**ЛАБОРАТОРНА РОБОТА №1**

***Тема:*** ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

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

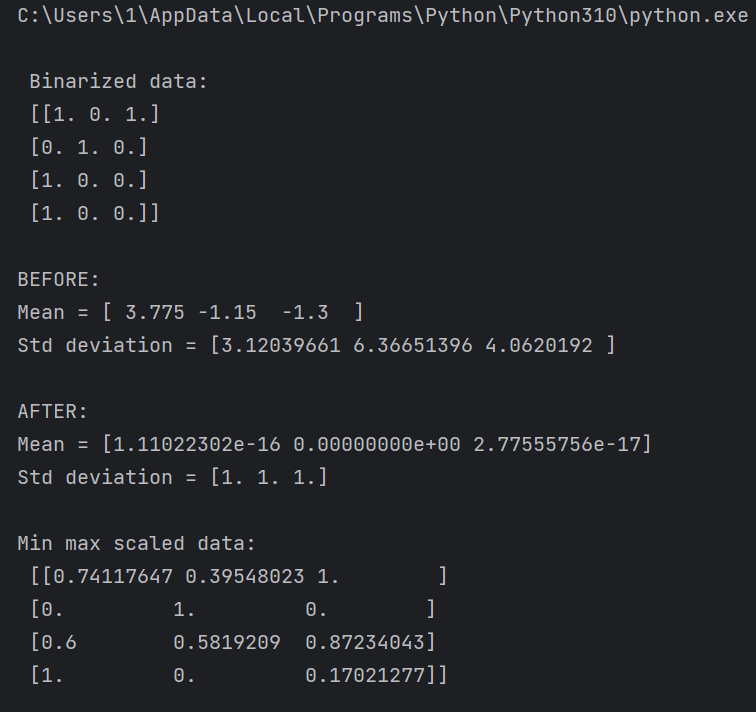
**Хід роботи:**

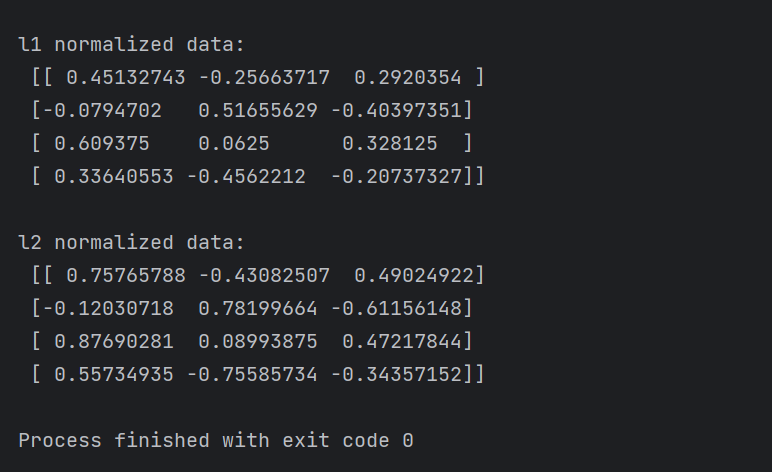
**Завдання 2.1.** Попередня обробка даних

**Лістинг програми:**

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("\n Binarized 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))  
  
# Виключення середнього  
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)

**Результат виконання програми:**

****

****

**Рис.1** Результат виконання програми

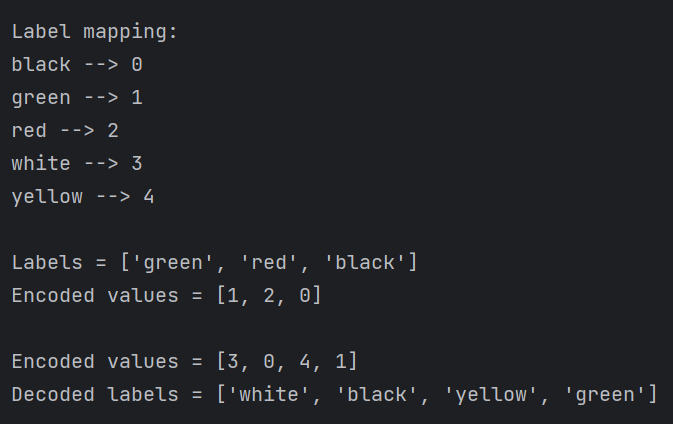
**Різниця L1-нормалізації від L2-нормалізації:**

На цих двух рівнях нормалізації використовують різні методи, наприклад, на рівні L1використовується метод найменших абсолютних відхилень, що забезпечує рівність 1 суми абсолютних значень в кожному рядку, а рівень L2 використовує метод найменших квадратів, що забезпечує рівність 1 суми квадратів значень. Взагалі, техніка *L1-нормалізації* вважається більш надійною по порівняно з *L2-нормалізацією*, оскільки вона менш чутлива до викидів.

