**ЛАБОРАТОРНА РОБОТА №1**

**ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ**

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

**Хід роботи**

***Завдання №2.1***. Попередня обробка даних.

Лістинг коду:

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]])  
  
data\_binarized = preprocessing.Binarizer(threshold=2.1).transform(input\_data)  
print(f"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("l2 normalized data:\n", data\_normalized\_l2)

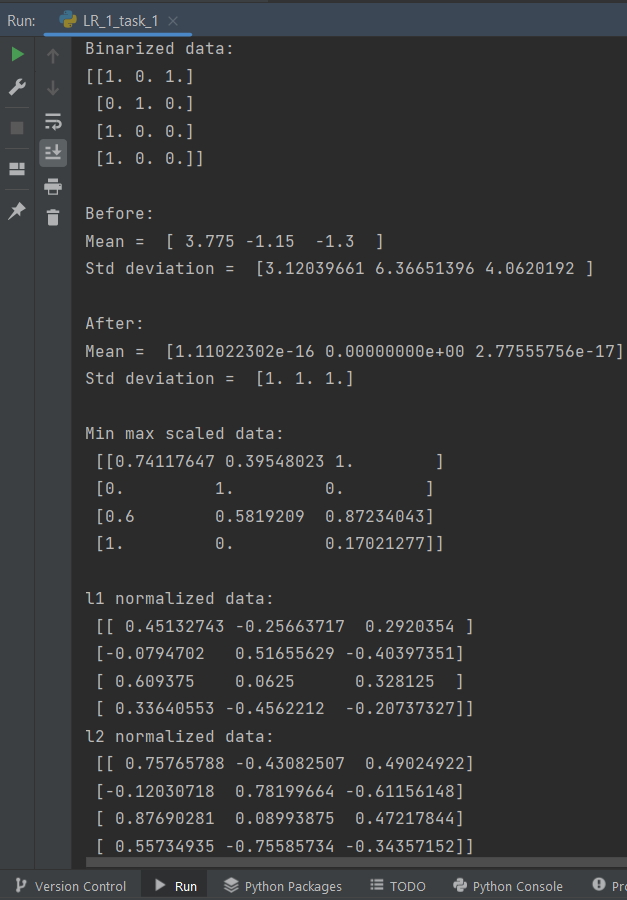


Рис. 1 – Бінаризація та виключення середнього, масштабування та нормалізація.

Нормалізація L1 та L2 розрізняється в підході до обробки значень при обчисленні сум. L1 використовує абсолютні значення, у той час як L2 оперує квадратами цих значень. Застосування L2 може призвести до меншої точності та надійності порівняно з L1. Однак L1 ефективніше виявляє неточності вхідних даних, такі як викиди, тоді як L2 не виявляє їх так ефективно.

**Кодування міток**

Лістинг файлу LR\_1\_task\_1.py

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(f"{item} --> {i}")  
  
test\_labels = ['green', 'red', 'black']  
encoded\_values = encoder.transform(test\_labels)  
print("\nLabels: ", test\_labels)  
print("Encoded values: ", encoded\_values)  
  
encoded\_values = [3, 0, 4, 1]  
decoded\_list = encoder.inverse\_transform(encoded\_values)  
print("\nEncoded values: ", encoded\_values)  
print("Decoded labels: ", decoded\_list)

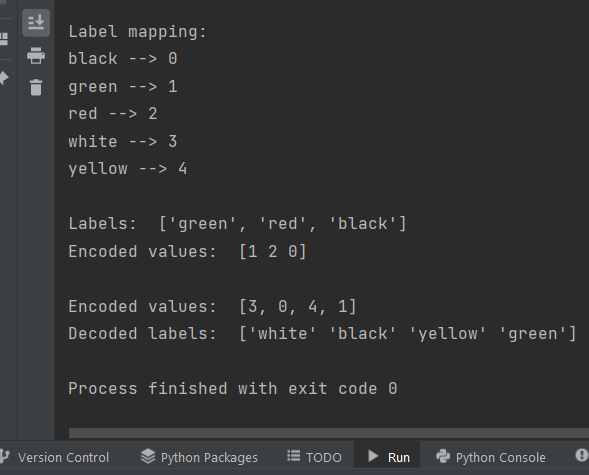


Рис. 2. – Кодування міток

***Завдання №2.2***. Попередня обробка нових даних

У коді програми попереднього завдання змінив дані по рядках (значення змінної input\_data) на значення відповідно варіанту таблиці 1 та виконав операції: Бінарізації, Виключення середнього, Масштабування, Нормалізації.

