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:

Для написання коду використовується середовище програмування PyCharm.

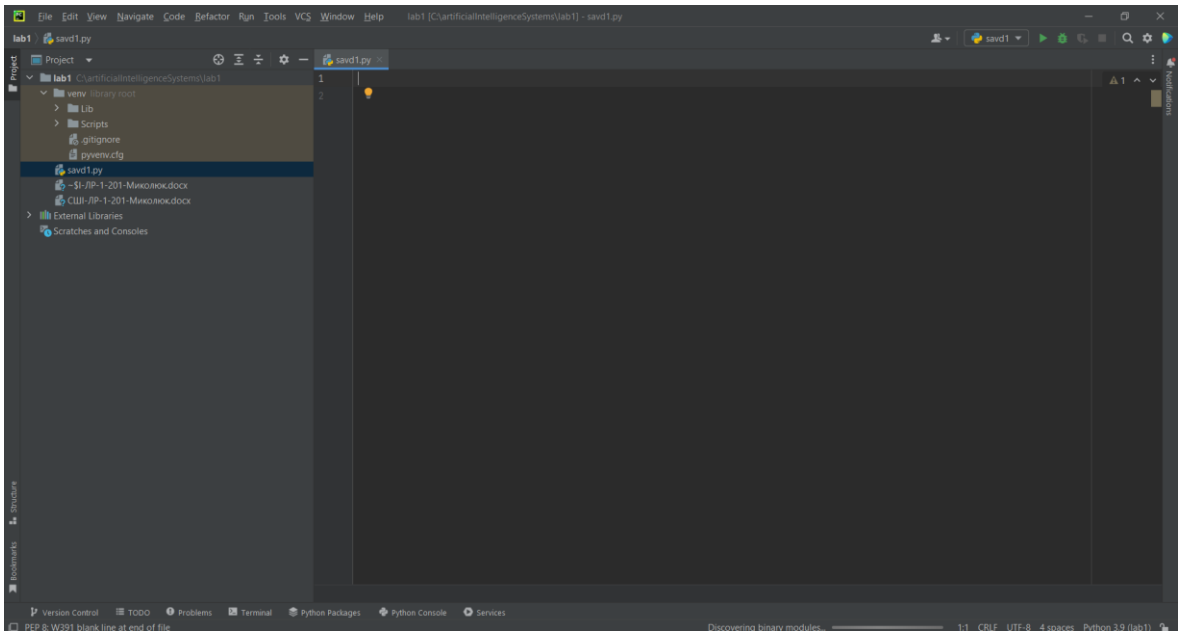

Рис 1. Середовище програмування PyCharm

Завдання 1.

```
import numpy as np
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("\nLabels =", test_labels)
print("Encoded values =", list(encoded_values))
encoded_values = [3, 0, 4, 1]
decoded_list = encoder.inverse_transform(encoded_values)
print("\nEncoded values =", encoded_values)
print("Decoded labels =", list(decoded_list))
```

Результат виконання програми зображено на рисунку 2.

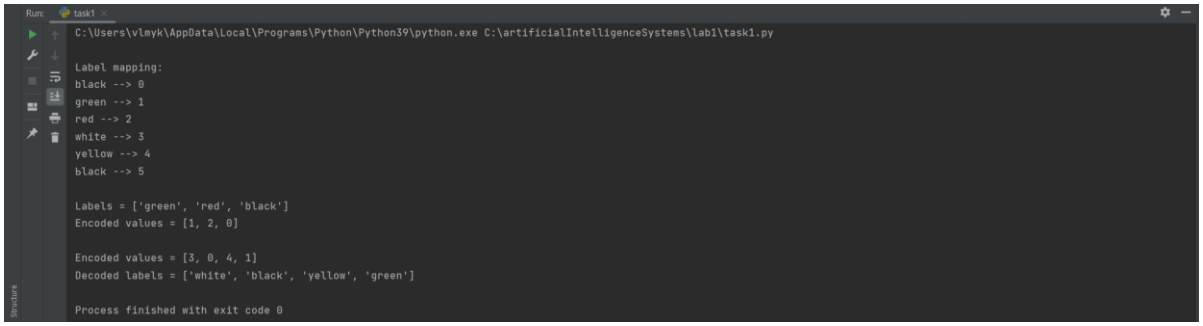

Рис 2. Виконання програми

Завдання 2.

Згідно варіанту 10.

