ЛАБОРАТОРНА РОБОТА № 1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

Хід роботи

Завдання 2.1.1

Код 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))

# 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. 1.1 Код програми

					Державний університет "Житомирська політехніка"	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Дроботун Д. Я.			Звіт з практичної роботи	Літ. / Арк. 1 / Аркушів
Перевір.						
Керівник						
Н. контр.						Гр. ІПЗК-19-1
Зав. каф.						

Результат виконання

```
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]]
```

Рис. 1.2 Результат виконання програми

Зробіть висновок чим відрізняються L1-нормалізація від L2-нормалізації.

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.

Завдання 2.1.2

Код програми

```
import numpy as np
from sklearn import preprocessing

# Надання позначок вхідних даних
Input_labels = ['red', 'black', 'red', 'green', 'yellow', 'white']

# між мітками та числами
# Створення кодувальника та встановлення відповідності
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. 1.3 Код програми

Результат виконання

```
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. 1.4 Результат виконання програми

Завдання 2.2. Попередня обробка нових даних

По списку журнала 5 варіант;

5.	-1.3	3.9	4.5	-5.3	-4.2	-1.3	5.2	-6.5	-1.1	-5.2	2.6	-2.2	3.0

Код програми

```
import numpy as np
from sklearn import preprocessing

input_data = np.array([[5.1, -2.9, 3.3],
 [-1.3,  3.9, 4.5],
 [-5.3, -4.2, -1.3],
 [5.2, -6.5, -1.1],
 [-5.2, 2.6, -2.2]])


# Бінаризація даних
data_binarized = preprocessing.Binarizer(threshold=3.0).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. 1.5 Код програми

Результат виконання

```
BEFORE:
Mean = [-0.3  -1.42  0.64]
Std deviation = [4.67803377 4.00469724 2.71411127]

AFTER:
Mean = [-8.8817842e-17 -4.4408921e-17  0.0000000e+00]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.99047619 0.34615385 0.82089552]
 [0.38095238 1.         1.        ]
 [0.         0.22115385 0.13432836]
 [1.         0.         0.1641791 ]
 [0.00952381 0.875      0.        ]]

l1 normalized data:
 [[ 0.45132743 -0.25663717  0.2920354 ]
 [-0.13402062  0.40206186  0.46391753]
 [-0.49074074 -0.38888889 -0.12037037]
 [ 0.40625    -0.5078125  -0.0859375 ]
 [-0.52        0.26       -0.22      ]]

l2 normalized data:
 [[ 0.75765788 -0.43082507  0.49024922]
 [-0.21328678  0.63986035  0.7383004 ]
 [-0.76965323 -0.60991388 -0.18878287]
 [ 0.61931099 -0.77413873 -0.13100809]
 [-0.83653629  0.41826814 -0.3539192 ]]
```

Рис. 1.6 Результат виконання програми

Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор.

Код програми