Обрав варіант №16 (відповідно номера за списком) відповідно до таблиці 1. Лістинг коду файлу LR\_1\_task\_2.py:

import numpy as np  
from sklearn import preprocessing  
  
input\_data = np.array([[-3.3, -1.6, 6.1],  
 [2.4, -1.2, 4.3],  
 [-3.2, 5.5, -6.1],  
 [-4.4, 1.4, -1.2]])  
  
data\_binarized = preprocessing.Binarizer(threshold=2.1).transform(input\_data)  
print(f"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("l2 normalized data:\n", data\_normalized\_l2)

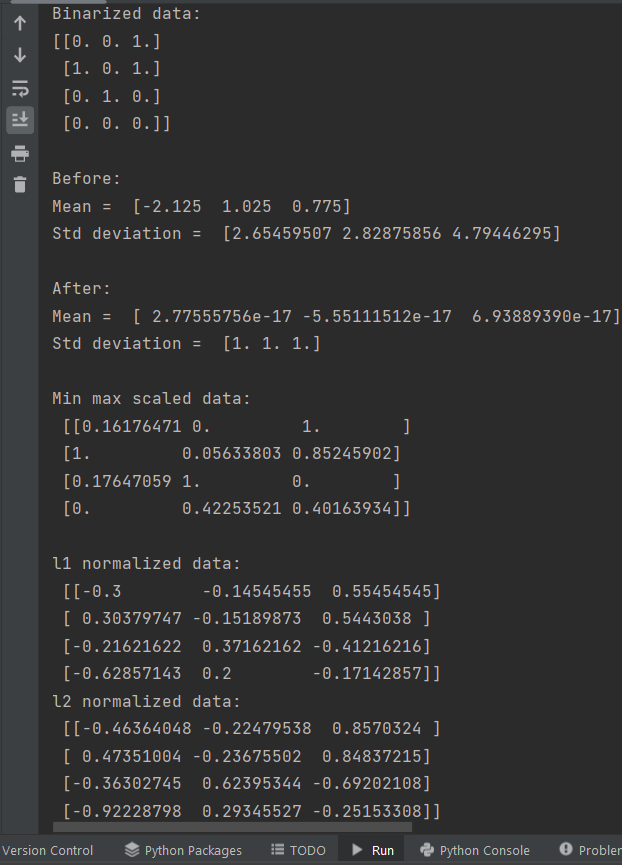


Рис. 3. – Бінаризація та виключення середнього власних даних, масштабування та нормалізація власних даних

***Завдання №2.3***. Класифікація логістичною регресією або логістичний класифікатор.

Лістинг коду файлу LR\_1\_task\_3.py:

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)

