**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 2.1.2-4. Попередня обробка даних**

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("\nBinarized 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 MinМax  
data\_scaler\_minmax = preprocessing.MinMaxScaler(feature\_range=(0, 1))  
data\_scaled\_minmax = data\_scaler\_minmax.fit\_transform(input\_data)  
print("\nМin 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)

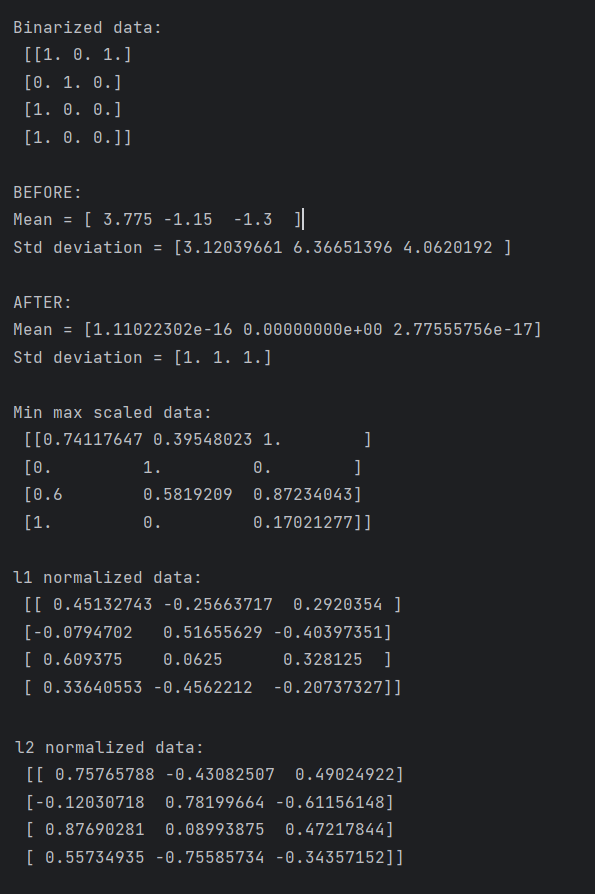


Рис.1. Результат виконання

**Завдання 2.1.5. Кодування міток**

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))

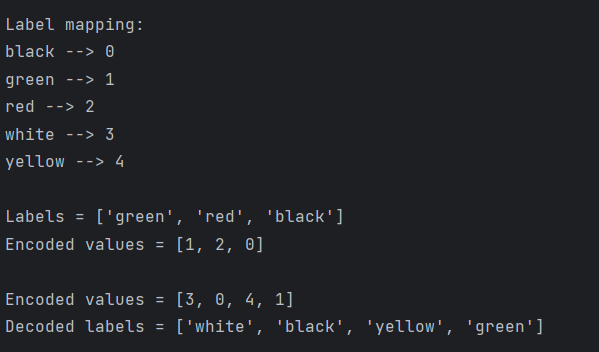


Рис.2. Результат виконання LR\_1\_task\_1.py

**Завдання 2.2. Попередня обробка нових даних**

import numpy as np  
from sklearn import preprocessing  
  
input\_data = np.array([[-2.3, -1.6, -6.1],  
 [-2.4, -1.2, 4.3],  
 [3.2, 3.1, 6.1],  
 [-4.4, 1.4, -1.2]])  
  
# Бінаризація даних  
data\_binarized = preprocessing.Binarizer(threshold=2.1).transform(input\_data)  
print("\nBinarized 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 MinМax  
data\_scaler\_minmax = preprocessing.MinMaxScaler(feature\_range=(0, 1))  
data\_scaled\_minmax = data\_scaler\_minmax.fit\_transform(input\_data)  
print("\nМin 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)