```
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. 1.7 Код програми

Результат виконання

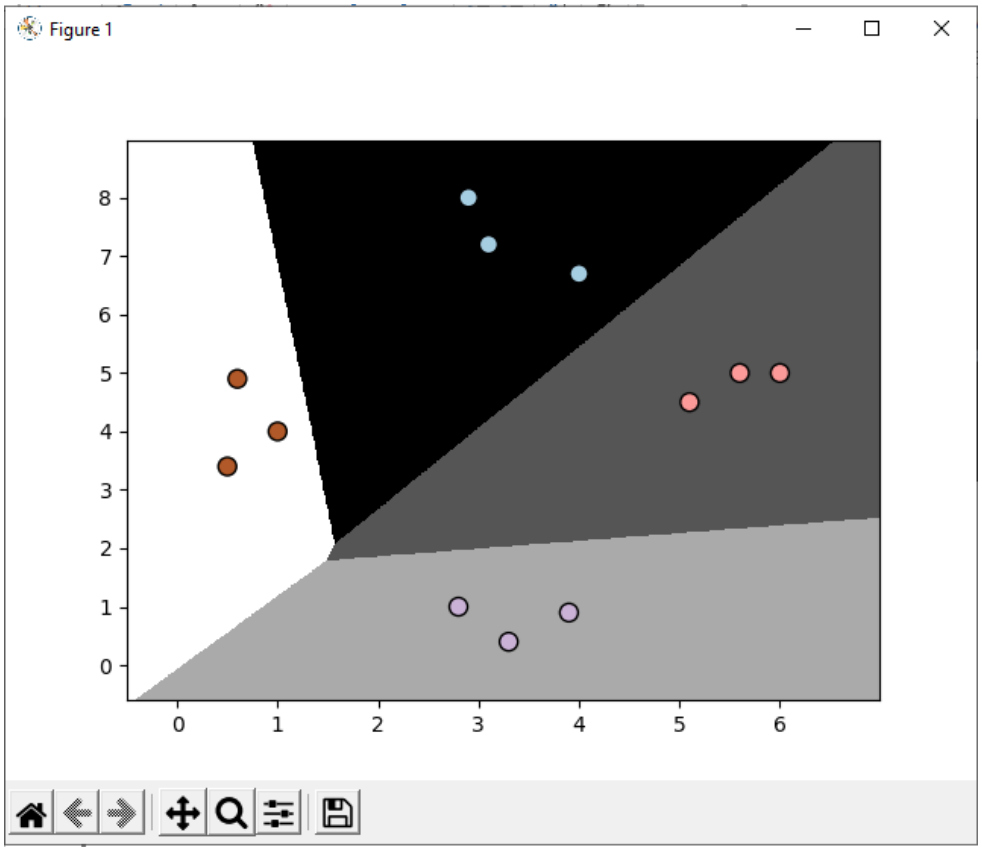

Рис. 1.8 Результат виконання програми

Завдання 2.4. Класифікація наївним байєсовським класифікатором

Код програми

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

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
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. 1.9 Код програми

