ЛАБОРАТОРНА РОБОТА № 1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

Завдання 2.1.1 – 2.1.4

```python
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))

# Исключение среднего
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 2.1 - Файл main.py

Результат:

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

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

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

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

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]]
PS C:\ztu\штучний інтелект>
```

Рис 2.2 – Результат виконання коду фалу main.py

Висновок: **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 4 значень. Взагалі, техніка *L1-нормалізації* вважається більш надійною по порівняно з *L2-нормалізацією*, оскільки вона менш чутлива до викидів

Завдання 2.1.5

```python
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))
```

Рис 2.3 Код файлу LR_1_task1.py

Результат:

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

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

Encoded values = [3, 0, 4, 1]
Decoded labels = ['white', 'black', 'yellow', 'green']
PS C:\ztu\штучний інтелект>
```

Рис 2.4 Результат файлу LR_1_task1.py

Завдання 2.2 Попередня обробка нових даних

18.	4.6	3.9	-3.5	-2.9	4.1	3.3	2.2	8.8	-6.1	3.9	1.4	2.2	2.2

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

input_data = np.array([[4.6, 3.9, -3.5], [-2.9, 4.1, 3.3], [2.2, 8.8, -6.1], [3.9, 1.4, 2.2]])
threshold_limit = 2.2

# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold = threshold_limit).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))

# Исключение среднего
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 2.5. Код файлу LR_1_task2.py

Результат:

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

BEFORE:
Mean = [ 1.95   4.55  -1.025]
Std deviation = [2.93300188 2.67441582 3.9047247 ]

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

Min max scaled data:
 [[1.         0.33783784 0.27659574]
 [0.         0.36486486 1.        ]
 [0.68       1.         0.        ]
 [0.90666667 0.         0.88297872]]

l1 normalized data:
 [[ 0.38333333  0.325      -0.29166667]
 [-0.2815534   0.39805825  0.32038835]
 [ 0.12865497  0.51461988 -0.35672515]
 [ 0.52        0.18666667  0.29333333]]

l2 normalized data:
 [[ 0.65970588  0.55931585 -0.50195013]
 [-0.4825966   0.68229174  0.54916164]
 [ 0.20125974  0.80503895 -0.55803836]
 [ 0.83129388  0.29841319  0.46893501]]
PS C:\ztu\штучний інтелект>
```