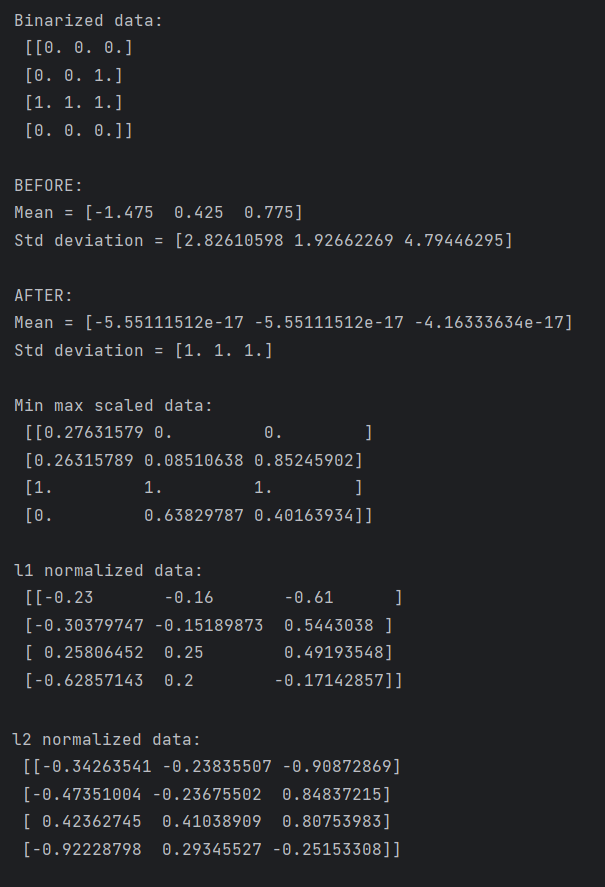


Рис.3. Результат виконання LR\_1\_task\_2.py

**Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор**

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)