Результат виконання

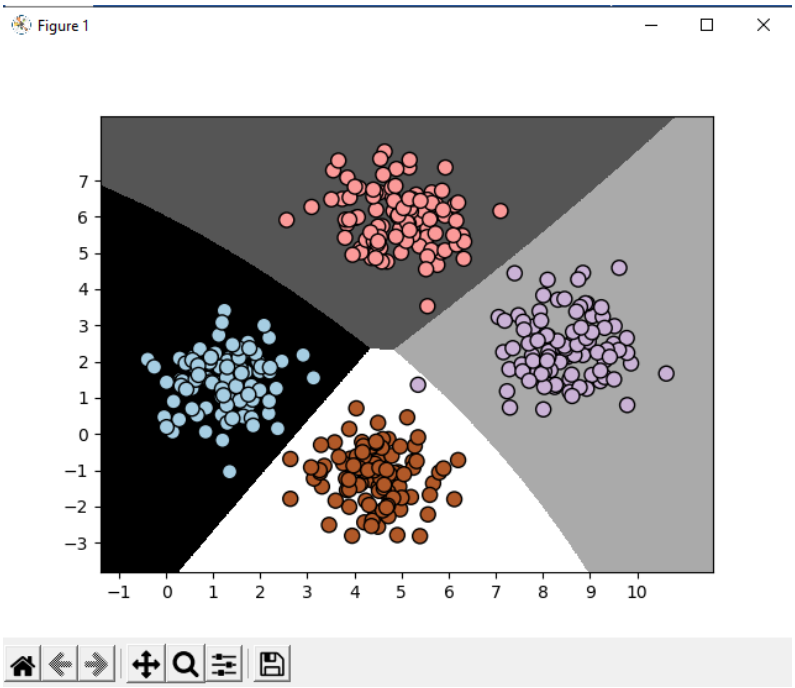

Рис. 2.1 Результат виконання програми

Завдання 2.4.1

Код програми

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

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
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.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 = train_test_split.cross_val_score(classifier,
 X, y, scoring='accuracy', cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2))
+ "%")
precision_values = train_test_split.cross_val_score(classifier,
 X, y, scoring='precision_weighted', cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(),
2)) + "%")
recall_values = train_test_split.cross_val_score(classifier,
