**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 2.1. - 2.1.4.**

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(f"\nBinarized data:\n{data\_binarized}")  
  
# Виведення середнього значення та стандартного відхилення  
print("\nBEFORE: ")  
print(f"Mean = {input\_data.mean(axis=0)}")  
print(f"Std deviation = {input\_data.std(axis=0)}")  
  
# Исключение среднего  
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)}")  
  
# Масштабування MinМax  
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}")  
  
# Нормалізація даних  
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:**

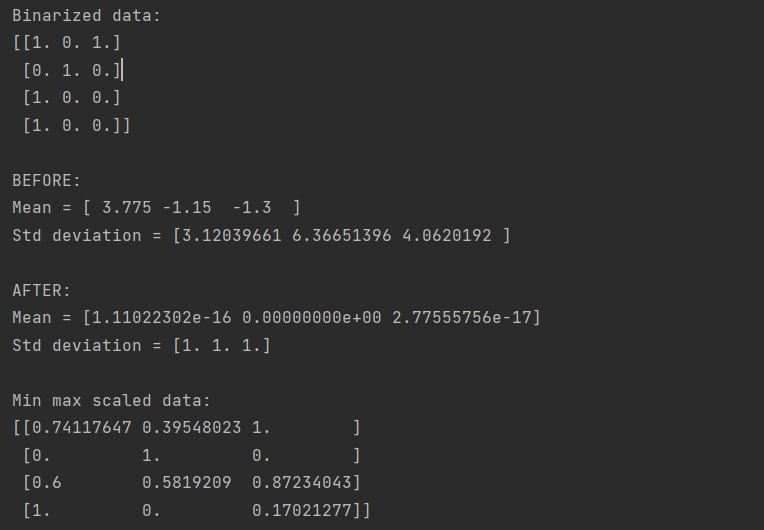
****

Рис.1 Результат виконання

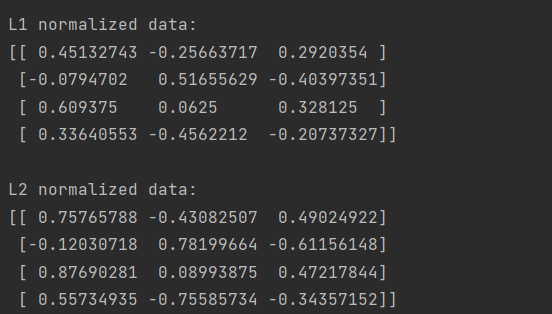
****

Рис.2 Результат виконання

**Завдання 2.1.5.**

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(f"\nLabels = {test\_labels}")  
print(f"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(f"\nEncoded values = {encoded\_values}")  
print(f"Decoded labels = {list(decoded\_list)}")

**Результат виконання:**

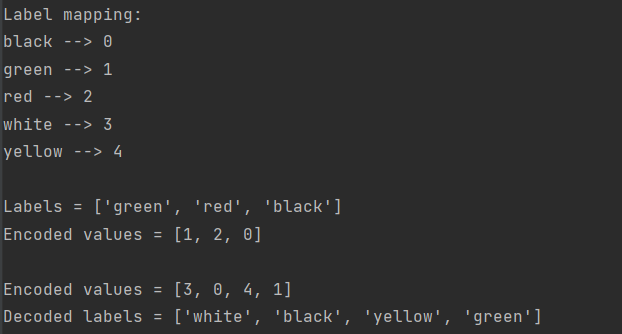


Рис.3 Результат виконання

**Завдання 2.2**



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, 5.5, -6.1],  
 [-4.4, 1.4, -1.2]])  
# Бінаризація даних  
data\_binarized = preprocessing.Binarizer(threshold=2.1).transform(input\_data)  
print(f"\nBinarized data:\n{data\_binarized}")  
  
# Виведення середнього значення та стандартного відхилення  
print("\nBEFORE: ")  
print(f"Mean = {input\_data.mean(axis=0)}")  
print(f"Std deviation = {input\_data.std(axis=0)}")  
  