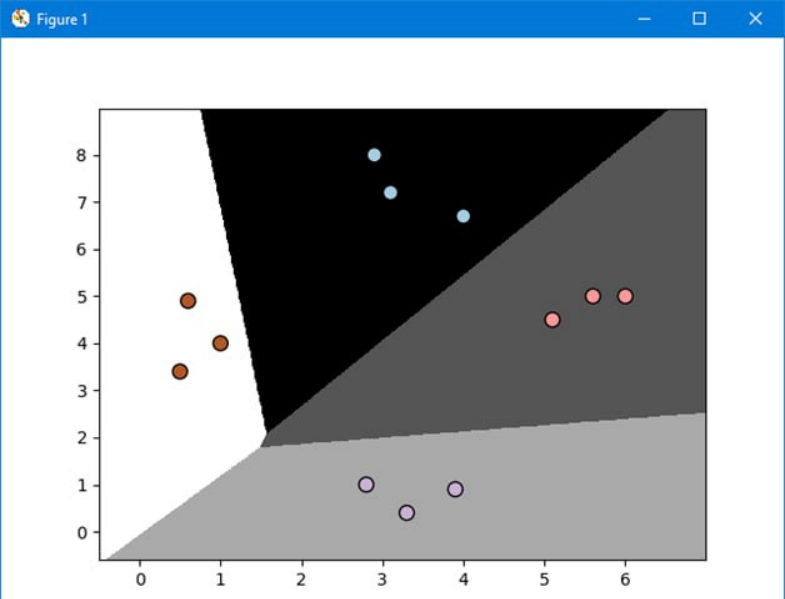


Рис. 4. – Результат класифікації лінійною регресією

**Завдання №2.4**. Класифікація наївним байєсовським класифікатором.

Лістинг коду файлу LR\_1\_task\_4.py:

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)

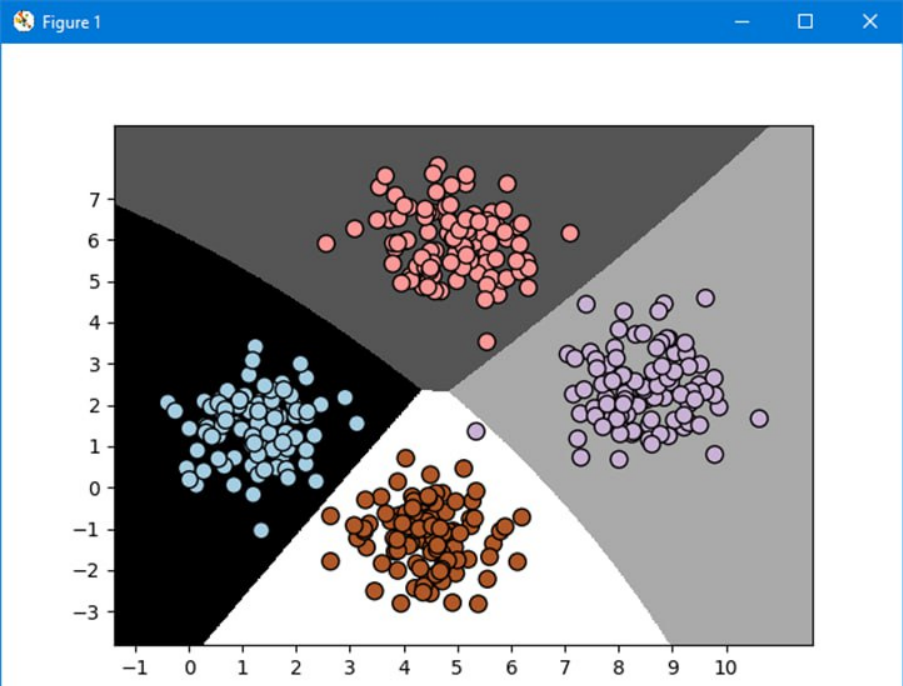


Рис. 5. – Якість класифікатора та відображення результату класифікації

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 = cross\_val\_score(classifier\_new, X\_test, y\_test, scoring='accuracy', cv=num\_folds)  
print(f"Accuracy: {round(100 \* accuracy\_values.mean(), 2)}%")  
precision\_values = cross\_val\_score(classifier\_new, X\_test, y\_test, scoring='precision\_weighted', cv=num\_folds)  
print(f"Precision: {round(100 \* precision\_values.mean(), 2)}%")  
recall\_values = cross\_val\_score(classifier\_new, X\_test, y\_test, scoring='recall\_weighted', cv=num\_folds)  
print(f"Recall: {round(100 \* recall\_values.mean(), 2)}%")  
f1\_values = cross\_val\_score(classifier\_new, X\_test, y\_test, scoring='f1\_weighted', cv=num\_folds)  
print(f"F1: {round(100 \* f1\_values.mean(), 2)}%")