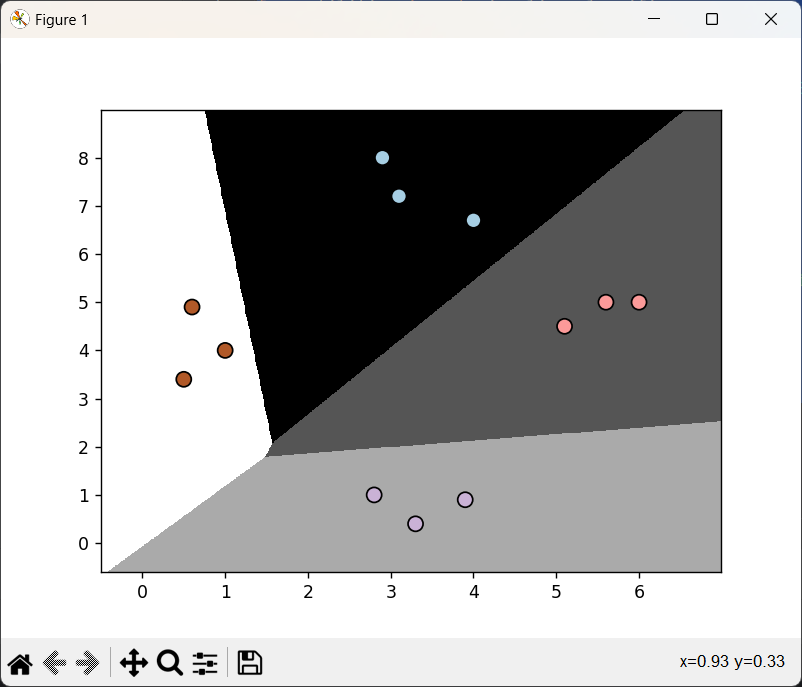


Рис.4. Результат виконання LR\_1\_task\_3.py

**Завдання 2.4. 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  
  
# 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)) + "%")

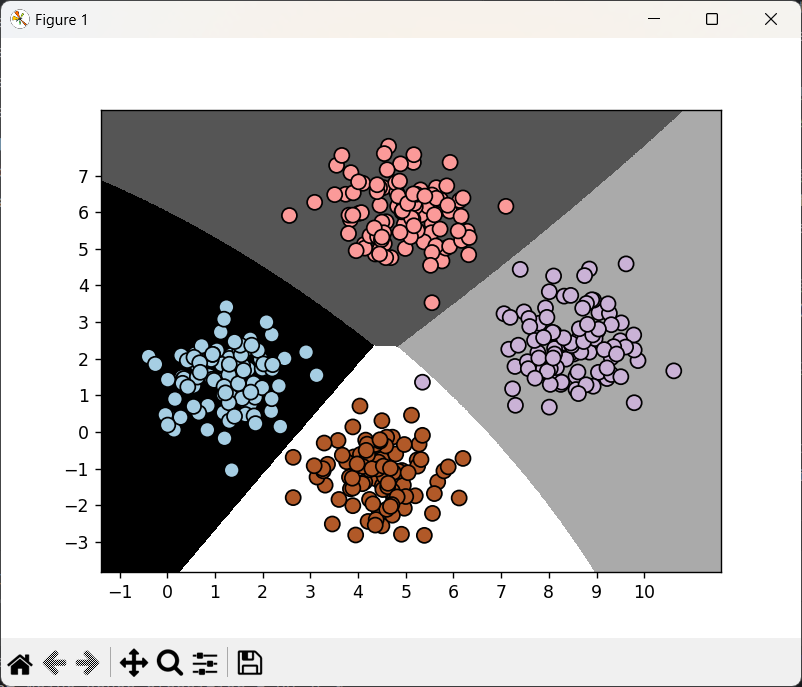


Рис.5. Результат виконання першого способу LR\_1\_task\_4.py

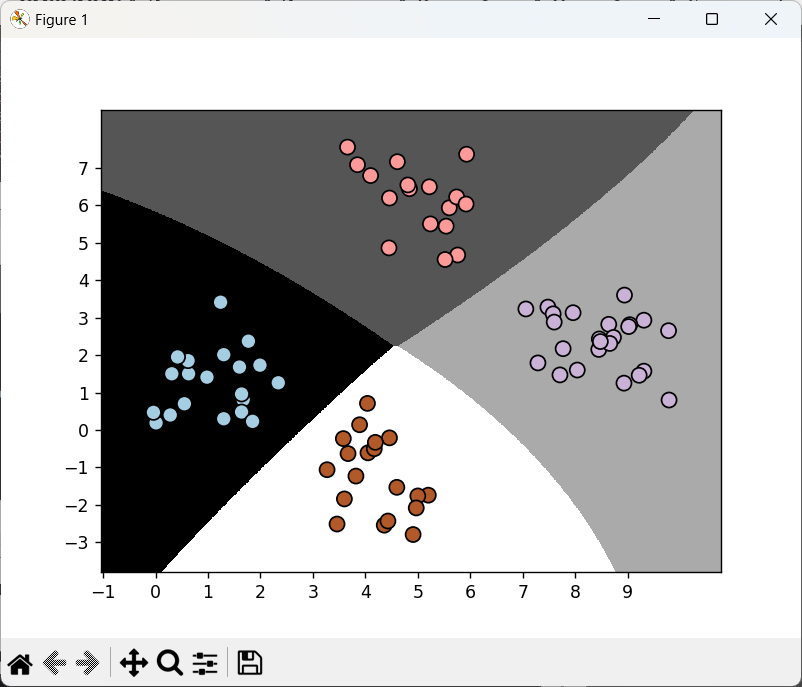


Рис.6. Результат виконання другого способу LR\_1\_task\_4.py

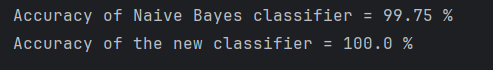


Рис.7. Порівняння точності способів LR\_1\_task\_4.py

Після використання перехресної перевірки, точність збільшилась до 100%

**Завдання 2.5. 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 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 voitko\_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 voitko\_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 voitko\_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' % (voitko\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))  
  
print(recall\_score(df.actual\_label.values, df.predicted\_RF.values))  
  
  
def voitko\_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 voitko\_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 voitko\_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' % (voitko\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall LR: %.3f' % (voitko\_recall\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
precision\_score(df.actual\_label.values, df.predicted\_RF.values)  
  
  
def voitko\_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 voitko\_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 voitko\_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' % (voitko\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision LR: %.3f' % (voitko\_precision\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
f1\_score(df.actual\_label.values, df.predicted\_RF.values)  
  
  
def voitko\_f1\_score(y\_true, y\_pred): # calculates the F1 score  
 recall = voitko\_recall\_score(y\_true, y\_pred)  
 precision = voitko\_precision\_score(y\_true, y\_pred)  
 return 2 \* (precision \* recall) / (precision + recall)  
  
  
assert voitko\_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 voitko\_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' % (voitko\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 LR: %.3f' % (voitko\_f1\_score(df.actual\_label.values, df.predicted\_LR.values)))  
print('scores with threshold = 0.5')  
  