# Исключение среднего  
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)}")  
  
# Масштабування MinМax  
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}")  
  
# Нормалізація даних  
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:**

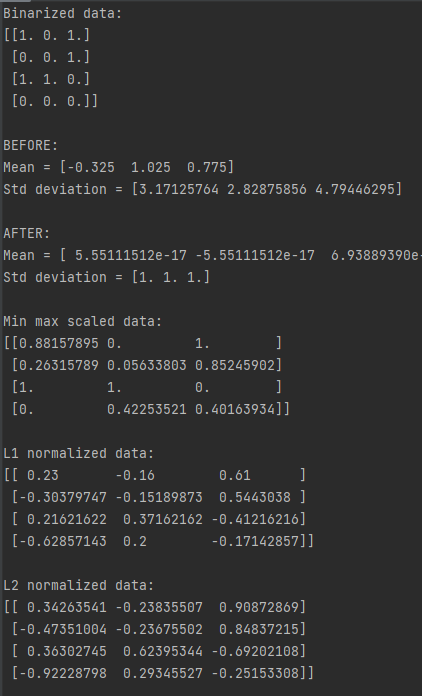


Рис.4 Результат виконання

**Завдання 2.3**

**utilities.py**

import numpy as np  
import matplotlib.pyplot as plt  
  
def visualize\_classifier(classifier, X, y):  
 # Define the minimum and maximum values for X and Y  
 # that will be used in the mesh grid  
 min\_x, max\_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0  
 min\_y, max\_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0  
  
 # Define the step size to use in plotting the mesh grid  
 mesh\_step\_size = 0.01  
  
 # Define the mesh grid of X and Y values  
 x\_vals, y\_vals = np.meshgrid(np.arange(min\_x, max\_x, mesh\_step\_size), np.arange(min\_y, max\_y, mesh\_step\_size))  
  
 # Run the classifier on the mesh grid  
 output = classifier.predict(np.c\_[x\_vals.ravel(), y\_vals.ravel()])  
  
 # Reshape the output array  
 output = output.reshape(x\_vals.shape)  
  
 # Create a plot  
 plt.figure()  
  
 # Choose a color scheme for the plot  
 plt.pcolormesh(x\_vals, y\_vals, output, cmap=plt.cm.gray)  
  
 # Overlay the training points on the plot  
 plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='black', linewidth=1, cmap=plt.cm.Paired)  
  
 # Specify the boundaries of the plot  
 plt.xlim(x\_vals.min(), x\_vals.max())  
 plt.ylim(y\_vals.min(), y\_vals.max())  
  
 # Specify the ticks on the X and Y axes  
 plt.xticks((np.arange(int(X[:, 0].min() - 1), int(X[:, 0].max() + 1), 1.0)))  
 plt.yticks((np.arange(int(X[:, 1].min() - 1), int(X[:, 1].max() + 1), 1.0)))  
  
 plt.show()

**LR\_1\_task\_3.py**

import numpy as np  
from sklearn import linear\_model  
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)

**Результат виконання:**

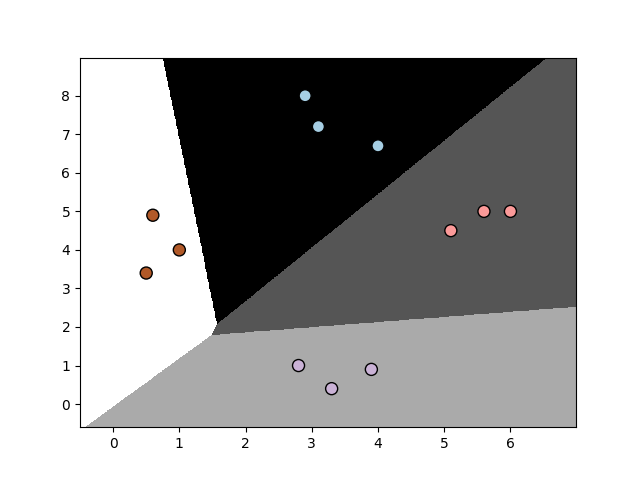
****

Рис.5 Результат виконання

**Завдання 2.4**

import numpy as np  
from utilities import visualize\_classifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.model\_selection import train\_test\_split  
  