Рис 2.6 Результат файлу LR_1_task2.py

Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор

```python
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 2.7 Код файлу LR_1_task3.py

Результат:

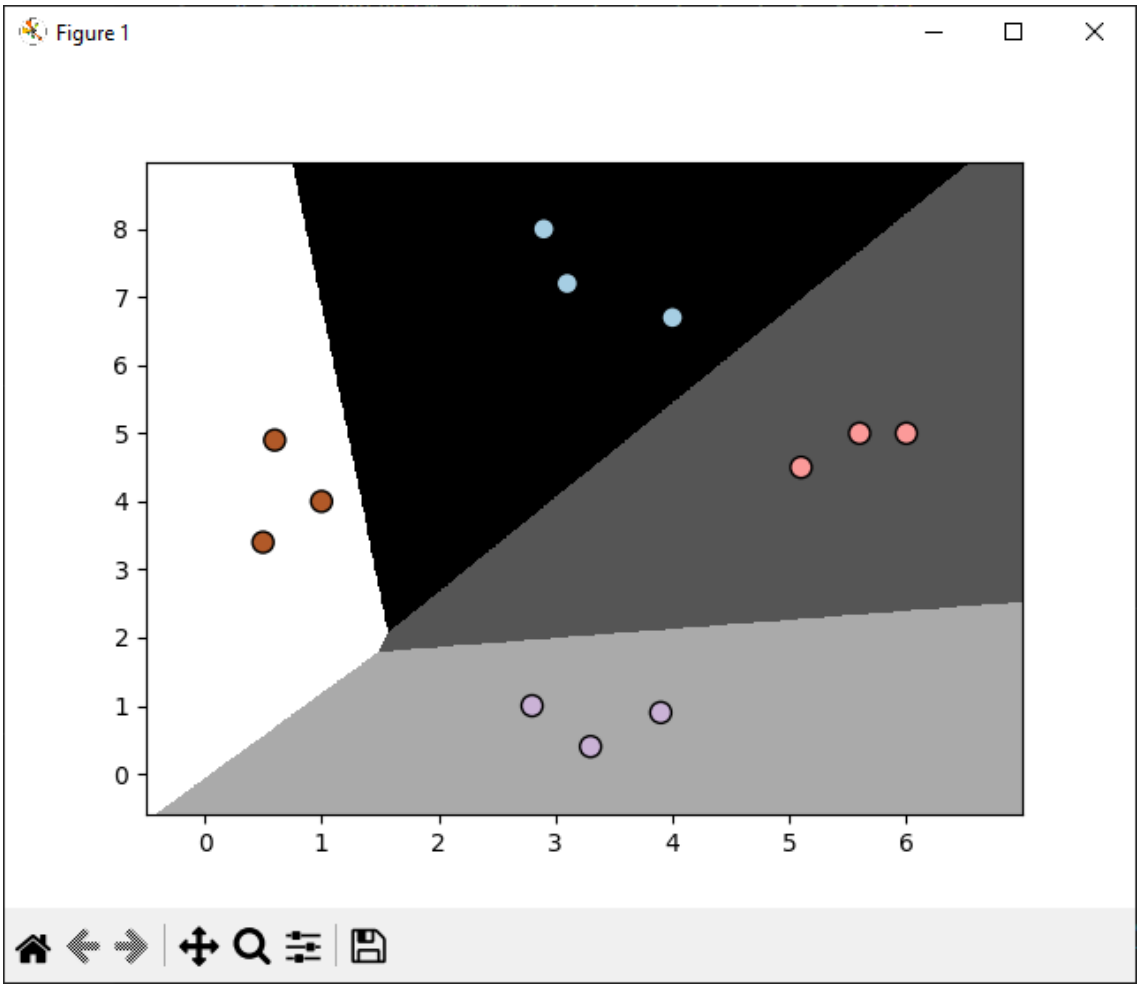

Рис 2.8 Результат файлу LR_1_task3.py

Завдання 2.4

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

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

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

# Створення наївного байєсовського класифікатора
classifier = GaussianNB()

# Тренування класифікатора
classifier.fit(X, y)

