**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 №1:**

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]])  
  
# Binarize data  
data\_binarized = preprocessing.Binarizer(threshold=2.1).transform(input\_data)  
print("\nBinarized data:\n", data\_binarized)  
  
# Print mean and standard deviation  
print("\nBEFORE:")  
print("Mean =", input\_data.mean(axis=0))  
print("Std deviation =", input\_data.std(axis=0))  
  
# Remove mean  
data\_scaled = preprocessing.scale(input\_data)  
print("\nAFTER:")  
print("Mean =", data\_scaled.mean(axis=0))  
print("Std deviation =", data\_scaled.std(axis=0))  
  
# Min max scaling  
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)  
  
# Normalize data  
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:

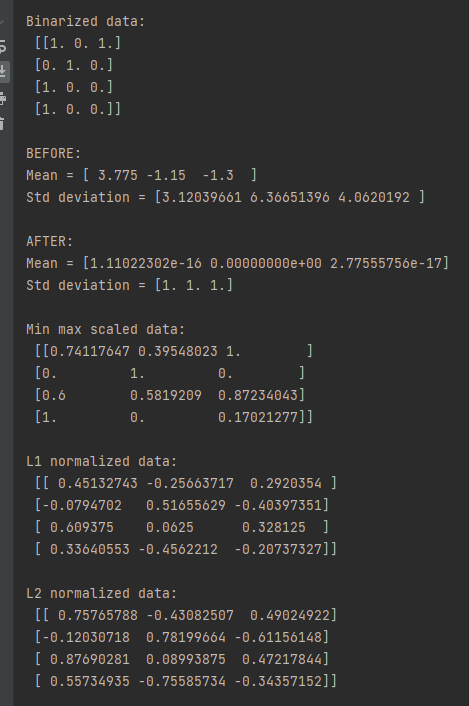


Рис. 1. Результати різних методів обробки даних

Існування двох видів нормалізації даних - L1 та L2 - є основними. У випадку L1-нормалізації, сума абсолютних значень елементів рядка визначає одиницю, тоді як у L2-нормалізації це сума квадратів цих значень. L1-нормалізація має перевагу у більшій надійності порівняно з L2, адже вона менш схильна до впливу викидів. Навпаки, L2-нормалізація використовується там, де викиди мають значний вплив.

**Завдання №2:**

Кодування міток:

import numpy as np  
from sklearn import preprocessing  
  
# Sample input labels  
input\_labels = ['red', 'black', 'red', 'green', 'black', 'yellow', 'white']  
  
# Create label encoder and fit the labels  
encoder = preprocessing.LabelEncoder()  
encoder.fit(input\_labels)  
  
# Print the mapping  
print("\nLabel mapping:")  
for i, item in enumerate(encoder.classes\_):  
 print(item, '-->', i)  
  
# Encode a set of labels using the encoder  
test\_labels = ['green', 'red', 'black']  
encoded\_values = encoder.transform(test\_labels)  
print("\nLabels =", test\_labels)  
print("Encoded values =", list(encoded\_values))  
  
# Decode a set of values using the encoder  
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:

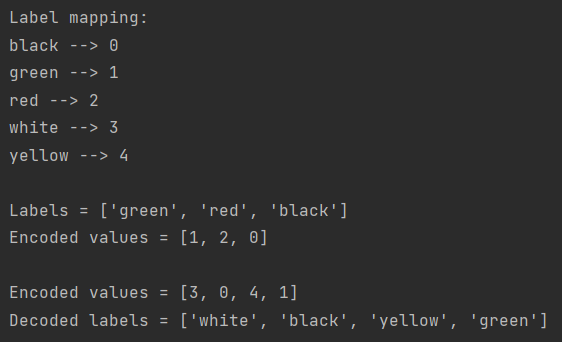


Рис. 2. Результати кодування міток

**Завдання №3:**

Змінимо дані відповідно до варіанту(№14) та здійснимо обробку даних різними методами:

import numpy as np  
from sklearn import preprocessing  
  
input\_data = np.array([[-1.3, 3.9, 6.2],  
 [-4.9, 2.2, -4.3],  
 [-2.2, 6.5, 4.1],  
 [-5.2, -3.4, -5.2]])  
  
# Binarize data  
data\_binarized = preprocessing.Binarizer(threshold=2.2).transform(input\_data)  
print("\nBinarized data:\n", data\_binarized)  
  
# Print mean and standard deviation  
print("\nBEFORE:")  
print("Mean =", input\_data.mean(axis=0))  
print("Std deviation =", input\_data.std(axis=0))  
  
