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 1.

```python
from sklearn import preprocessing

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

encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)


print("\nLabel mapping: ")
for i, item in enumerate(encoder.classes_):
    print(item, '-->', i)

test_labels = ['green', 'red', 'black']
encoded_values = encoder.transform(test_labels)
print(f"\nLabels = {test_labels}")
print(f"Encoded values = {list(encoded_values)}")

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:

```
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 - Результат виконання

Завдання 2

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

```
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]])
data_binarized = preprocessing.Binarizer(threshold=2.2).transform(input_data)
print(f"\nBinarized data:\n{data_binarized}")

print("\nBEFORE: ")
print(f"Mean = {input_data.mean(axis=0)}")
print(f"Std deviation = {input_data.std(axis=0)}")

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)}")

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}")
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:

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

BEFORE:
Mean = [ 4.6  9.9 -3.5 -2.9  4.1  3.3 -2.2  8.8 -6.1  3.9  1.4  2.2]
Std deviation = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

AFTER:
Mean = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Std deviation = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

Min max scaled data:
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

L1 normalized data:
[[ 0.08695652  0.18714556 -0.06616257 -0.05482042  0.07750473  0.06238185
  -0.0415879   0.16635161 -0.11531191  0.07372401  0.02646503  0.0415879 ]]

L2 normalized data:
[[ 0.26167215  0.56316399 -0.19909838 -0.16496723  0.23322953  0.18772133
  -0.12514755  0.50059021 -0.34700003  0.22185248  0.07963935  0.12514755]]
```

Рис.2 - Результат виконання

Завдання 3.

```
import numpy as np
from sklearn import linear_model
from utilities import visualize_classifier

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])

classifier = linear_model.LogisticRegression(solver='liblinear', C=1)

classifier.fit(X, y)

visualize_classifier(classifier, X, y)
```

Результат виконання:

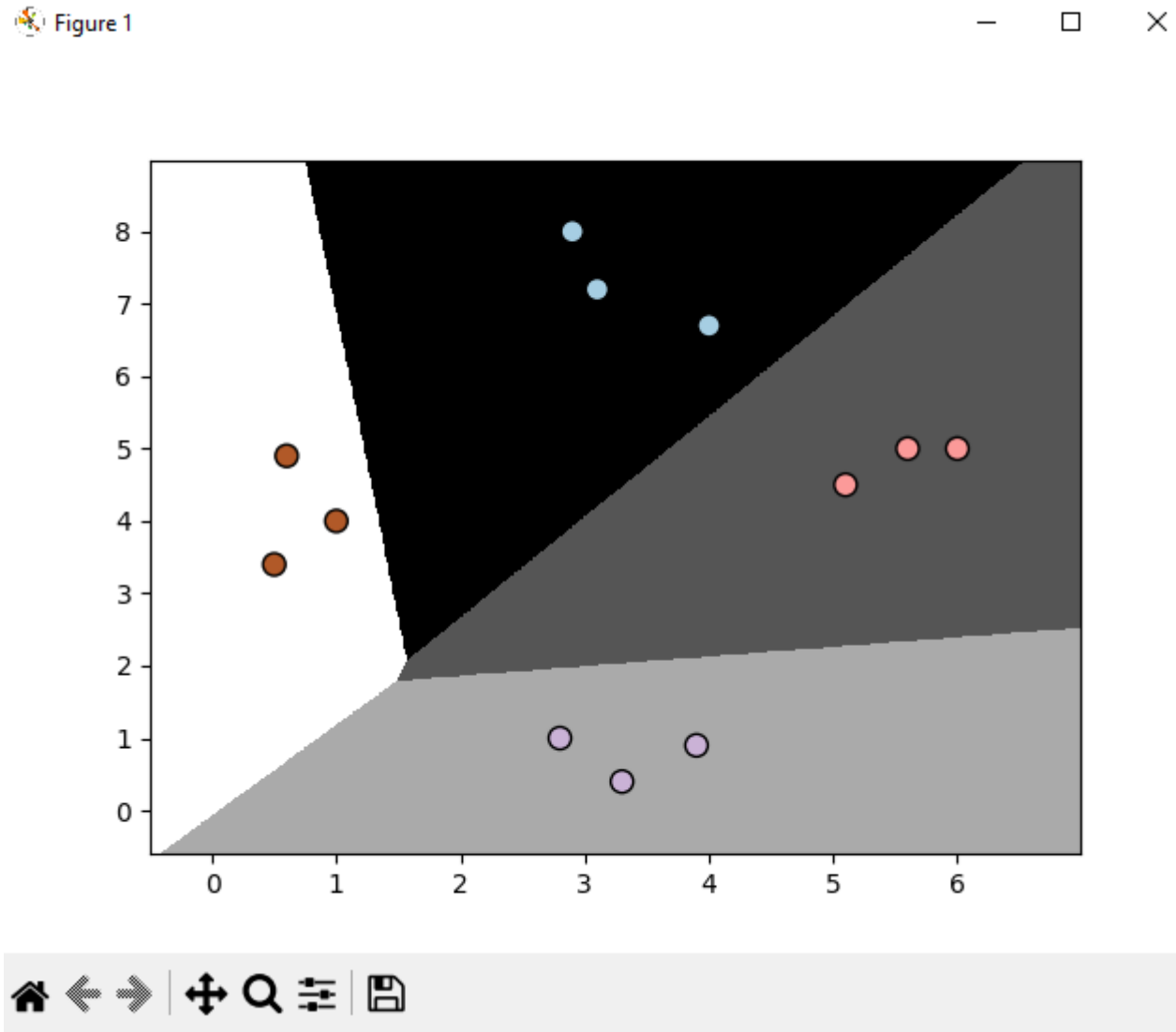

Рис.3 - Результат виконання

Завдання 4.

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

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(f"Accuracy of Naive Bayes classifier = {round(accuracy,2)}%")


visualize_classifier(classifier, X, y)
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)


accuracy = 100.0 * (y_test == y_test_pred).sum()/X_test.shape[0]
print(f"Accuracy of the new classifier = {round(accuracy, 2)}%")


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='precision_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:

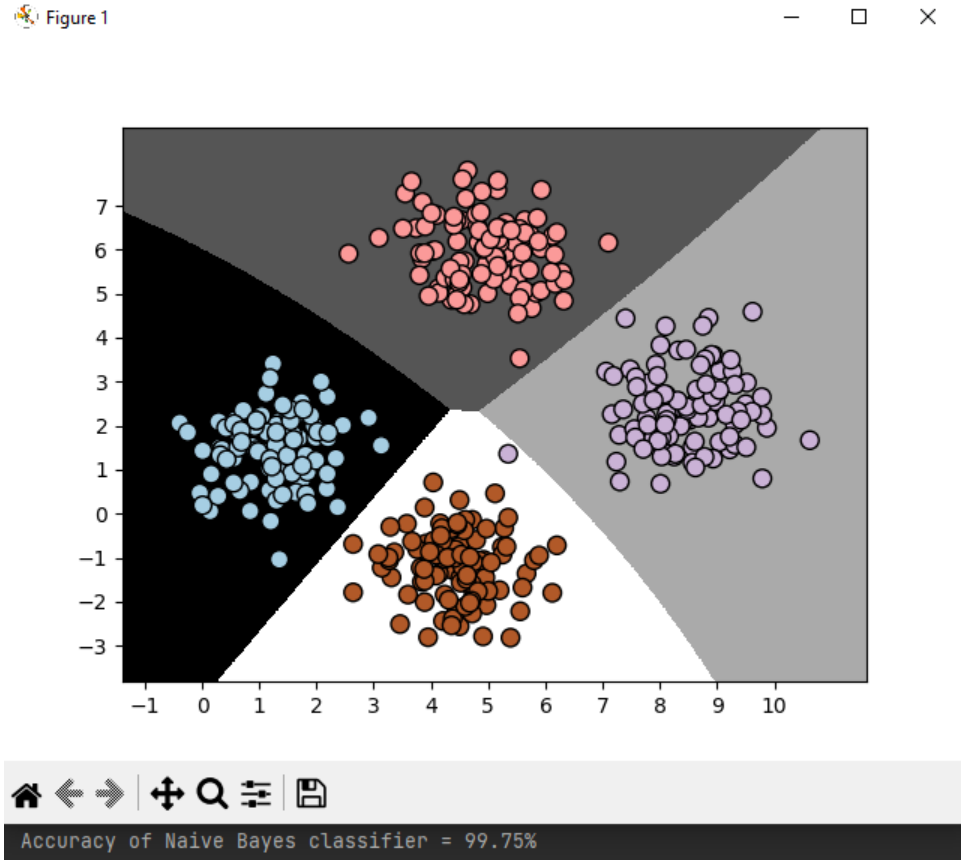

Рис.4 - Результат виконання

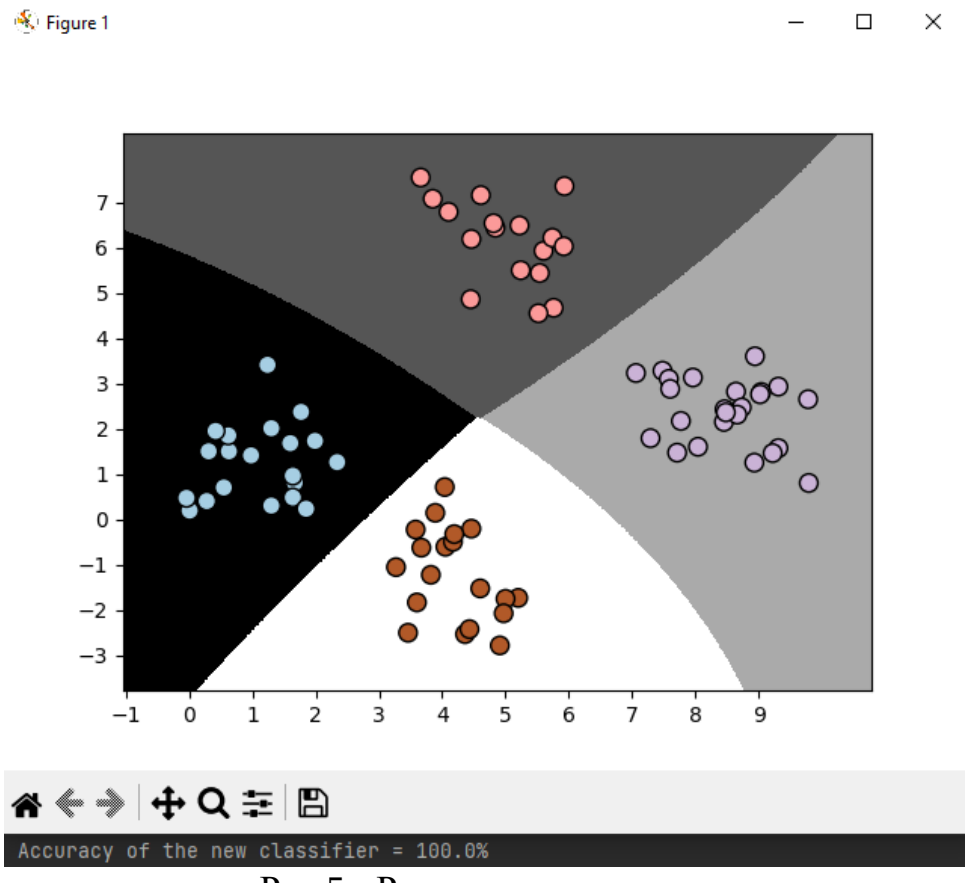

Рис.5 - Результат виконання

Завдання 5.

```
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')
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()

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))


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 voitko_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]])
```

```
voitko_confusion_matrix(df.actual_label.values, df.predicted_RF.values)

assert np.array_equal(voitko_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(voitko_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 koptiaiev_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 koptiaiev_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 koptiaiev_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' % (koptiaiev_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))

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


def koptiaiev_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 koptiaiev_recall_score(df.actual_label.values, df.predicted_RF.values) ==
recall_score(df.actual_label.values,

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

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

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

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


def koptiaiev_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 koptiaiev_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 koptiaiev_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' % (koptiaiev_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision LR: %.3f' % (koptiaiev_precision_score(df.actual_label.values,
df.predicted_LR.values)))

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


def koptiaiev_f1_score(y_true, y_pred):  # calculates the F1 score
    recall = koptiaiev_recall_score(y_true, y_pred)
    precision = koptiaiev_precision_score(y_true, y_pred)
    return 2 * (precision * recall) / (precision + recall)


assert koptiaiev_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 koptiaiev_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' % (koptiaiev_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 LR: %.3f' % (koptiaiev_f1_score(df.actual_label.values,
df.predicted_LR.values)))
print('scores with threshold = 0.5')

print('Accuracy RF: % .3f' % (koptiaiev_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall RF: %.3f' % (koptiaiev_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF: % .3f' % (koptiaiev_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 RF: %.3f' % (koptiaiev_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('')

threshold = 0.75

print(f'Scores with threshold = {threshold}')
print('Accuracy RF: % .3f' % (koptiaiev_accuracy_score(df.actual_label.values,
(df.model_RF >= threshold).astype('int').values)))
print('Recall RF: %.3f' % (koptiaiev_recall_score(df.actual_label.values,
(df.model_RF >= threshold).astype('int').values)))
print('Precision RF: %.3f' % (koptiaiev_precision_score(df.actual_label.values,
(df.model_RF >= threshold).astype('int').values)))
print('F1 RF: %.3f' % (koptiaiev_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()
```