# 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)

# 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_new = GaussianNB()

classifier_new.fit(X_train, y_train)

y_test_pred = classifier_new.predict(X_test)

# 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)

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 2.9 Код файлу LR_1_task4.py

Результат:

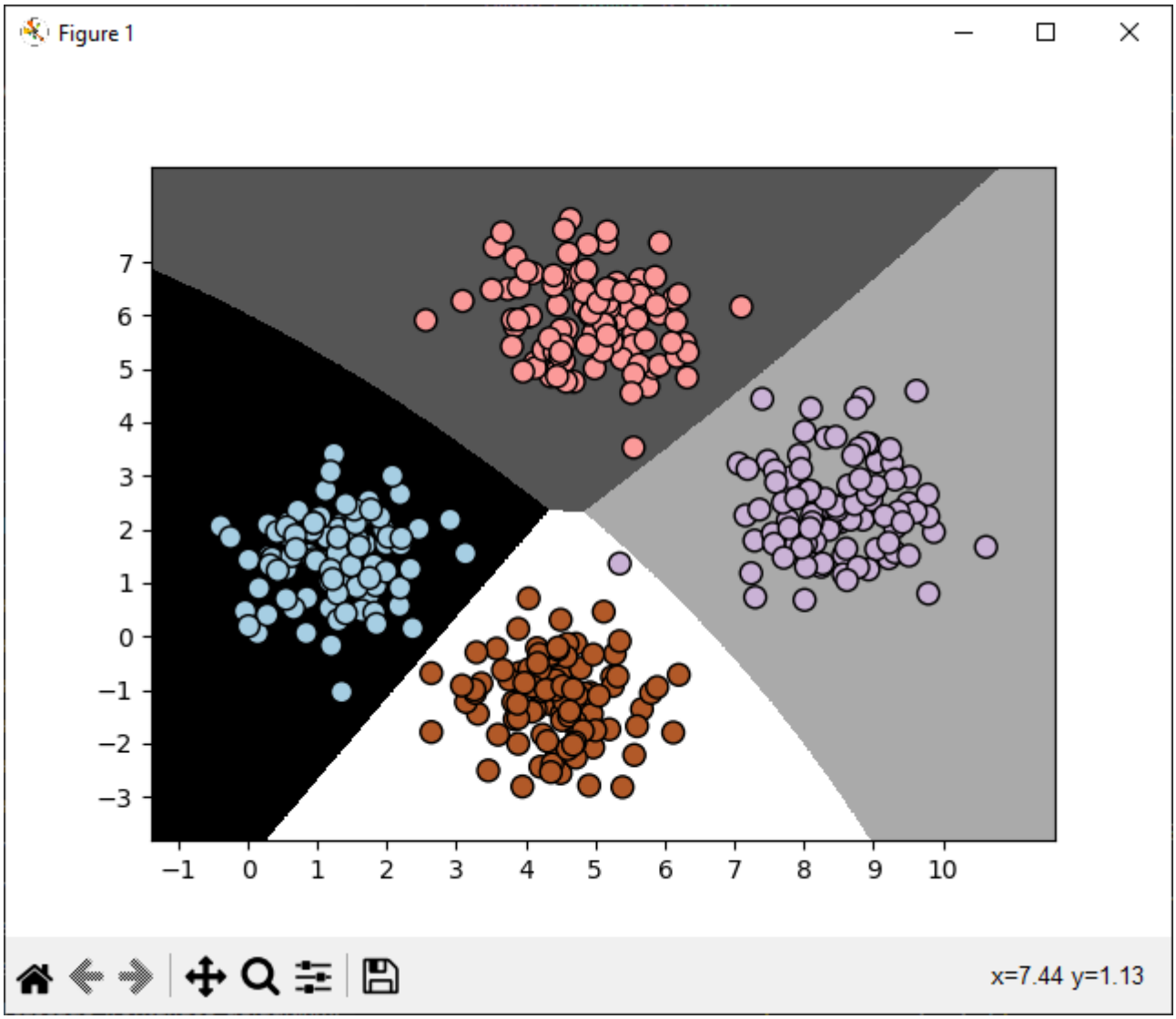

Рис 2.10 Результат файлу LR_1_task4.py

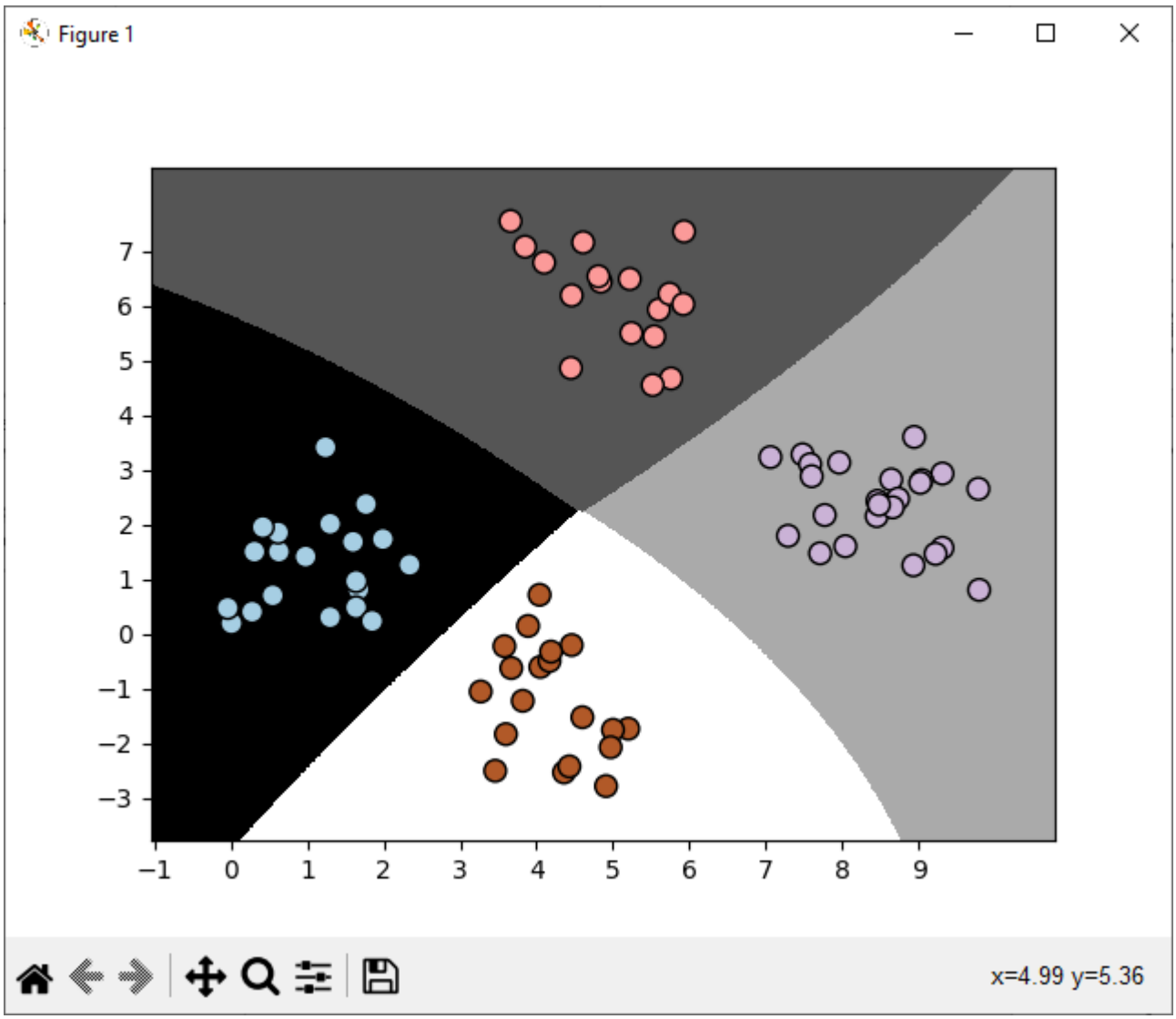

Рис 2.11 Результат файлу LR_1_task4.py

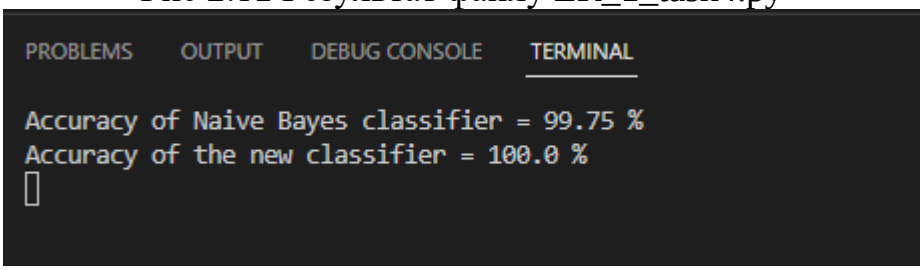

Рис 2.12 Результат файлу LR_1_task4.py

Завдання 2.5.Вивчити метрики якості класифікації

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

df = pd.read_csv('data_metrics.csv')
thresh = 0.5

df.head()

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

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


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

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

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

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

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