 X, y, scoring='recall_weighted', cv=num_folds)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) +
"%")
f1_values = train_test_split.cross_val_score(classifier, X, y,
scoring='f1_weighted', cv=num_folds)
print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")
```

Рис. 2.2 Код програми

Результат виконання

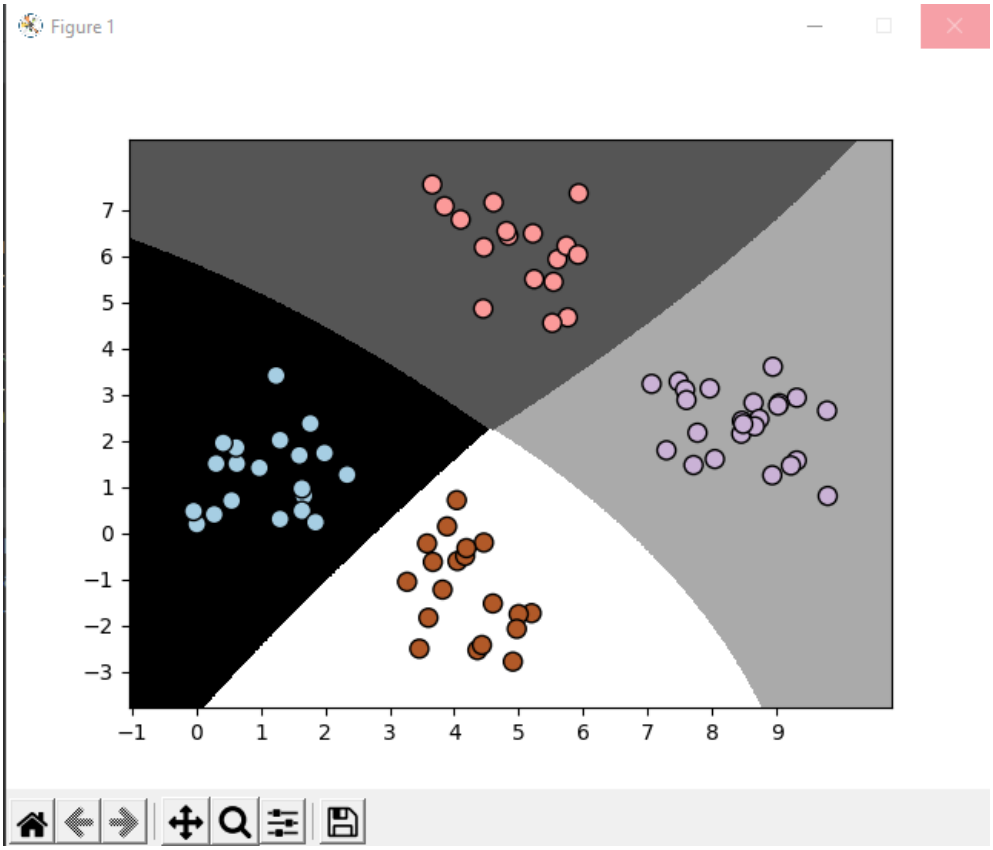

Рис. 2.3 Результат виконання програми

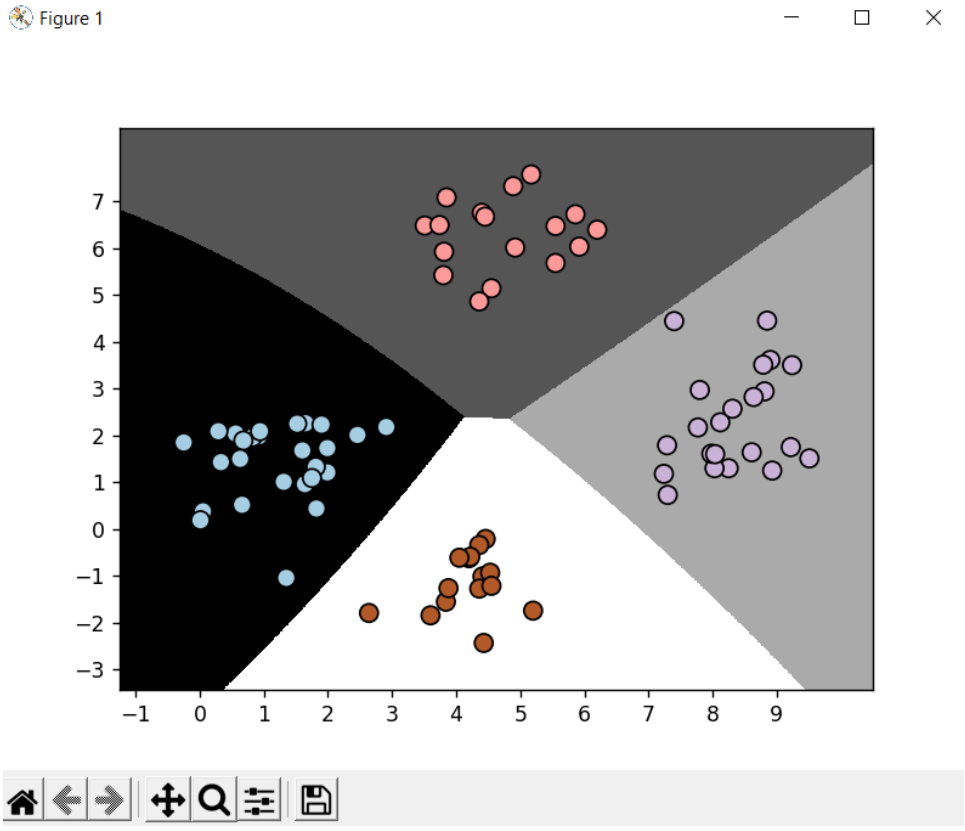

Рис. 2.4 Результат виконання програми після другого прогону

Висновок: після другого прогону результати відрізняються, тому що, були обрані інші вхідні дані.

Завдання 2.5. Вивчити метрики якості класифікаці

Код програми