10.	1.3	-3.9	6.5	-4.9	-2.2	1.3	2.2	6.5	-6.1	3.4	-3.4	-2.2	1.2

```
import numpy as np
from sklearn import preprocessing
input_data = np.array([[1.3, -3.9, 6.5],
                       [-4.9, -2.2, 1.3],
                       [2.2, 6.5, -6.1],
                       [3.4, -3.4, -2.2]])
data_binarized = preprocessing.Binarizer(threshold=1.2).transform(input_data)
print("\n Binarized data:\n", data_binarized)
print("\nBEFORE: ")
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))
data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis=0))
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)
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)
```

Результат виконання програми зображено на рисунку 3.

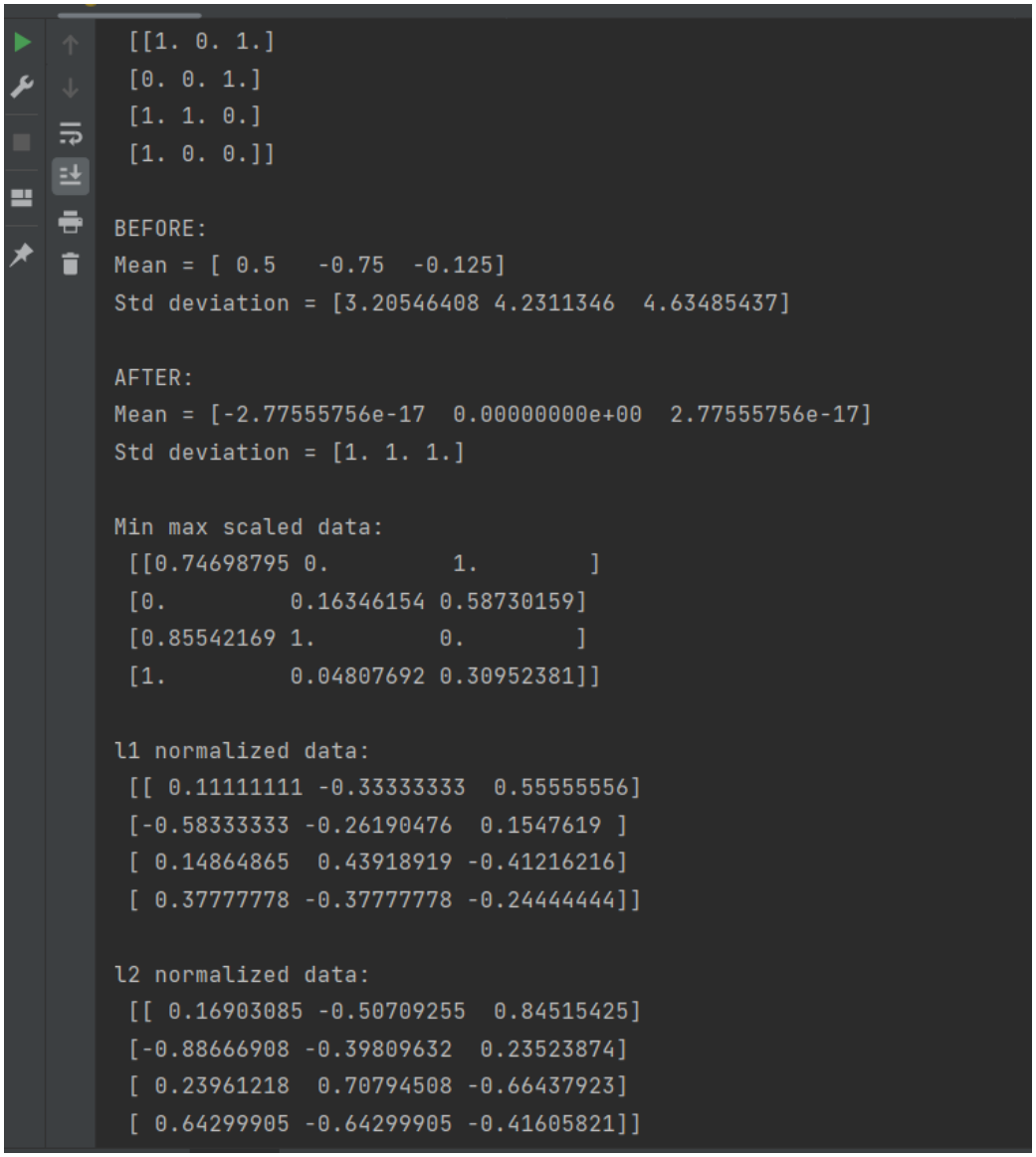

Рис 3. Виконання програми

Завдання 3. Класифікація логістичною регресією або логістичний класифікатор

Виникли проблеми з package utilities. Було завантажено цей package з https://github.com/PacktPublishing/Artificial-Intelligence-with-Python/tree/master/Chapter%2002/code