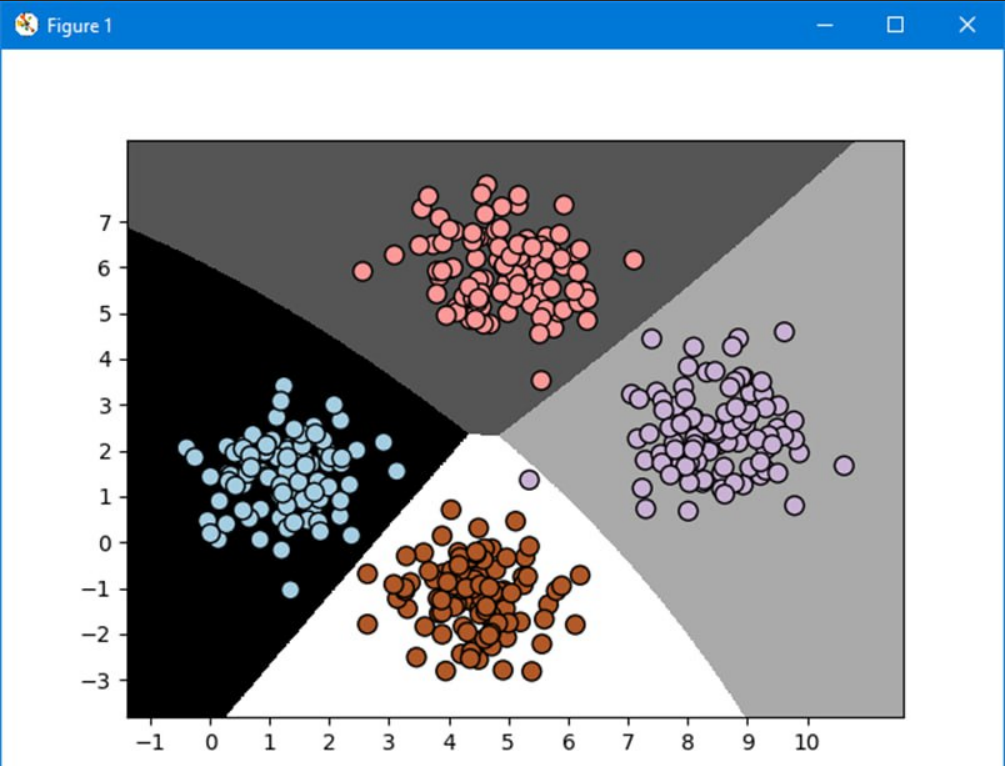


Рис. 6. – Отримані дані

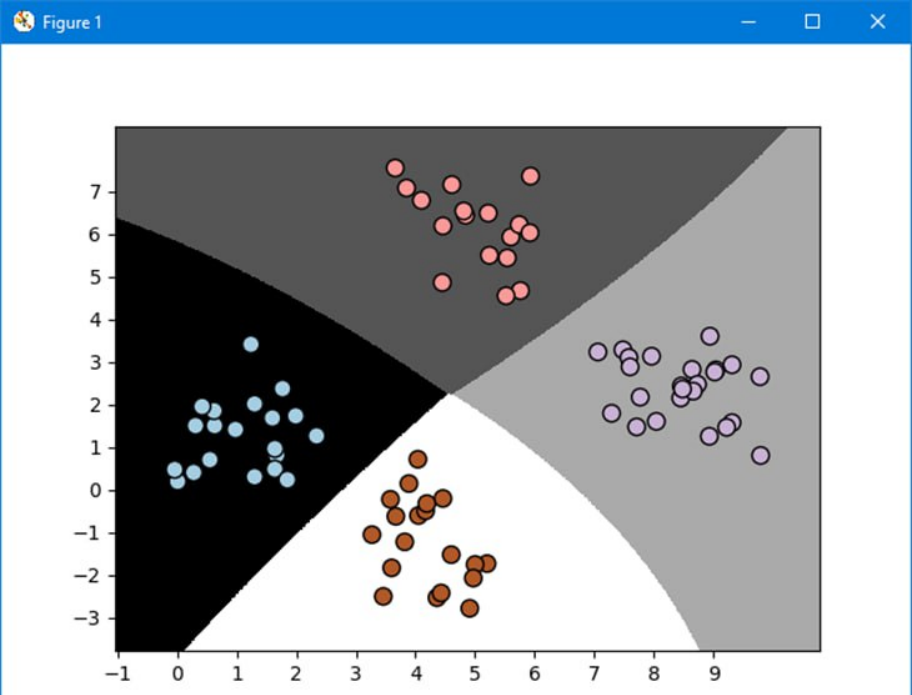
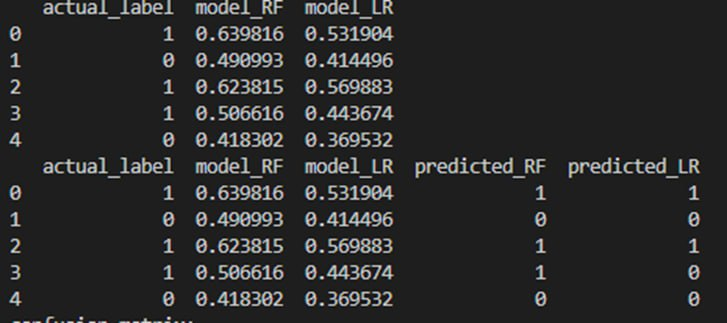