**Лістинг програми:**

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))

**Результат виконання програми:**

****

**Рис.2** Результат виконання програми

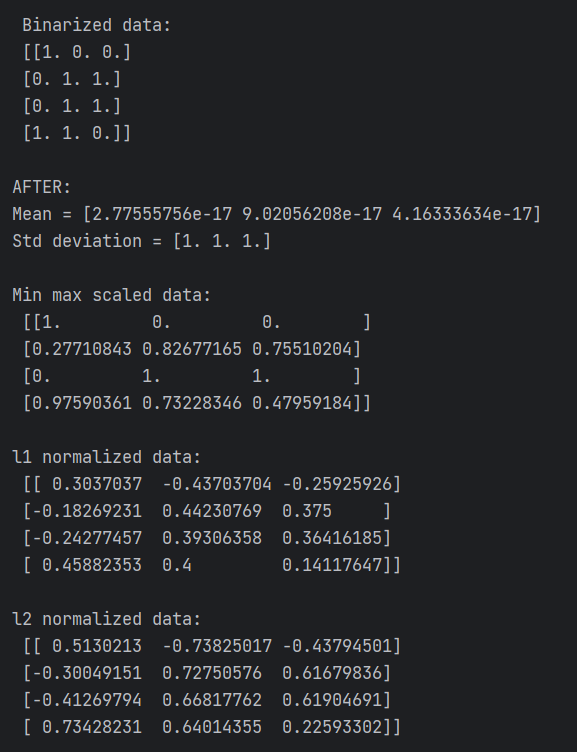
**Завдання 2.2.** Попередня обробка нових даних

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | Значення змінної  input\_data | | | | | | | | | | | | Поріг  бінаризації |
| 2. | 4.1 | -5.9 | -3.5 | -1.9 | 4.6 | 3.9 | -4.2 | 6.8 | 6.3 | 3.9 | 3.4 | 1.2 | 3.2 |

**Лістинг програми:**

import numpy as np  
from sklearn import preprocessing  
  
input\_data = np.array([[4.1, -5.9, -3.5],  
 [-1.9, 4.6, 3.9],  
 [-4.2, 6.8, 6.3],  
 [3.9, 3.4, 1.2]])  
  
# Бінаризація даних  
data\_binarized = preprocessing.Binarizer(threshold=3.2).transform(input\_data)  
print("\n Binarized data:\n", data\_binarized)  
  
# Виключення середнього  
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)

**Результат виконання програми:**

****

**Рис.3** Результат виконання програми

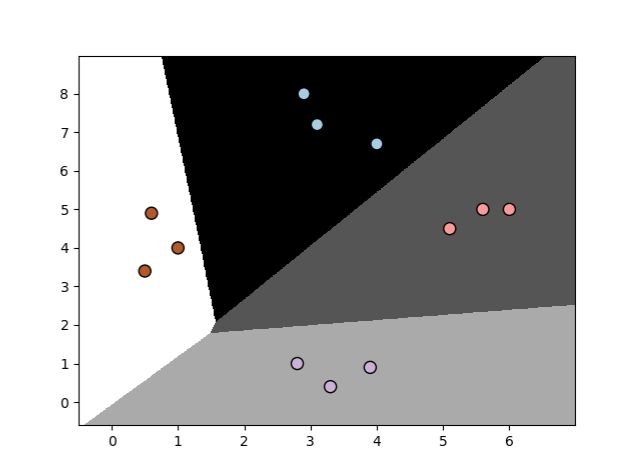
**Завдання 2.3.** Класифікація логістичною регресією або логістичний

класифікатор

**Лістинг програми:**

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)