Код програми:

```
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
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 роботи програми зображено на рисунку 4.

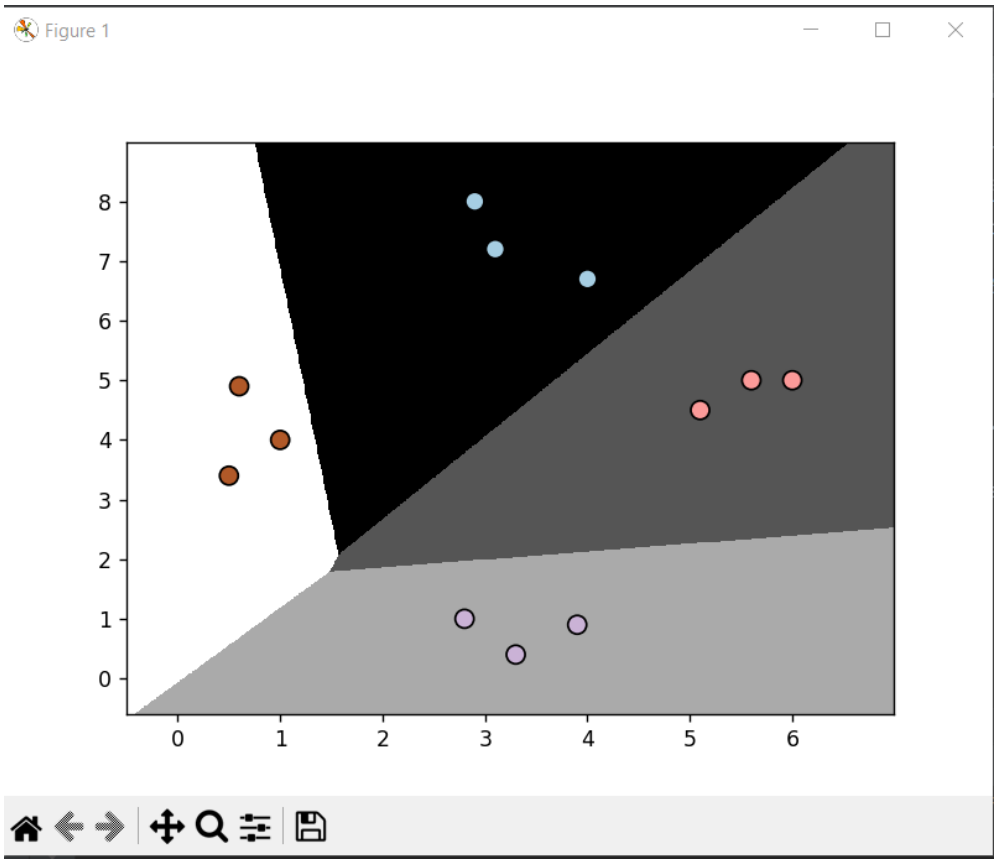

Рис 4. Результат роботи програми

Завдання 4. Класифікація наївним байєсовським класифікатором