print('Accuracy RF: % .3f' % (voitko\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall RF: %.3f' % (voitko\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision RF: % .3f' % (voitko\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 RF: %.3f' % (voitko\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('')  
  
threshold = 0.75  
print(f'Scores with threshold = {threshold}')  
print('Accuracy RF: % .3f' % (voitko\_accuracy\_score(df.actual\_label.values, (df.model\_RF >= threshold).astype('int').values)))  
print('Recall RF: %.3f' % (voitko\_recall\_score(df.actual\_label.values, (df.model\_RF >= threshold).astype('int').values)))  
print('Precision RF: %.3f' % (voitko\_precision\_score(df.actual\_label.values, (df.model\_RF >= threshold).astype('int').values)))  
print('F1 RF: %.3f' % (voitko\_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()

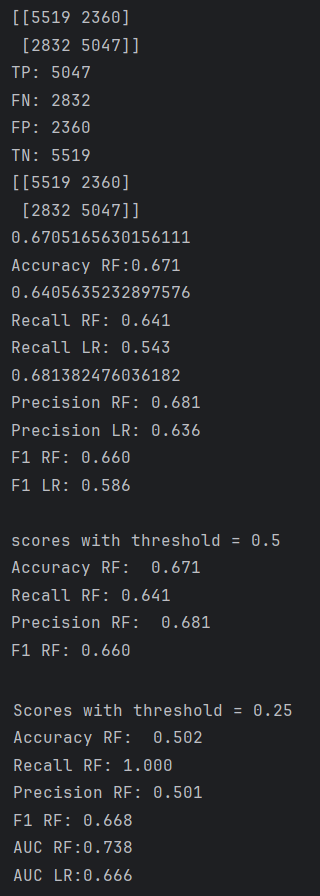


Рис.8. Результат виконання в консолі LR\_1\_task\_5.py

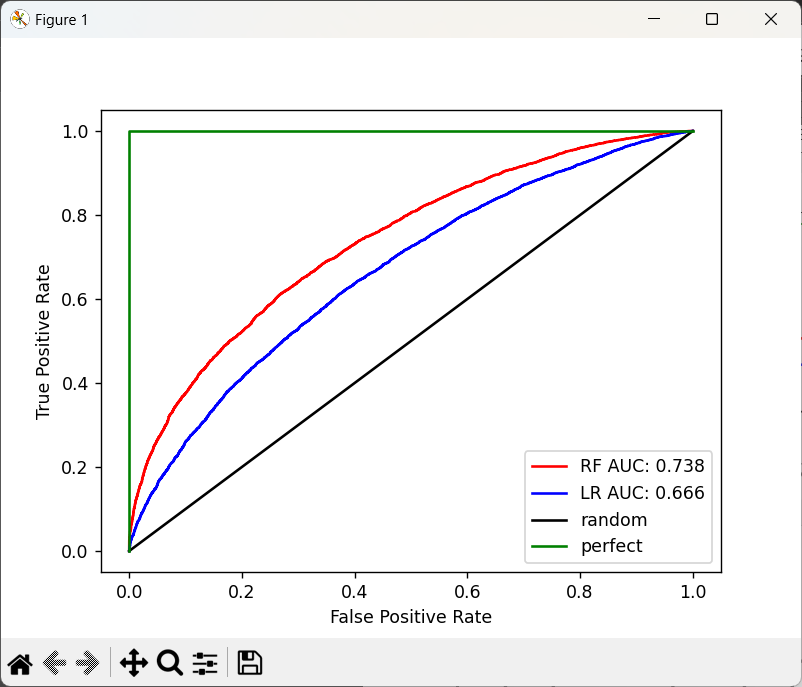
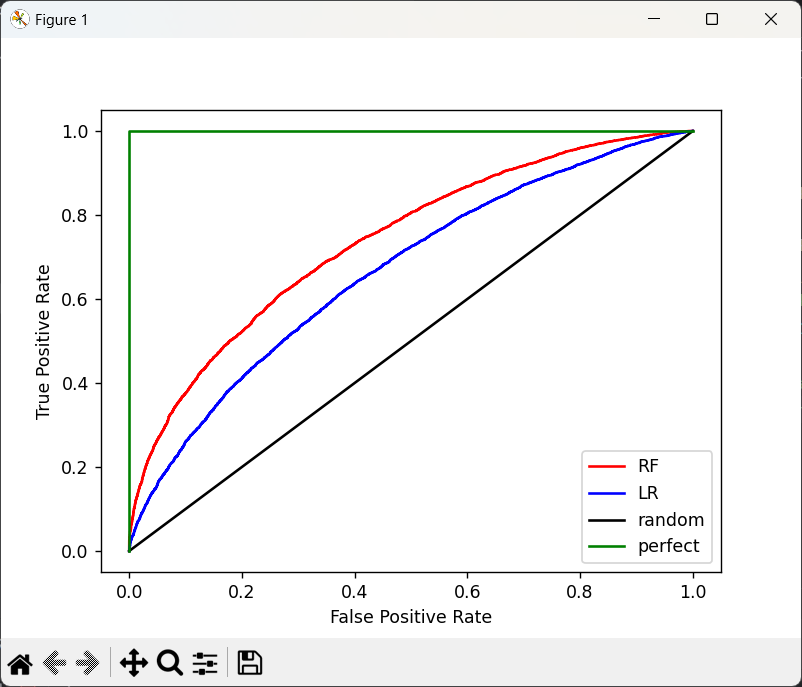