```
import pandas as pd
from sklearn.metrics import confusion_matrix
import numpy as np
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
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score

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

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

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))


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 dmytro_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]])


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

assert np.array_equal(dmytro_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(dmytro_confusion_matrix(df.actual_label.values,
                                              df.predicted_LR.values),
confusion_matrix(df.actual_label.values,

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

accuracy score(df.actual label.values, df.predicted RF.values)


def my_accuracy_score(y_true, y_pred):
    # calculates the fraction of samples
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return (TP + TN) / (TP + TN + FP + FN)


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

df.predicted_RF.values), 'my_accuracy_score failed on'

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

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

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

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


def my_recall_score(y_true, y_pred):
    # calculates the fraction of positive samples predicted correctly
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FN)


assert my_recall_score(df.actual_label.values, df.predicted_RF.values) ==
recall_score(df.actual_label.values, df.predicted_RF.values), 'my_accuracy_score
failed on RF'

assert my_recall_score(df.actual_label.values, df.predicted_LR.values) ==
recall_score(df.actual_label.values, df.predicted_LR.values), 'my_accuracy_score
failed on LR'
```

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

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


def my_precision_score(y_true, y_pred):
    # calculates the fraction of predicted positives samples that are actually
positive
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return TP / (TP + FP)


assert my_precision_score(df.actual_label.values, df.predicted_RF.values) == \
       precision_score(df.actual_label.values, df.predicted_RF.values),
'my_accuracy_score failed on RF'
assert my_precision_score(df.actual_label.values, df.predicted_LR.values) == \
       precision_score(df.actual_label.values, df.predicted_LR.values),
'my_accuracy_score failed on LR'

print('Precision RF: %.3f' % (my_precision_score(df.actual_label.values,
df.predicted RF.values)))

print('Precision LR: %.3f' % (my_precision_score(df.actual_label.values,