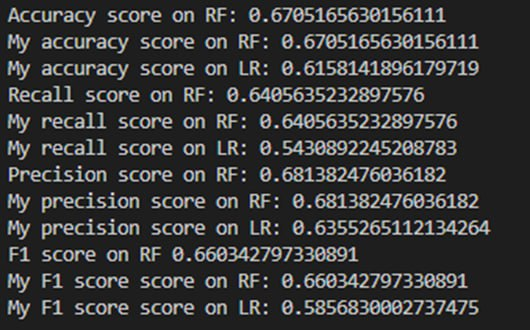
Рис. 7. – Зображення результату класифікації тестових даних

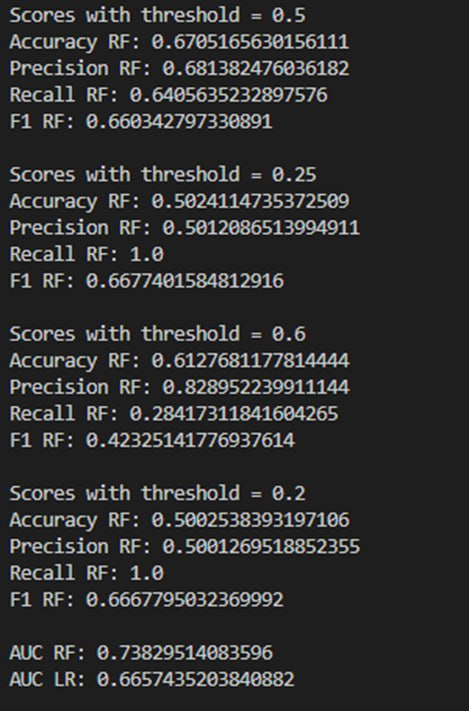
***Завдання №2.5***. Вивчити метрики якості класифікації

Лістинг коду файлу LR\_1\_task\_5.py:

import numpy as np  
import pandas as pd  
from sklearn.metrics import confusion\_matrix, accuracy\_score, recall\_score, precision\_score, f1\_score, \  
 roc\_curve, roc\_auc\_score  
import matplotlib.pyplot as plt  
  
df = pd.read\_csv('data\_metrics.csv')  
print(df.head())  
thresh = 0.5  
df['predicted\_RF'] = (df.model\_RF >= thresh).astype('int')  
df['predicted\_LR'] = (df.model\_LR >= thresh).astype('int')  
print(df.head())  
actual = df.actual\_label.values  
model\_RF = df.model\_RF.values  
model\_LR = df.model\_LR.values  
predicted\_RF = df.predicted\_RF.values  
predicted\_LR = df.predicted\_LR.values  
conf\_matr = confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)  
print("confusion\_matrix:\n", conf\_matr)  
  
  
def find\_TP(y\_true, y\_pred):  
 return sum((y\_true == 1) & (y\_pred == 1))  
  