```
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
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 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("Accuracy of the new classifier =", round(accuracy, 2), "%")
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)) + "%")
```

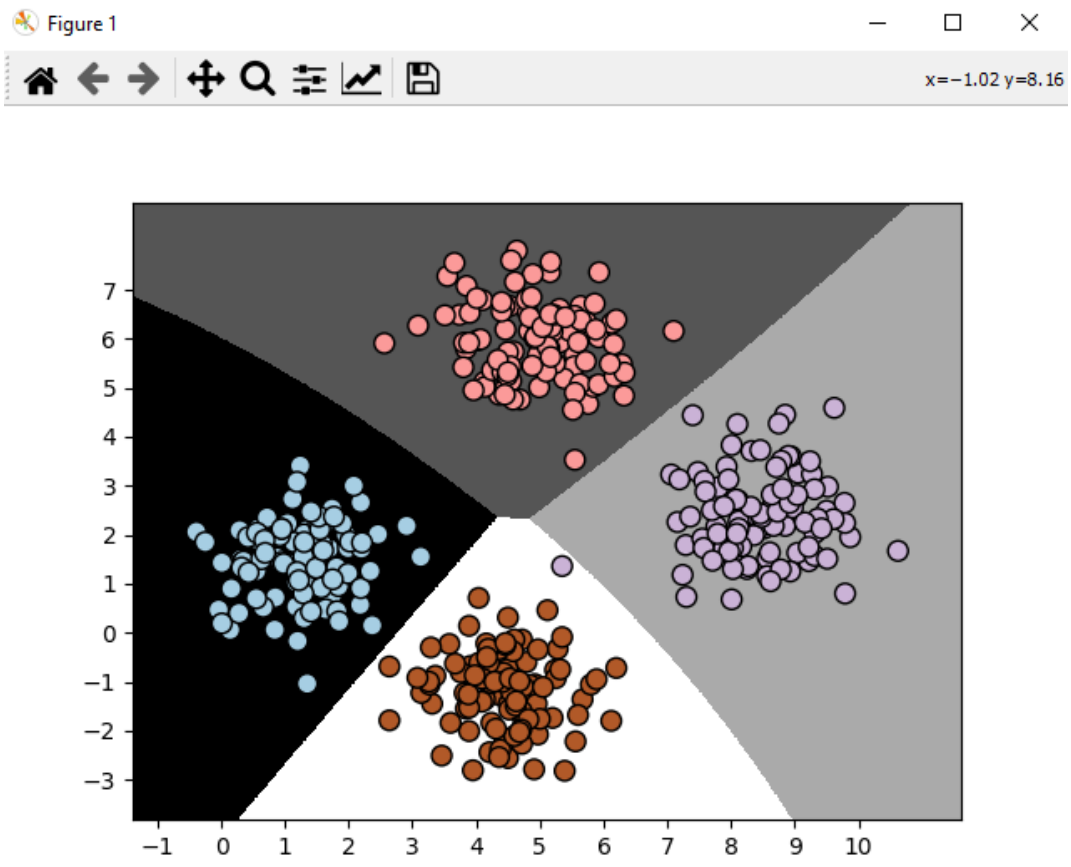

Рис 5. Результат виконання програми

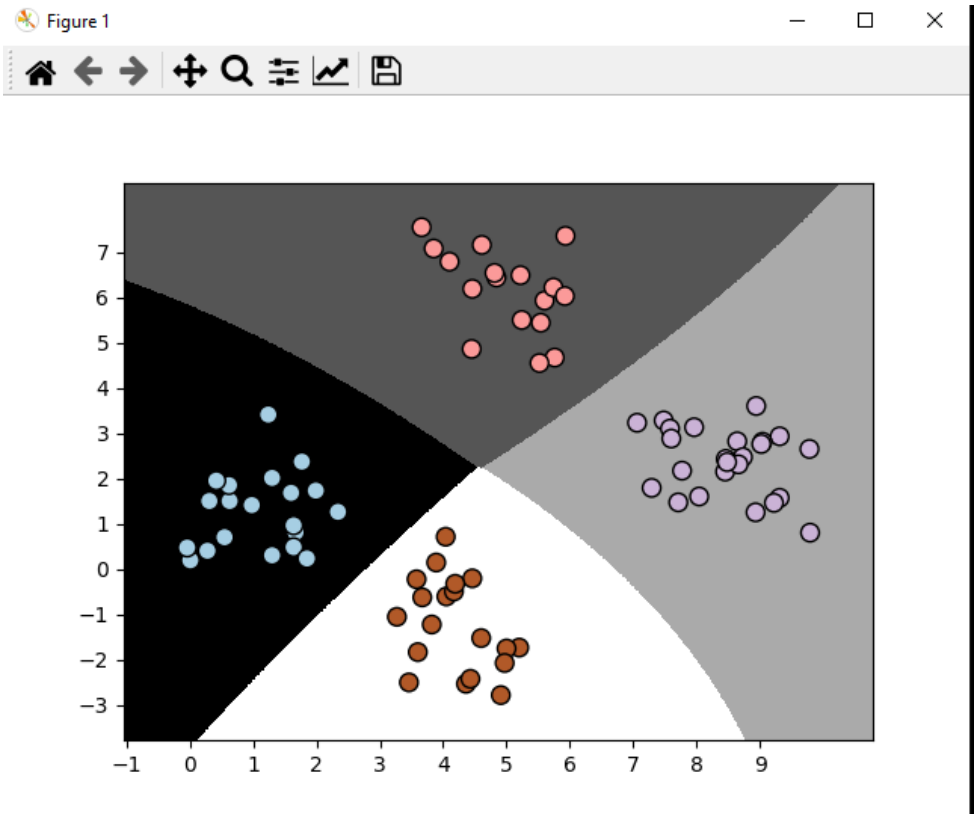

Рис 6. Результат виконання програми

Завдання 5. Вивчити метрики якості класифікації