```python
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 sobolevskyi_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]])


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

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

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

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

print('Accuracy RF:%.3f' % (sobolevskyi_accuracy_score(df.actual_label.values, df.predicted_RF.values)))
print(recall_score(df.actual_label.values, df.predicted_RF.values))
```

```python
def sobolevskyi_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 sobolevskyi_recall_score(df.actual_label.values, df.predicted_RF.values) == recall_score(df.actual_label.values, df.predicted_RF.values), 'sobolevskyi_accuracy_score failed on RF'
assert sobolevskyi_recall_score(df.actual_label.values, df.predicted_LR.values) == recall_score(df.actual_label.values, df.predicted_LR.values), 'sobolevskyi_accuracy_score failed on LR'

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

def sobolevskyi_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 sobolevskyi_precision_score(df.actual_label.values, df.predicted_RF.values) == precision_score(df.actual_label.values, df.predicted_RF.values), 'my_accuracy_score failed on RF'
assert sobolevskyi_precision_score(df.actual_label.values, df.predicted_LR.values) == precision_score(df.actual_label.values, df.predicted_LR.values), 'my_accuracy_score failed on LR'

print('Precision RF: %.3f' % (sobolevskyi_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision LR: %.3f' % (sobolevskyi_precision_score(df.actual_label.values, df.predicted_LR.values)))
f1_score(df.actual_label.values, df.predicted_RF.values)

def sobolevskyi_f1_score(y_true, y_pred):  # calculates the F1 score
    recall = sobolevskyi_recall_score(y_true, y_pred)
    precision = sobolevskyi_precision_score(y_true, y_pred)
    return 2 * (precision * recall) / (precision + recall)

assert sobolevskyi_f1_score(df.actual_label.values, df.predicted_RF.values) == f1_score(df.actual_label.values, df.predicted_RF.values), 'my_accuracy_score failed on RF'
assert sobolevskyi_f1_score(df.actual_label.values, df.predicted_LR.values) == f1_score(df.actual_label.values, df.predicted_LR.values), 'my_accuracy_score failed on LR'

```

```python
print('F1 RF: %.3f' % (sobolevskyi_f1_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 LR: %.3f' % (sobolevskyi_f1_score(df.actual_label.values, df.predicted_LR.values)))
print('scores with threshold = 0.5')
print('Accuracy RF: % .3f' % (sobolevskyi_accuracy_score(df.actual_label.values, df.predicted_RF.values)))
print('Recall RF: %.3f' % (sobolevskyi_recall_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision RF: % .3f' % (sobolevskyi_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 RF: %.3f' % (sobolevskyi_f1_score(df.actual_label.values, df.predicted_RF.values)))
print('')

threshold = 0.75

print(f'Scores with threshold = {threshold}')
print('Accuracy RF: % .3f' % (sobolevskyi_accuracy_score(df.actual_label.values, (df.model_RF >= threshold).astype('int').values)))
print('Recall RF: %.3f' % (sobolevskyi_recall_score(df.actual_label.values, (df.model_RF >= threshold).astype('int').values)))
print('Precision RF: %.3f' % (sobolevskyi_precision_score(df.actual_label.values, (df.model_RF >= threshold).astype('int').values)))
print('F1 RF: %.3f' % (sobolevskyi_f1_score(df.actual_label.values, (df.model_RF >= threshold).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)

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()

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. 2.13 Код файлу LR_1_task5.py

Результат:

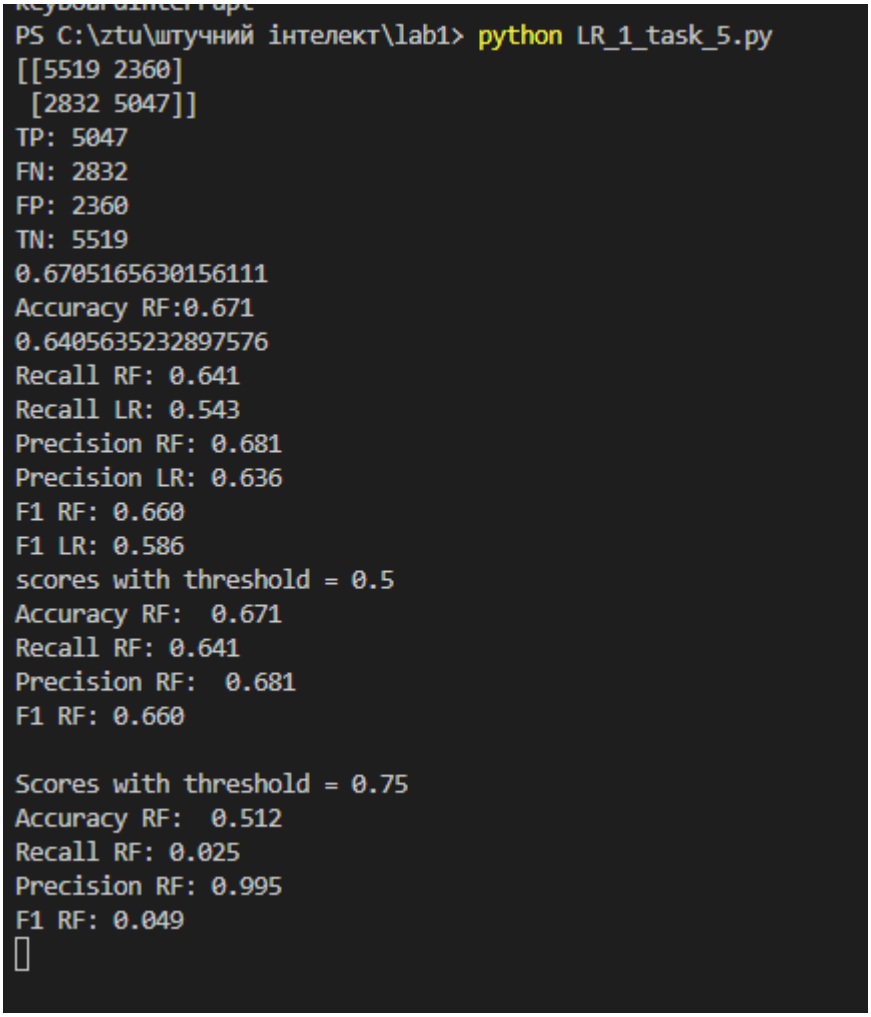

Рис 2.14 Результат файлу LR_1_task5.py

F1 міра зменшується в результаті збільшення порогу.