**Результат виконання програми:**

****

**Рис.4** Результат виконання програми

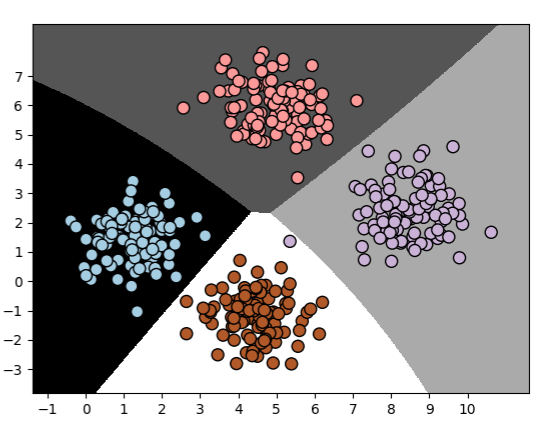
**Завдання 2.4.** Класифікація наївним байєсовським класифікатором

**Лістинг програми:**

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)

**Accuracy of Naive Bayes classifier = 99.75 %**

**Результат виконання програми:**

****

**Рис.5** Результат виконання програми

Оскільки попередній метод обчислення є не надійним, ми виконаємо перехресну перевірку та зробимо ще один прогін. Додамо до минулого коду наступне:

**Лістинг програми:**

# 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("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='recall\_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)) + "%")

**Accuracy of the new classifier = 100.0 %**

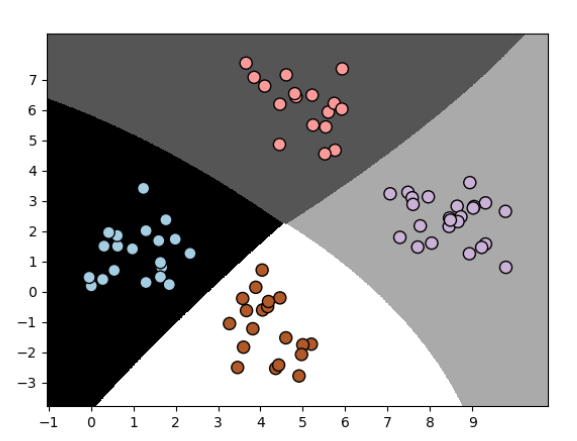
**Accuracy: 99.75%**

**Precision: 99.76%**

**Recall: 99.75%**

**F1: 99.75%**

**Результат виконання програми:**

****

**Рис.6** Результат виконання програми

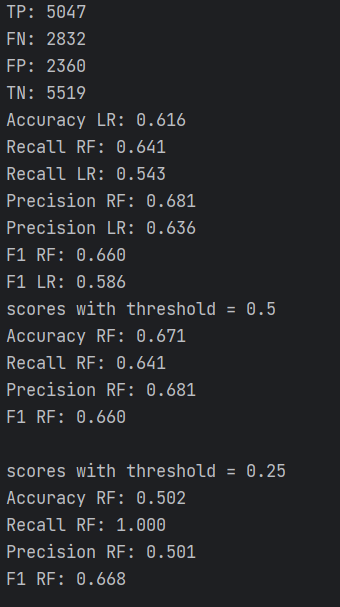
Порівнюючи два зоображення ми бачимо, що результат другої класифікації є більш коректним й точним, також має менше міток.

**Завдання 2.5.** Вивчити метрики класифікації

**Лістинг програми:**

import pandas as pd  
import numpy as np  
from sklearn.metrics import confusion\_matrix  
  
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()  
  
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()  
  
confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)  
  
def boyko\_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 boyko\_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 boyko\_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 boyko\_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:', boyko\_find\_TP(df.actual\_label.values, df.predicted\_RF.values))  
print('FN:', boyko\_find\_FN(df.actual\_label.values, df.predicted\_RF.values))  
print('FP:', boyko\_find\_FP(df.actual\_label.values, df.predicted\_RF.values))  
print('TN:', boyko\_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 = boyko\_find\_TP(y\_true,y\_pred)  
 FN = boyko\_find\_FN(y\_true,y\_pred)  
 FP = boyko\_find\_FP(y\_true,y\_pred)  
 TN = boyko\_find\_TN(y\_true,y\_pred)  
 return TP,FN,FP,TN  
def boyko\_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]])  
  
boyko\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)  
  
assert np.array\_equal(boyko\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values), confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values) ), 'boyko\_confusion\_matrix() is not correct for RF'  
assert np.array\_equal(boyko\_confusion\_matrix(df.actual\_label.values, df.predicted\_LR.values),confusion\_matrix(df.actual\_label.values, df.predicted\_LR.values) ), 'boyko\_confusion\_matrix() is not correct for LR'  
  