```
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 MyConfusionMatrix(y_true, y_pred):
```

```
    TP, FN, FP, TN = find_conf_matrix_values(y_true, y_pred)
    return np.array([[TN, FP], [FN, TP]])


MyConfusionMatrix(df.actual_label.values,
                  df.predicted_RF.values)
assert np.array_equal(MyConfusionMatrix(df.actual_label.values,
df.predicted_RF.values), confusion_matrix(
    df.actual_label.values, df.predicted_RF.values)), 'MyConfusionMatrix() is not
correct for RF'
assert np.array_equal(MyConfusionMatrix(df.actual_label.values,
                                        df.predicted_LR.values),
                      confusion_matrix(df.actual_label.values,
                                       df.predicted_LR.values)),
'MyConfusionMatrix() is not correct for LR'
print(accuracy_score(df.actual_label.values, df.predicted_RF.values))
# calculates the fraction of samples

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


MyAccuracyScore(df.actual_label.values,
                df.predicted_RF.values) == accuracy_score(df.actual_label.values,

df.predicted_RF.values),
'MyAccuracyScore failed on'
assert MyAccuracyScore(df.actual_label.values,
                       df.predicted_LR.values) ==
accuracy_score(df.actual_label.values,

df.predicted_LR.values), 'MyAccuracyScore failed on LR'
print('Accuracy RF: % .3f ' % (MyAccuracyScore(df.actual_label.values,
                                               df.predicted_RF.values)))
print(recall_score(df.actual_label.values, df.predicted_RF.values))
f1_score(df.actual_label.values, df.predicted_RF.values)

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


def MyRecallScore(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)


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

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

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

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

```
def MyPrecisionScore(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)


MyPrecisionScore(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 MyPrecisionScore(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' % (MyPrecisionScore(df.actual_label.values,
                                               df.predicted_RF.values)))
print('Precision LR: %.3f' % (MyPrecisionScore(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 = MyRecallScore(y_true, y_pred)
    precision = MyPrecisionScore(y_true, y_pred)
    return 2 * (precision * recall) / (precision + recall)


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' % (MyAccuracyScore(df.actual_label.values,
                                              df.predicted_RF.values)))
print('Recall RF: %.3f' % (MyRecallScore(df.actual_label.values,
                                         df.predicted_RF.values)))
print('Precision RF: % .3f' % (MyPrecisionScore(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' % (
    MyAccuracyScore(df.actual_label.values, (df.model_RF >=
                                             0.25).astype('int').values)))
print('Recall RF: %.3f' % (MyRecallScore(df.actual_label.values, (df.model_RF >=

0.25).astype('int').values)))
print('Precision RF: %.3f' % (
    MyPrecisionScore(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()
```

Результат роботи програми зображено на рисунку 7.

