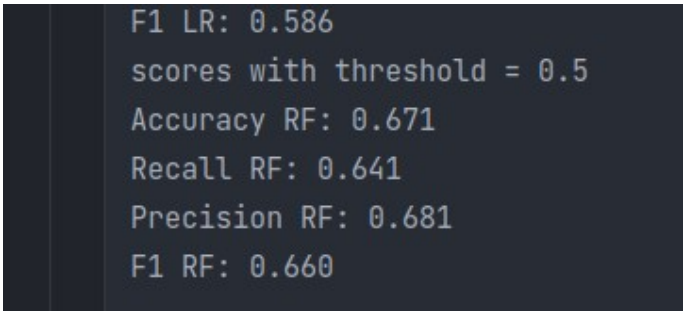

Рис. 24. Результат виконання для порогу 0.5

Як можна побачити з рис.23-24 при порозі 0.25: акуратність зменшилась, чутливість більша, точність стала меншою і оцінка змінилися трохи.

На графіку(рис. 25) продемонстровано що RF модель є більш зрозумілою, проти LR моделі. Але залежить також і від складності моделі. Тому це не завжди є основним показникомю.

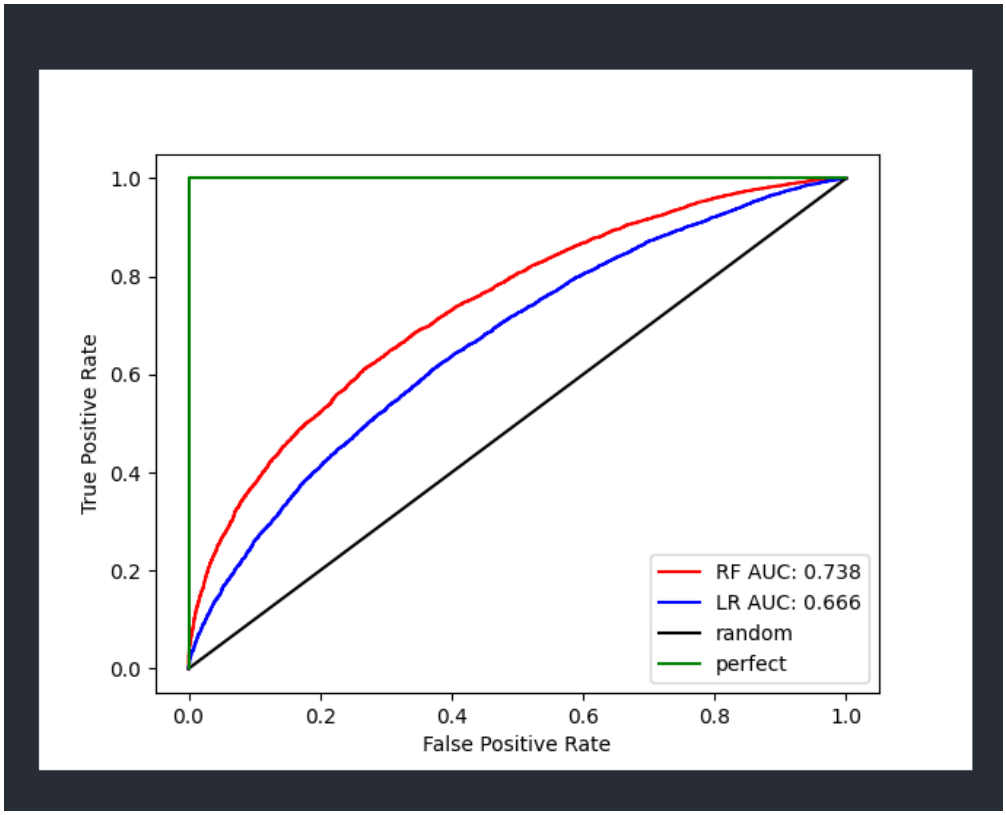

Рис. 25. Графік кривої ROC з урахуванням AUC

Завдання 2.6. Розробіть програму класифікації даних в файлі data_multivar_nb.txt за допомогою машини опорних векторів (Support Vector Machine - SVM). Розрахуйте показники якості класифікації. Порівняйте їх з показниками наївного байєсівського класифікатора. Зробіть висновки яку модель класифікації краще обрати і чому.

LR_1_task_6.py

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics

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.astype(int), test_size=0.2, random_state=3)
cls = svm.SVC(kernel='linear')
cls.fit(X_train, y_train)
pred = cls.predict(X_test)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred=pred))
print("Precision: ", metrics.precision_score(y_test, y_pred=pred, average='macro'))
print("Recall", metrics.recall_score(y_test, y_pred=pred, average='macro'))
print(metrics.classification_report(y_test, y_pred=pred))
visualize_classifier(cls, X_test, y_test)
```

Рис. 26. Відображення коду програми.

```
/home/xtr99/labs/ai/lab01/venv/bin/python /home/xtr99/labs/ai/lab01/LR_1_task_6.py
Accuracy: 1.0
Precision:  1.0
Recall 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00        17
           2       1.00      1.00      1.00        24
           3       1.00      1.00      1.00        19

    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80


Process finished with exit code 0
```

Рис. 27. Результати виконання програми.

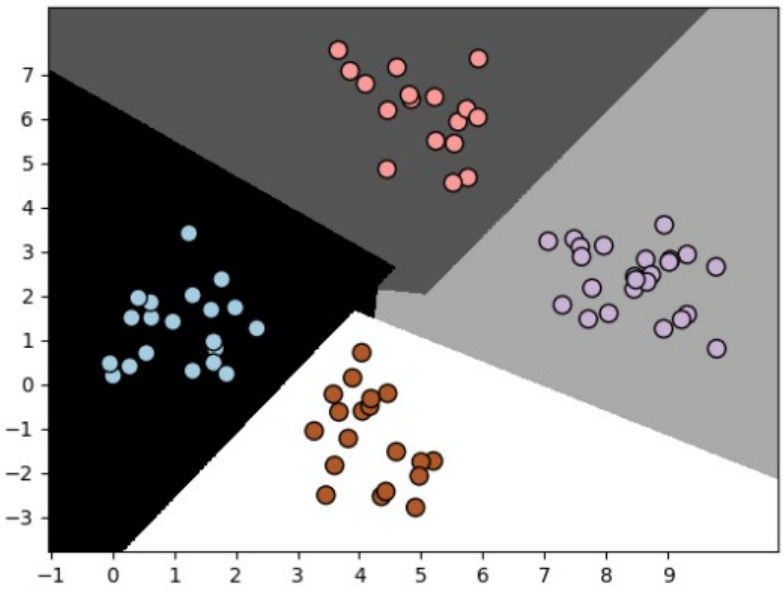

Рис. 28. Результати виконання програми.

Покликання на github: https://github.com/mrkulish/ai-labs/tree/master/lab01

Висновок: якість класифікації при використанні SVM становить 100%, тоді як при використанні наївного байєсівського класифікатора досягає 99.75%. Однак, враховуючи, що при наївному байєсівському класифікаторі значення порогу відіграють важливу роль і можуть впливати на результати, навіть дуже невелике відхилення від 100% може стати вирішальним. Тому, хоча різниця між значенням 99.75% та 100% є дуже незначною, краще використовувати модель SVM, оскільки вона працює без залежності від певних встановлених значень або порогів. Це робить її більш надійною та стабільною моделлю класифікації.