  
def find\_FN(y\_true, y\_pred):  
 return sum((y\_true == 1) & (y\_pred == 0))  
  
  
def find\_FP(y\_true, y\_pred):  
 return sum((y\_true == 0) & (y\_pred == 1))  
  
  
def find\_TN(y\_true, y\_pred):  
 return sum((y\_true == 0) & (y\_pred == 0))  
  
  
def find\_conf\_matrix\_values(y\_true, y\_pred):  
 *"""* ***:param*** *y\_true: List with true data of classification* ***:param*** *y\_pred: List with predicted data of classification* ***:return****: 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 Lonskiy\_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]])  
  
  
print("Lonskiy\_confusion\_matrix:\n", Lonskiy\_confusion\_matrix(actual, predicted\_RF))  
assert np.array\_equal(Lonskiy\_confusion\_matrix(actual, predicted\_RF),  
 confusion\_matrix(actual, predicted\_RF)), \  
 'my confusion\_matrix() is not correct for RF'  
assert np.array\_equal(Lonskiy\_confusion\_matrix(actual, predicted\_LR),  
 confusion\_matrix(actual, predicted\_LR)), \  
 'my confusion\_matrix() is not correct for lR'  
# Accuracy  
score = accuracy\_score(actual, predicted\_RF)  
print("Accuracy score on RF:", score)  
  
  
def Lonskiy\_accuracy\_score(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return (TP + TN) / (TP + FN + FP + TN)  
  
  
assert Lonskiy\_accuracy\_score(actual, predicted\_RF) == accuracy\_score(actual, predicted\_RF), \  
 'my accuracy\_score failed RF'  
assert Lonskiy\_accuracy\_score(actual, predicted\_LR) == accuracy\_score(actual, predicted\_LR), \  
 'my accuracy\_score failed LR'  
print("My accuracy score on RF:", Lonskiy\_accuracy\_score(actual, predicted\_RF))  
print("My accuracy score on LR:", Lonskiy\_accuracy\_score(actual, predicted\_LR))  
# Recall  
print('Recall score on RF:', recall\_score(actual, predicted\_RF))  
  
  
def Lonskiy\_recal\_score(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return TP / (TP + FN)  
  
  
assert Lonskiy\_recal\_score(actual, predicted\_RF) == recall\_score(actual,  
 predicted\_RF), \  
 'my recal\_score fails on RF'  
assert Lonskiy\_recal\_score(actual, predicted\_LR) == recall\_score(actual,  
 predicted\_LR), \  
 'my recal\_score fails on LR'  
print("My recall score on RF:", Lonskiy\_recal\_score(actual, predicted\_RF))  
print("My recall score on LR:", Lonskiy\_recal\_score(actual, predicted\_LR))  
# Precision  
print("Precision score on RF:", precision\_score(actual, predicted\_RF))  
  
  
def Lonskiy\_precision\_score(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return TP / (TP + FP)  
  
  
assert Lonskiy\_precision\_score(actual, predicted\_RF) == precision\_score(actual, predicted\_RF), \  
 'my precision\_score fails on RF'  
assert Lonskiy\_precision\_score(actual, predicted\_LR) == precision\_score(actual, predicted\_LR), \  
 'my precision\_score fails on LR'  
print("My precision score on RF:", Lonskiy\_precision\_score(actual, predicted\_RF))  
print("My precision score on LR:", Lonskiy\_precision\_score(actual, predicted\_LR))  
# F1 score  
print("F1 score on RF", f1\_score(actual, predicted\_RF))  
  
  
def Lonskiy\_f1\_score(y\_true, y\_pred):  
 precision = Lonskiy\_precision\_score(y\_true, y\_pred)  
 recall = Lonskiy\_recal\_score(y\_true, y\_pred)  
 return (2 \* (precision \* recall)) / (precision + recall)  
  