Рис.9. Результат виконання LR\_1\_task\_5.py

**Завдання 2.6.**

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)

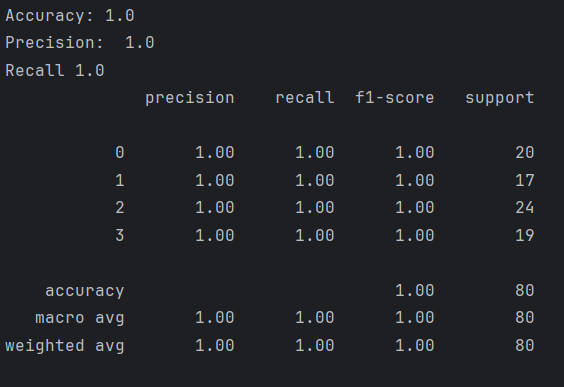
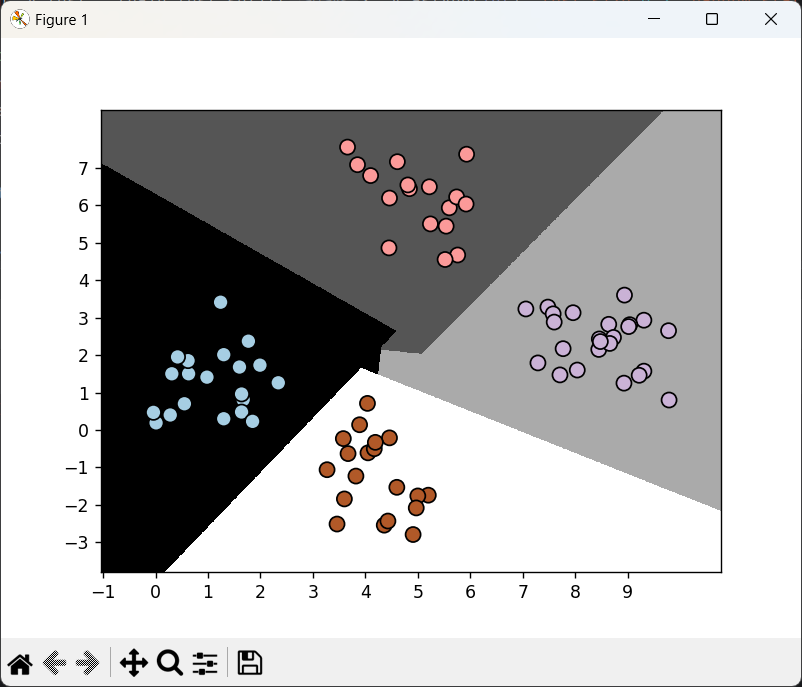


Рис.10. Результат виконання LR\_1\_task\_6.py

***Висновки:*** в ході виконання лабораторної роботи ми використовуючи спеціалізовані бібліотеки та мову програмування Python дослідили попередню обробку та класифікацію даних.