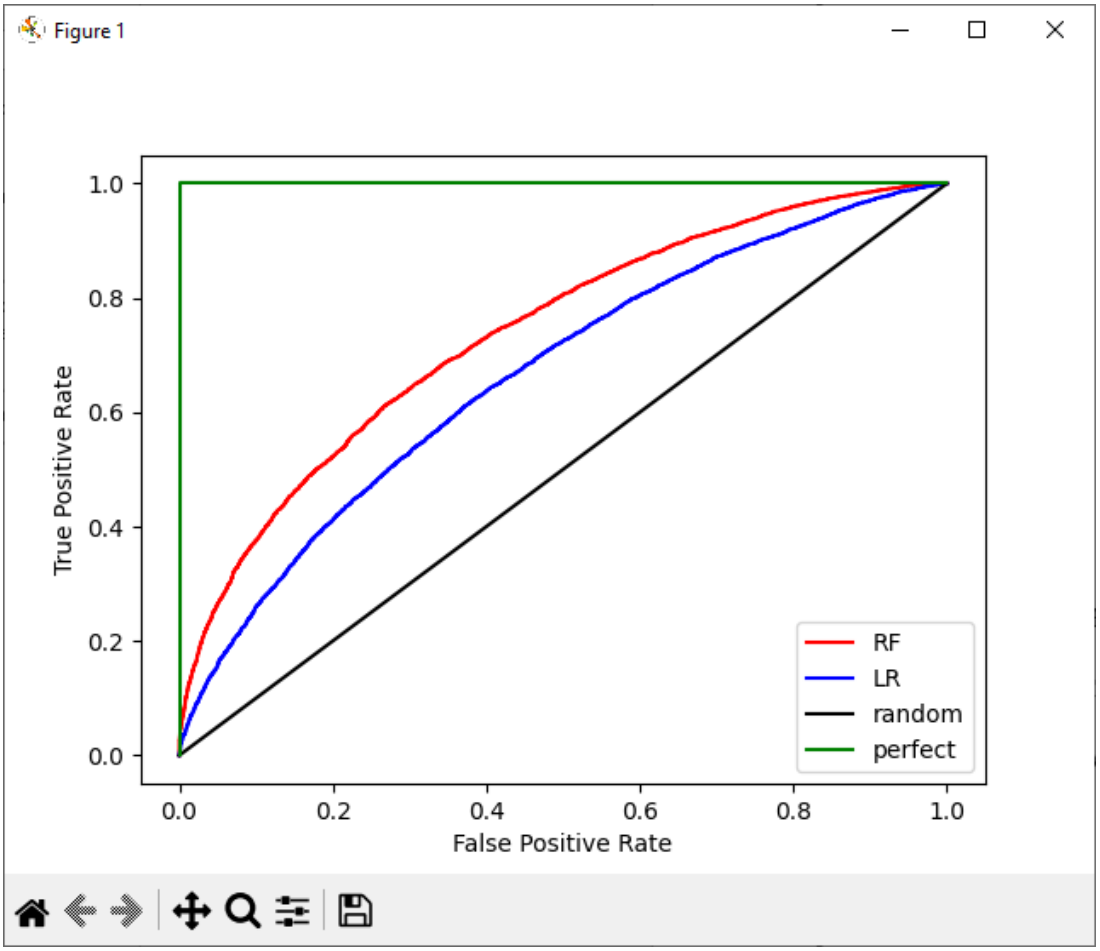

Рис 2.15. ROC – крива.

Подивившись на модель, бачимо що RF модель має більшу зрозумілість, аніж LR модель. Але залежить також і від складності моделі. Тому це не завжди є основним показником.

Завдання 2.6

```python
import numpy as np
from sklearn import datasets
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. 2.16 Код файлу LR_1_task6.py

```
PS C:\ztu\штучний інтелект\lab1> python 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
```

Рис 2.17 Результат файлу LR_1_task6.py

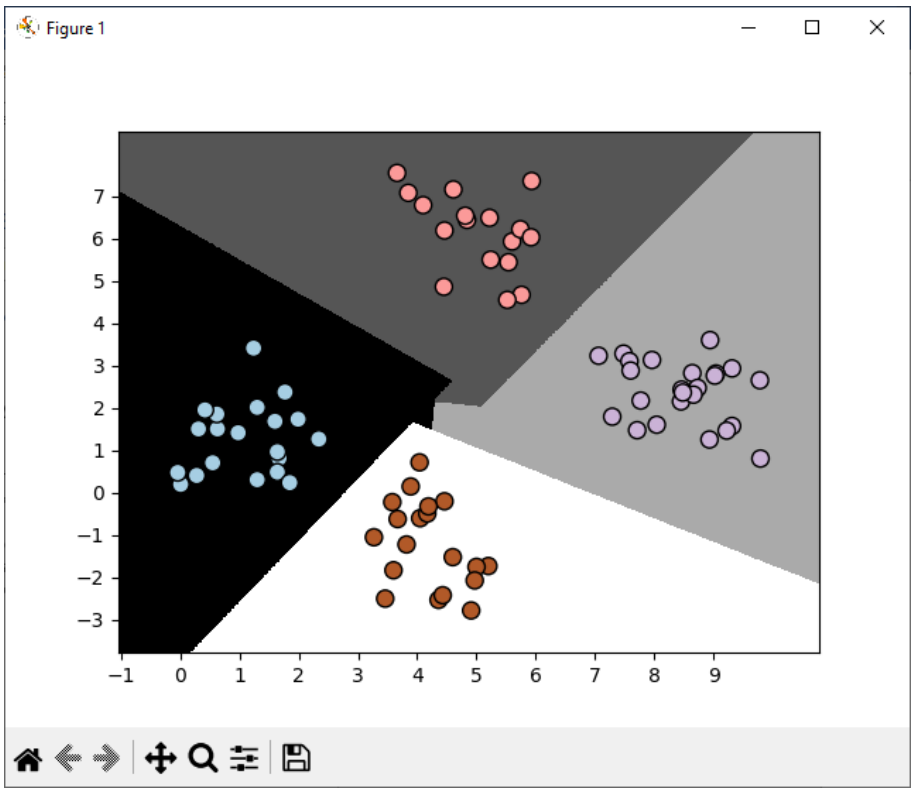

Рис 2.18 Результат файлу LR_1_task6.py

Висновок: після виконання лабораторної роботи навчився використовувати спеціалізовані бібліотеки та мову програмування Python, дослідитв попередню обробку та класифікацію даних.