  
assert Lonskiy\_f1\_score(actual, predicted\_RF) == f1\_score(actual, predicted\_RF), \  
 'my f1\_score fails on RF'  
assert Lonskiy\_f1\_score(actual, predicted\_LR) == f1\_score(actual, predicted\_LR), \  
 'my f1\_score fails on LR'  
print("My F1 score score on RF:", Lonskiy\_f1\_score(actual, predicted\_RF))  
print("My F1 score score on LR:", Lonskiy\_f1\_score(actual, predicted\_LR))  
print()  
  
  
def test\_thresholds(threshold: float = .5):  
 print(f"Scores with threshold = {threshold}")  
 predicted = (df.model\_RF >= threshold).astype('int')  
 print("Accuracy RF:", Lonskiy\_accuracy\_score(actual, predicted))  
 print("Precision RF:", Lonskiy\_precision\_score(actual, predicted))  
 print("Recall RF:", Lonskiy\_recal\_score(actual, predicted))  
 print("F1 RF:", Lonskiy\_f1\_score(actual, predicted))  
 print()  
 test\_thresholds()  
 test\_thresholds(.25)  
 test\_thresholds(.6)  
 test\_thresholds(.20)  
 # ROC  
 # Curve  
 fpr\_RF, tpr\_RF, thresholds\_RF = roc\_curve(actual, model\_RF)  
 fpr\_LR, tpr\_LR, thresholds\_LR = roc\_curve(actual, model\_LR)  
 # AUC  
 auc\_RF = roc\_auc\_score(actual, model\_RF)  
 auc\_LR = roc\_auc\_score(actual, model\_LR)  
 print("AUC RF:", auc\_RF)  
 print("AUC LR:", auc\_LR)  
 plt.plot(fpr\_RF, tpr\_RF, 'r-', label=f'AUC RF: {auc\_RF}')  
 plt.plot(fpr\_LR, tpr\_LR, 'b-', label=f'AUC LR: {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()







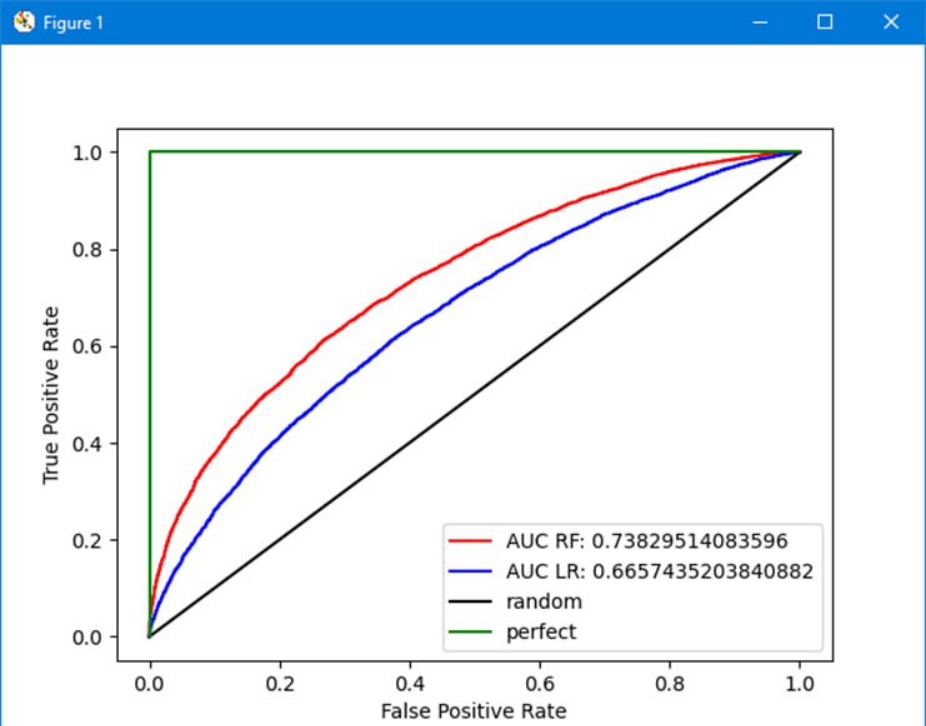


Рис. 8 – Графік отриманих значень ROC

На підставі наданих метрик, можна визначити, що випадковий ліс (RF) видається оптимальним вибором для цих даних, оскільки він проявляє вищу точність, більший recall і precision, а також кращий AUC порівняно з лінійною регресією (LR).