Результат виконання:

```
[[5519 2360]
 [2832 5047]]
TP: 5047
FN: 2832
FP: 2360
TN: 5519
0.6705165630156111
Accuracy RF:0.671
0.6405635232897576
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

Scores with threshold = 0.75
Accuracy RF:  0.512
Recall RF: 0.025
Precision RF: 0.995
F1 RF: 0.049
AUC RF:0.738
AUC LR:0.666
```

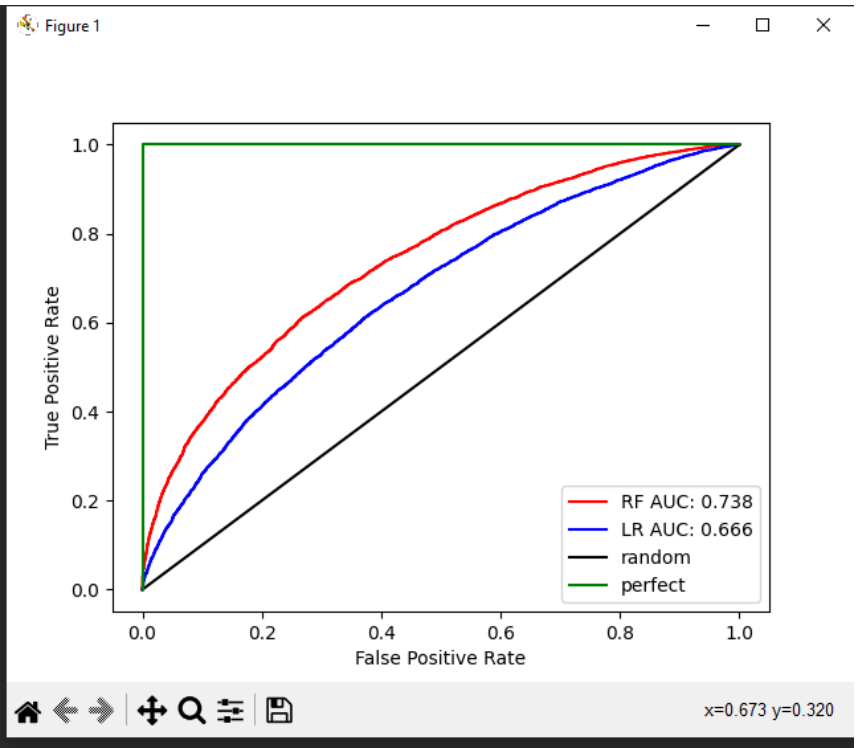

Рис. 6 - Результат виконання

Завдання 6.

```
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)
```

Результат виконання:

```
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
```

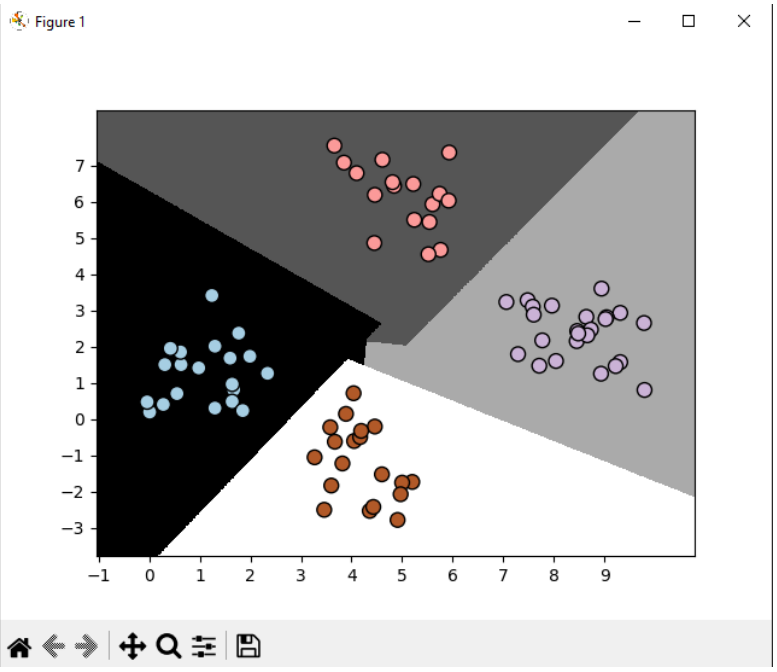

Рис. 7 - Результат виконання

Висновок: в ході виконання лабораторної роботи ми навчилися використовувати спеціалізовані бібліотеки та мову програмування Python для дослідження попередньої обробки та класифікації даних.