# 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(f"Accuracy of Naive Bayes classifier = {round(accuracy,2)}%")  
  
# Візуалізація результатів роботи класифікатора  
visualize\_classifier(classifier, X, y)

**Результат виконання:**

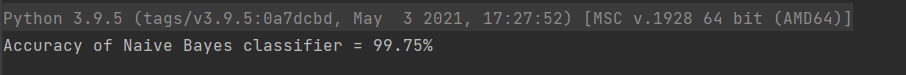
****

Рис.6 Результат виконання

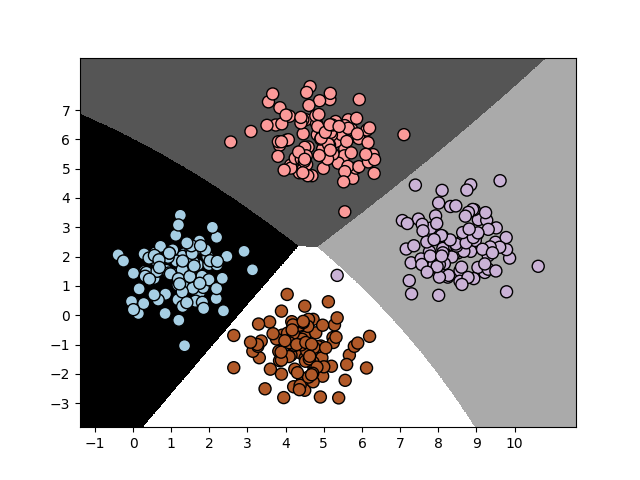


Рис.7 Результат виконання

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  
  
# 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(f"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(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(f"Accuracy of the new classifier = {round(accuracy, 2)}%")  
  
# Візуалізація роботи класифікатора  
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)) + "%")

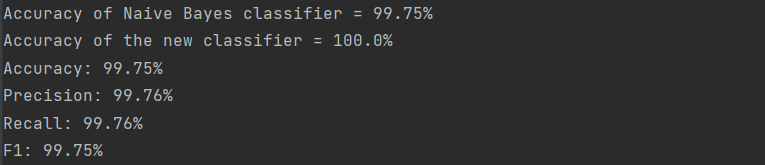


Рис.8 Результат виконання



Рис.9 Результат виконання

Після того як ми виконали перехресну перевірку та розбили дані на тестові та тренувальні точність підвищилась до 100%.

**Завдання 2.5**

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 saukh\_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]])  
  
  
saukh\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)  
  
assert np.array\_equal(saukh\_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(saukh\_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 saukh\_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 saukh\_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 saukh\_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' % (saukh\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))  
  
print(recall\_score(df.actual\_label.values, df.predicted\_RF.values))  
  
  
def saukh\_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 saukh\_recall\_score(df.actual\_label.values, df.predicted\_RF.values) == recall\_score(df.actual\_label.values,  
 df.predicted\_RF.values), 'saukh\_accuracy\_score failed on RF'  
assert saukh\_recall\_score(df.actual\_label.values, df.predicted\_LR.values) == recall\_score(df.actual\_label.values,  
 df.predicted\_LR.values), 'saukh\_accuracy\_score failed on LR'  
  
print('Recall RF: %.3f' % (saukh\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall LR: %.3f' % (saukh\_recall\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
precision\_score(df.actual\_label.values, df.predicted\_RF.values)  
  
  
def saukh\_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 saukh\_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 saukh\_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' % (saukh\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision LR: %.3f' % (saukh\_precision\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
f1\_score(df.actual\_label.values, df.predicted\_RF.values)  
  
  
def saukh\_f1\_score(y\_true, y\_pred): # calculates the F1 score  
 recall = saukh\_recall\_score(y\_true, y\_pred)  
 precision = saukh\_precision\_score(y\_true, y\_pred)  
 return 2 \* (precision \* recall) / (precision + recall)  
  
  
assert saukh\_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 saukh\_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' % (saukh\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 LR: %.3f' % (saukh\_f1\_score(df.actual\_label.values, df.predicted\_LR.values)))  
print('scores with threshold = 0.5')  
  
print('Accuracy RF: % .3f' % (saukh\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall RF: %.3f' % (saukh\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision RF: % .3f' % (saukh\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 RF: %.3f' % (saukh\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('')  
  
threshold = 0.25  
  
print(f'Scores with threshold = {threshold}')  
print('Accuracy RF: % .3f' % (saukh\_accuracy\_score(df.actual\_label.values, (df.model\_RF >= threshold).astype('int').values)))  
print('Recall RF: %.3f' % (saukh\_recall\_score(df.actual\_label.values, (df.model\_RF >= threshold).astype('int').values)))  
print('Precision RF: %.3f' % (saukh\_precision\_score(df.actual\_label.values, (df.model\_RF >= threshold).astype('int').values)))  
print('F1 RF: %.3f' % (saukh\_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**:

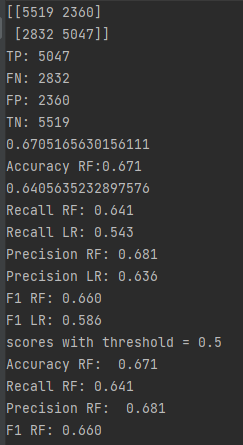
****

Рис.10 Результат виконання

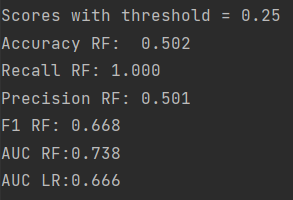
****

Рис.11 Результат виконання для порогу 0.25

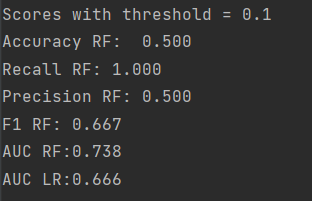


Рис.12 Результат виконання для порогу 0.10

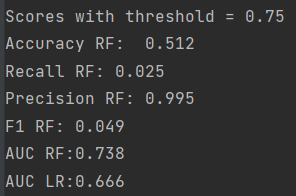


Рис.13 Результат виконання для порогу 0.75

**Висновки**: в результаті збільшення порогу, F1 міра зменшується.

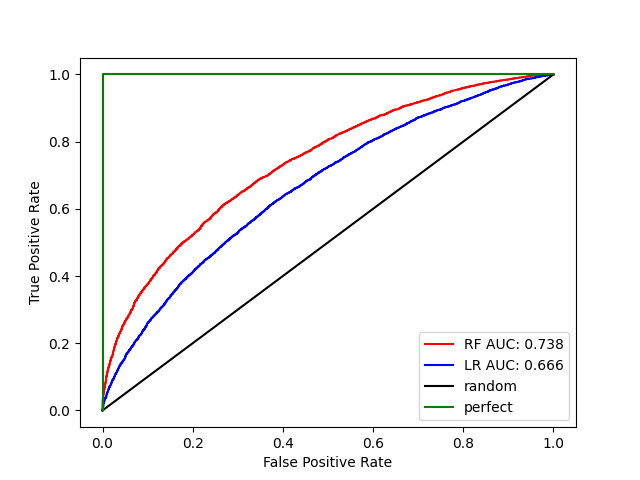


Рис.14 ROC - крива

Як бачимо з графіку RF модель має більшу точність, аніж LR модель. Звісно, що можуть бути ситуації, коли LR має переваги перед RF, але я думаю, що важливіше, що слід враховувати, — це складність моделі.

**Завдання 2.6.**

import numpy as np  
from sklearn import datasets  
from sklearn.model\_selection import train\_test\_split  
from sklearn import svm  
from sklearn import metrics  
  
# Вхідний файл, який містить дані  
from utilities import visualize\_classifier  
input\_file = 'data\_multivar\_nb.txt'  
# Завантаження даних із вхідного файлу  
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**:

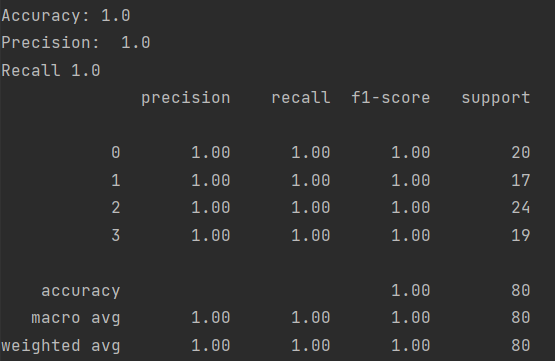


Рис. 15 Результат виконання

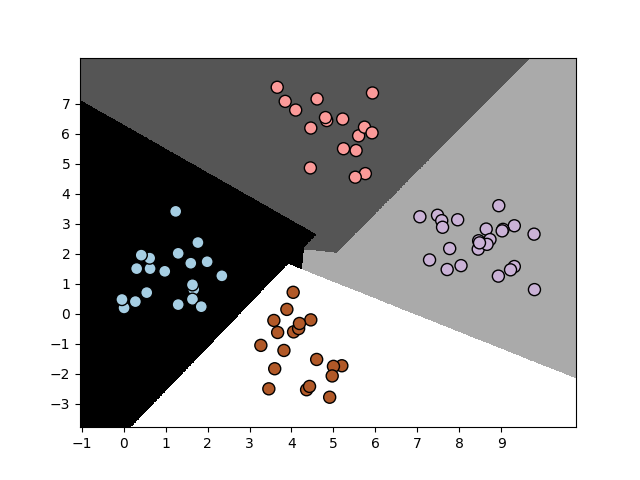


Рис. 16 Результат виконання

Результат порівняння: Наївний класифікатор байесовського і метод опорних векторів (SVM) мають різні параметри, включаючи вибір функції ядра для

кожного з них. Обидва алгоритми є дуже чутливими до оптимізації

параметрів, тобто вибір різних параметрів може суттєво змінити їхній вихід. Отже, якщо результат показує, що NBC працює краще, ніж SVM, це вірно тільки для вибраних параметрів. Тим не менш, за інших параметрів можна виявити, що SVM працює краще.