***Завдання №2.6.*** Розробіть програму класифікації даних в файлі data\_multivar\_nb.txt за допомогою машини опорних векторів (Support Vector Machine - SVМ). Розрахуйте показники якості класифікації.

Порівняйте їх з показниками наївного байєсівського класифікатора.

Зробіть висновки яку модель класифікації краще обрати і чому.

Лістинг коду файлу LR\_1\_task\_6.py

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn import svm  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score  
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, test\_size=0.2, random\_state=3)  
  
classifier = svm.SVC()  
  
classifier.fit(X\_train, y\_train)  
  
y\_test\_pred = classifier.predict(X\_test)  
visualize\_classifier(classifier, X\_test, y\_test)  
num\_folds = 3  
accuracy\_values = cross\_val\_score(classifier, X\_test, y\_test, scoring='accuracy', cv=num\_folds)  
print(f"Accuracy: {round(100 \* accuracy\_values.mean(), 2)}%")  
precision\_values = cross\_val\_score(classifier, X\_test, y\_test, scoring='precision\_weighted', cv=num\_folds)  
print(f"Precision: {round(100 \* precision\_values.mean(), 2)}%")  
recall\_values = cross\_val\_score(classifier, X\_test, y\_test, scoring='recall\_weighted', cv=num\_folds)  
print(f"Recall: {round(100 \* recall\_values.mean(), 2)}%")  
f1\_values = cross\_val\_score(classifier, X\_test, y\_test, scoring='f1\_weighted', cv=num\_folds)  
print(f"F1: {round(100 \* f1\_values.mean(), 2)}%")

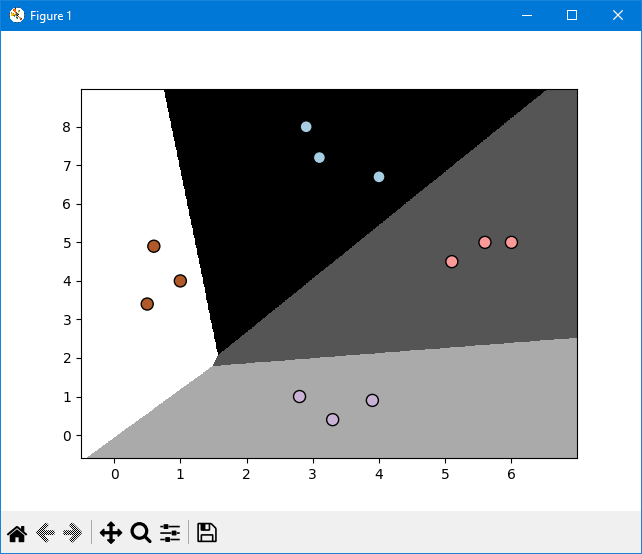


Рис. 9 – Показники класифікації та зображення результату класифікації тестових даних за допомоги SVM

Метод опорних векторів (SVM) може бути привабливим для задач класифікації завдяки своїй швидкості та простоті, проте він може виявитися обмеженим у випадку багатокласової класифікації. Крім того, якщо всі показники в даних мають подібні або малозначущі значення, це може обмежити кількість даних, необхідних для ефективного навчання моделі.

***Висновок***: Під час виконання лабораторної роботи було отримано нові навички та успішно виконано ряд завдань. Були вивчені та застосовані різноманітні методи попередньої обробки даних, такі як бінаризація, виключення середнього, масштабування, нормалізація та кодування міток. Здійснено класифікацію даних за допомогою методу логістичної регресії. Розроблено власні функції для отримання та обробки метрик якості, і вони були порівняні з вбудованими функціями. Також проведено застосування методу машини опорних векторів (SVM) для класифікації даних. Загалом, виконана робота сприяла розвитку навичок з обробки та аналізу даних, використання різних алгоритмів класифікації, а також розроблено функції для обчислення та порівняння метрик якості класифікації.