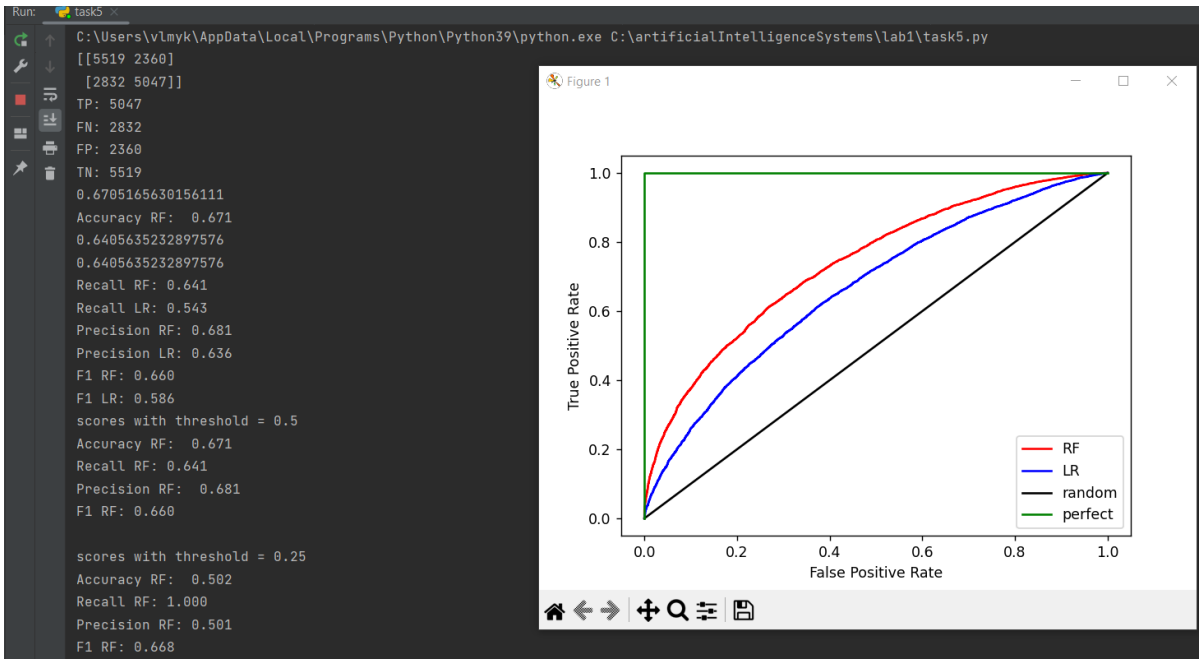

Рис 7. Результат роботи програми

Завдання 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.

```
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 роботи програми зображено на рисунку 8.

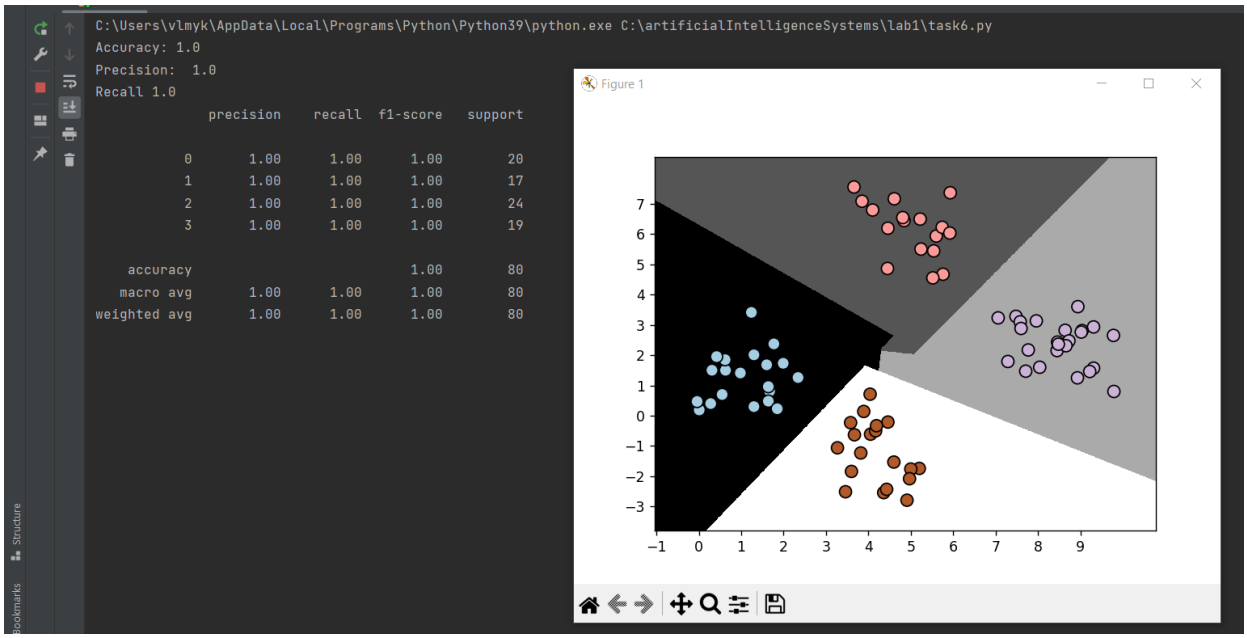

Рис 8. Результат роботи програми

Результат показує, що NBC працює краще, ніж SVM, це вірно тільки для відповідних параметрів, але за інших параметрів можна виявити, що SVM працює краще.

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