df.predicted_LR.values)))

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


def my_f1_score(y_true, y_pred):
    # calculates the F1 score
    recall = my_recall_score(y_true, y_pred)
    precision = my_precision_score(y_true, y_pred)
    return (2 * precision * recall) / (precision + recall)


assert my_f1_score(df.actual_label.values, df.predicted_RF.values) == \
       f1_score(df.actual_label.values, df.predicted_RF.values),
'my_accuracy_score failed on RF'

assert my_f1_score(df.actual_label.values, df.predicted_LR.values) == \
       f1_score(df.actual_label.values, df.predicted_LR.values),
'my_accuracy_score failed on LR'
print('F1 RF: %.3f' % (my_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 LR: %.3f' % (my_f1_score(df.actual_label.values,
df.predicted_LR.values)))

print('scores with threshold = 0.5')
print('Accuracy RF:%.3f' % (my_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall RF: %.3f' % (my_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF:%.3f' % (my_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 RF: %.3f' % (my_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF:%.3f' % (my_accuracy_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
```

```
print('Recall RF: %.3f' % (my_recall_score(df.actual_label.values, (df.model_RF
>= 0.25).astype('int').values)))
print('Precision RF:%.3f' % (my_precision_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('F1 RF: %.3f' % (my_f1_score(df.actual_label.values, (df.model_RF >=
0.25).astype('int').values)))

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. 2.5 Код програми

Результат виконання

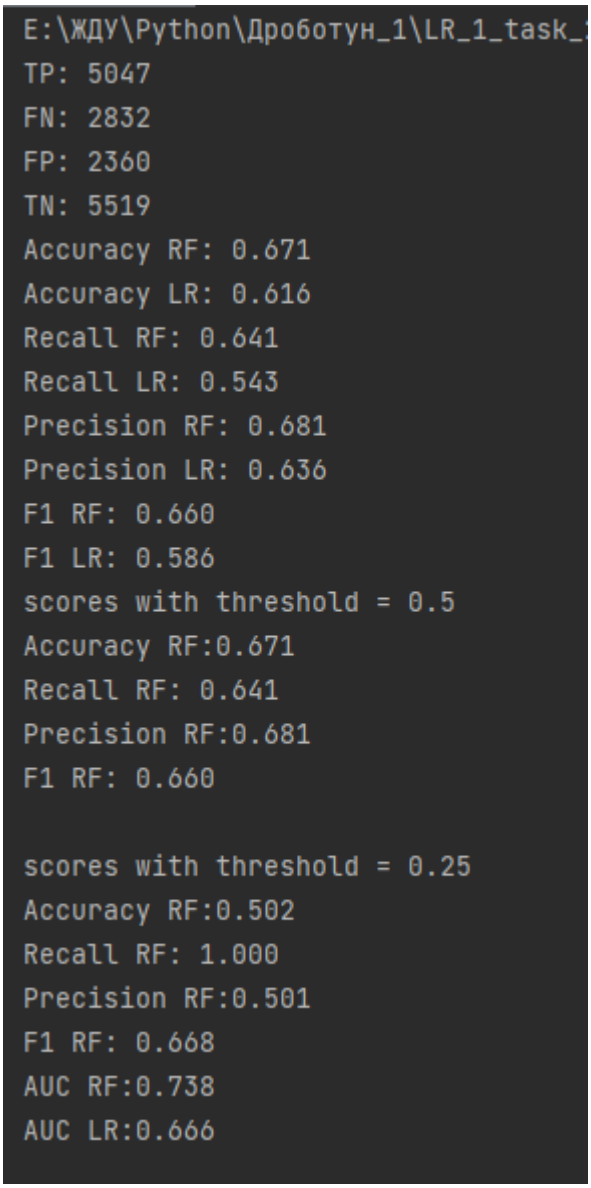

Рис. 2.6 Результат виконання програми

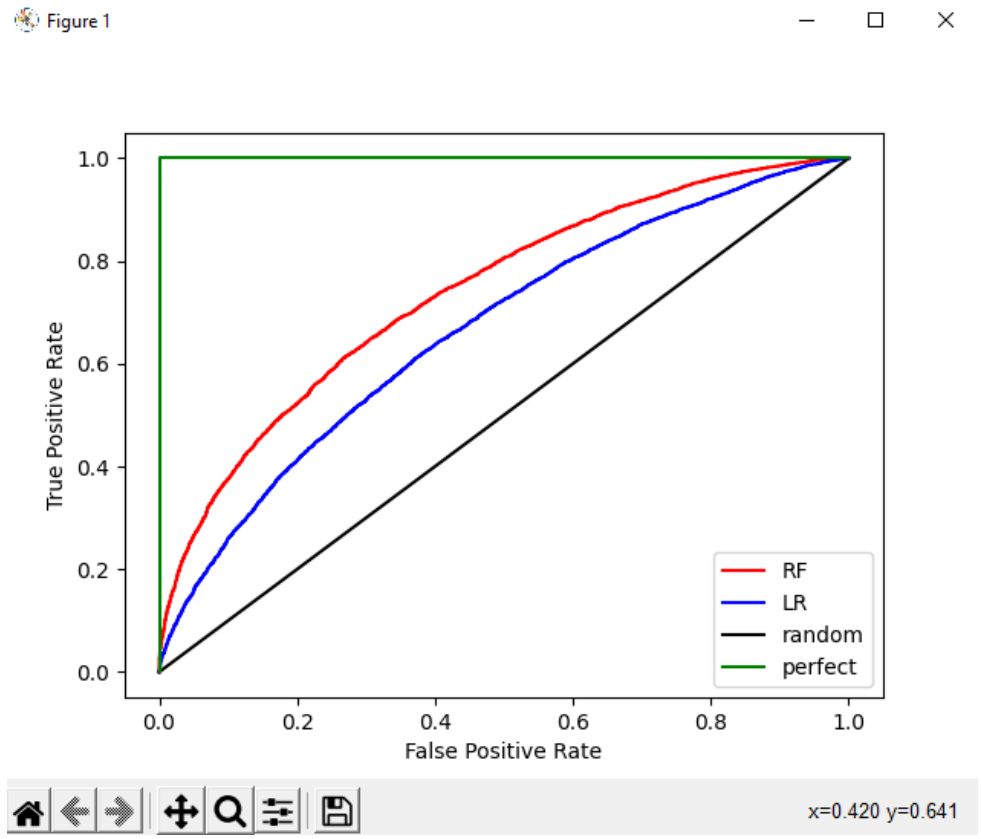

Рис. 2.7 Результат виконання програми

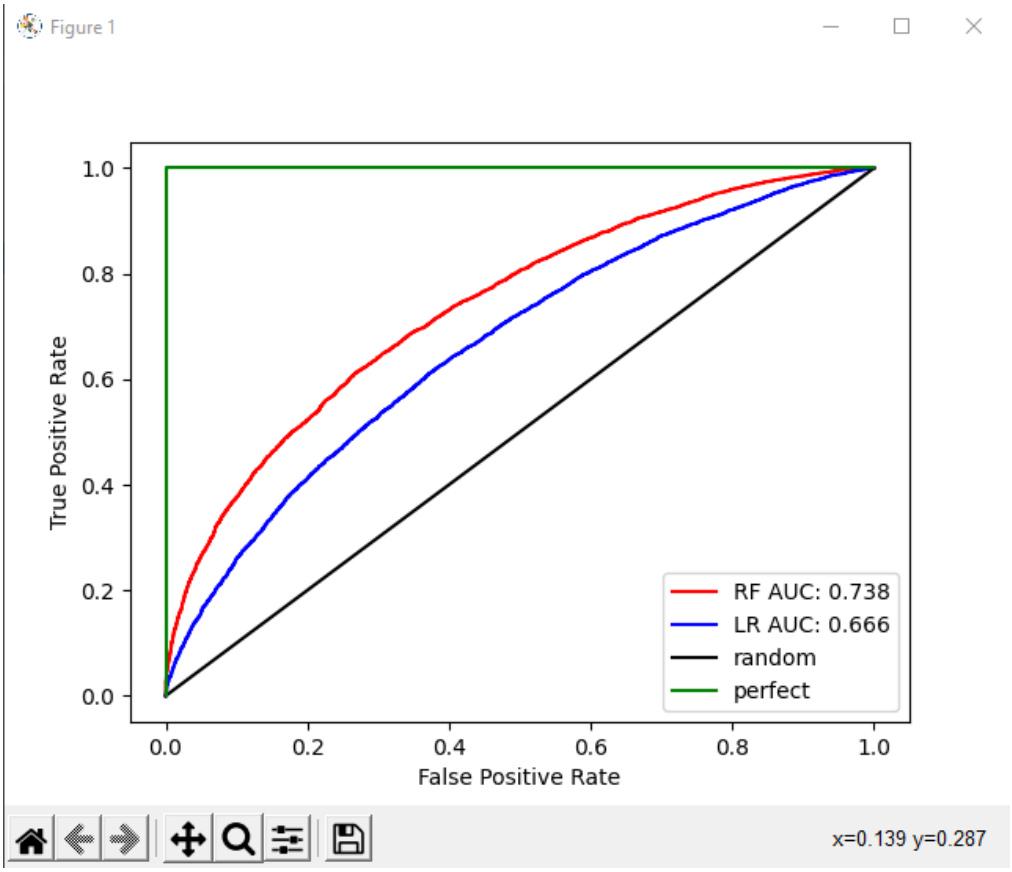

Рис. 2.8 Результат виконання програми

Висновок: я використав спеціалізовані бібліотеки мов програмування Python дослідивши попередню обробку та класифікацію даних.

Випадковий ліс (Random forest, RF) – це алгоритм навчання з учителем. Його можна використовувати як класифікації, так регресії. Також це найбільш гнучкий та простий у використанні алгоритм. Ліс складається із дерев. Кажуть, що чим більше дерев у лісі, тим він міцніший. RF створює дерева рішень для випадково вибраних семплов даних, отримує прогноз від кожного дерева та вибирає найкраще рішення у вигляді голосування. Він також надає досить ефективний критерій важливості показників (ознаки).

LR додатково використовуватиме функцію Sigmoid, щоб зробити нелінійне відображення з використанням сигмоїдної функції.

Позитивні та негативні точки вибірки з обох сторін класифікаційної суперплощини перетворюються на дві категорії, що розкладаються на 0,5: Категорія 0 та категорію 1 за допомогою функції стиснення.