from sklearn.metrics import accuracy\_score  
accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)  
def boyko\_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 boyko\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values) == accuracy\_score(df.actual\_label.values, df.predicted\_RF.values), 'boyko\_accuracy\_score failed on'  
assert boyko\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values) == accuracy\_score(df.actual\_label.values, df.predicted\_LR.values), 'boyko\_accuracy\_score failed on LR'  
print('Accuracy LR: %.3f'%(boyko\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
from sklearn.metrics import recall\_score  
recall\_score(df.actual\_label.values, df.predicted\_RF.values)  
def boyko\_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 boyko\_recall\_score(df.actual\_label.values, df.predicted\_RF.values) == recall\_score(df.actual\_label.values, df.predicted\_RF.values), 'my\_accuracy\_score failed on RF'  
assert boyko\_recall\_score(df.actual\_label.values, df.predicted\_LR.values) == recall\_score(df.actual\_label.values, df.predicted\_LR.values), 'my\_accuracy\_score failed on LR'  
print('Recall RF: %.3f'%(boyko\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall LR: %.3f'%(boyko\_recall\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
from sklearn.metrics import precision\_score  
precision\_score(df.actual\_label.values, df.predicted\_RF.values)  
def boyko\_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 boyko\_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 boyko\_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'%(boyko\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision LR: %.3f'%(boyko\_precision\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
from sklearn.metrics import f1\_score  
f1\_score(df.actual\_label.values, df.predicted\_RF.values)  
def boyko\_f1\_score(y\_true, y\_pred):  
 # calculates the F1 score  
 recall = boyko\_recall\_score(y\_true,y\_pred)  
 precision = boyko\_precision\_score(y\_true,y\_pred)  
 return 2\*(precision\*recall)/(precision+recall)  
  
assert boyko\_f1\_score(df.actual\_label.values, df.predicted\_RF.values) == f1\_score(df.actual\_label.values, df.predicted\_RF.values), 'boyko\_accuracy\_score failed on RF'  
assert boyko\_f1\_score(df.actual\_label.values, df.predicted\_LR.values) == f1\_score(df.actual\_label.values, df.predicted\_LR.values), 'boyko\_accuracy\_score failed on LR'  
print('F1 RF: %.3f'%(boyko\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 LR: %.3f'%(boyko\_f1\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
  
print('scores with threshold = 0.5')  
print('Accuracy RF: %.3f'%(boyko\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall RF: %.3f'%(boyko\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision RF: %.3f'%(boyko\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 RF: %.3f'%(boyko\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('')  
print('scores with threshold = 0.25')  
print('Accuracy RF: %.3f'%(boyko\_accuracy\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('Recall RF: %.3f'%(boyko\_recall\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('Precision RF: %.3f'%(boyko\_precision\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('F1 RF: %.3f'%(boyko\_f1\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))

**Результат виконання програми:**

****

**Рис.7** Результат виконання програми

На основі поданих результатів для різних порогів можна зробити висновок:

**Поріг 0.5:**

Accuracy (точність) RF: 0.671 - Це означає, що модель правильно класифікувала 67.1% випадків.

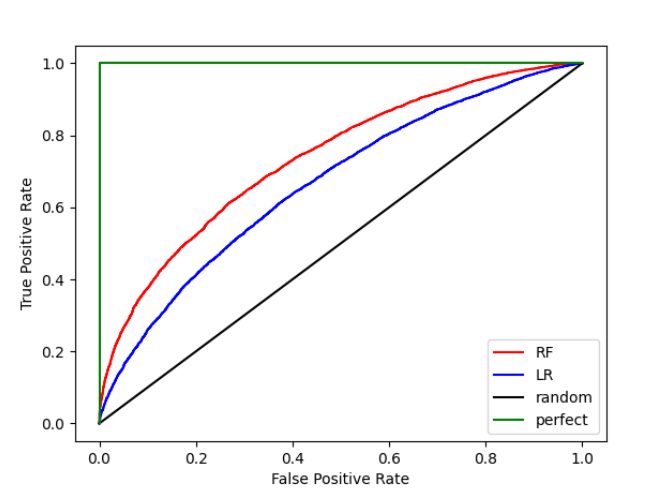
* + Recall (повнота) RF: 0.641 - Модель виявила 64.1% усіх позитивних випадків.
  + Precision (точність) RF: 0.681 - З 68.1% випадків, які модель відзначила як позитивні, вони дійсно були позитивними.
  + F1 RF: 0.660 - F1-мера об'єднує як Recall, так і Precision в одну метрику, і вона становить середнє гармонічне між ними. У вас F1 дорівнює 0.660.

**Поріг 0.25:**

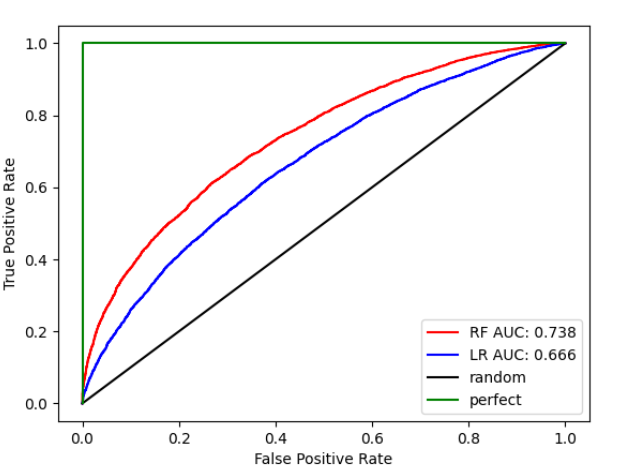
* + Accuracy (точність) RF: 0.502 - Тут модель має меншу точність, вона правильно класифікувала тільки 50.2% випадків.
  + Recall (повнота) RF: 1.000 - Модель виявила всі можливі позитивні випадки, що свідчить про те, що вона не пропускає жодного позитивного випадку.
  + Precision (точність) RF: 0.501 - Проте, точність моделі при цьому порозі дуже низька, всього 50.1% випадків, які модель відзначила як позитивні, дійсно були позитивними.
  + F1 RF: 0.668 - F1-мера знову вказує на баланс між точністю і повнотою, і в цьому випадку вона є вищою, ніж при порозі 0.5.

**Висновок:**

* За порогом 0.5 модель має кращу точність, але меншу повноту.
* За порогом 0.25 модель має високу повноту, але низьку точність.
* Вибір порогу залежить від конкретних вимог задачі. Якщо важливо уникнути пропусків позитивних випадків, може бути кращим вибором поріг 0.25. Якщо важлива точність класифікації, то краще залишити поріг на рівні 0.5.

******

**Рис.8** криві ROC для кожної моделі

******

**Рис.9** Додали AUC

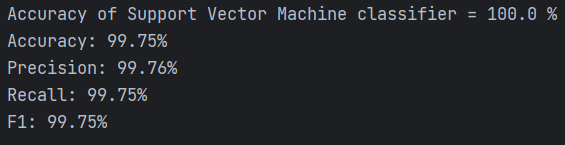
Порівняймо дві криві, а саме RF та LR і визначемо, яка з них краща. AUC вимірює площу під цими кривими й ми можемо побачити, що крива RF має більшу площу.

**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.

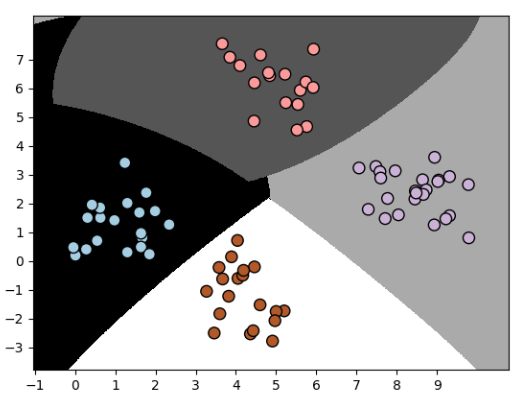
**Лістинг програми:**

import numpy as np  
from sklearn import svm  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score  
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 класифікатора машини опорних векторів  
classifier = svm.SVC()  
  
# Тренування класифікатора  
classifier.fit(X, y)  
  
# Прогнозування значень для тренувальних даних  
y\_pred = classifier.predict(X)  
  
# 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 = svm.SVC()  
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 Support Vector Machine 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='recall\_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 програми:**

****

**Рис.10** Результат виконання програми

****

**Рис.11** Результат виконання програми

Порівнюючи показники моєї програми класифікації данних з показниками наївного байєсівського класифікатора ми помітимо, що точність у байєсьвського складає **= 99.75 %,** тим часом як у моєї програми всі **100%.** Тому навіть по цьому показнику вже можна ствердужати, що данний метод переважає байєвський.

***Висновок:*** Навчився використовуючи спеціалізовані бібліотеки та мову програмування Python, також дослідив попередню обробку та класифікацію даних***.***

***Посилання на GitHub:*** https://github.com/BOYYYKO/ai