# Remove mean  
data\_scaled = preprocessing.scale(input\_data)  
print("\nAFTER:")  
print("Mean =", data\_scaled.mean(axis=0))  
print("Std deviation =", data\_scaled.std(axis=0))  
  
# Min max scaling  
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)  
  
# Normalize data  
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:

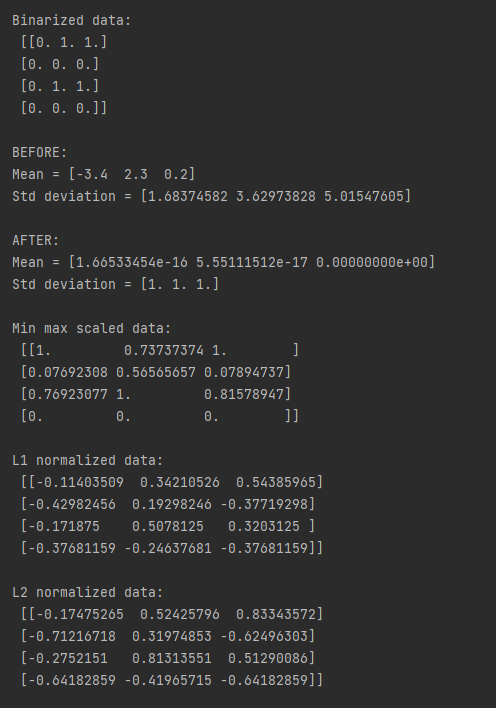


Рис. 3. Результати різних методів обробки даних відповідно до варіанту

**Завдання №4:**

Логістична регресія:

import numpy as np  
from sklearn import linear\_model  
import matplotlib.pyplot as plt  
  
from utils import visualize\_classifier  
  
# Define sample input data  
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])  
  
# Create the logistic regression classifier  
classifier = linear\_model.LogisticRegression(solver='liblinear', C=1)  
# classifier = linear\_model.LogisticRegression(solver='liblinear', C=100)  
  
# Train the classifier  
classifier.fit(X, y)  
  
# Visualize the performance of the classifier  
visualize\_classifier(classifier, X, y)

Результат:

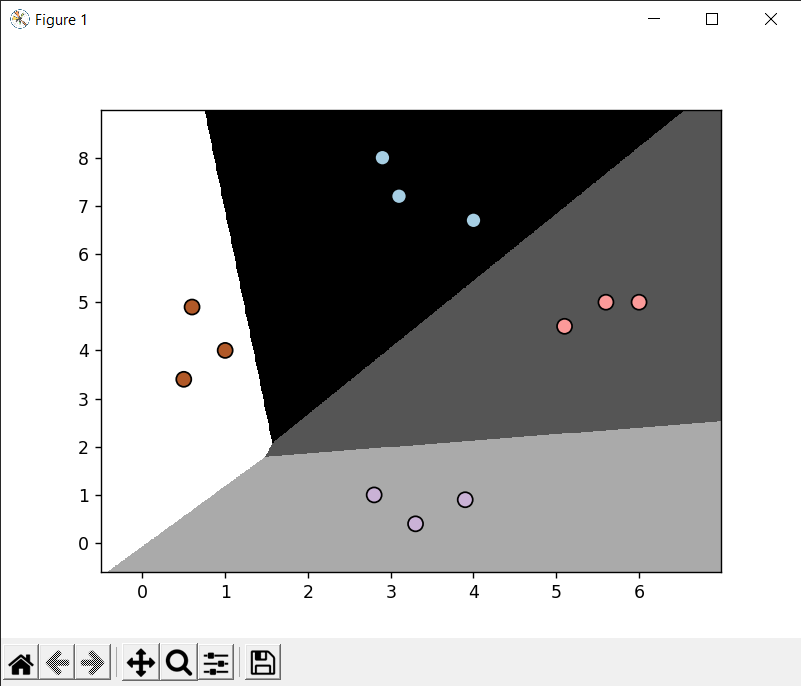


Рис. 4. Логістична регресія

**Завдання №5:**

Класифікація наївним байєсовським класифікатором:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.naive\_bayes import GaussianNB  
  
from utils import visualize\_classifier  
  
# Input file containing data  
input\_file = 'data\_multivar\_nb.txt'  
  
# Load data from input file  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
# Create Naive Bayes classifier  
classifier = GaussianNB()  
  
# Train the classifier  
classifier.fit(X, y)  
  
# Predict the values for training data  
y\_pred = classifier.predict(X)  
  
# Compute accuracy  
accuracy = 100.0 \* (y == y\_pred).sum() / X.shape[0]  
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")  
  
# Visualize the performance of the classifier  
visualize\_classifier(classifier, X, y)  
  
###############################################  
# Cross validation

Результат:

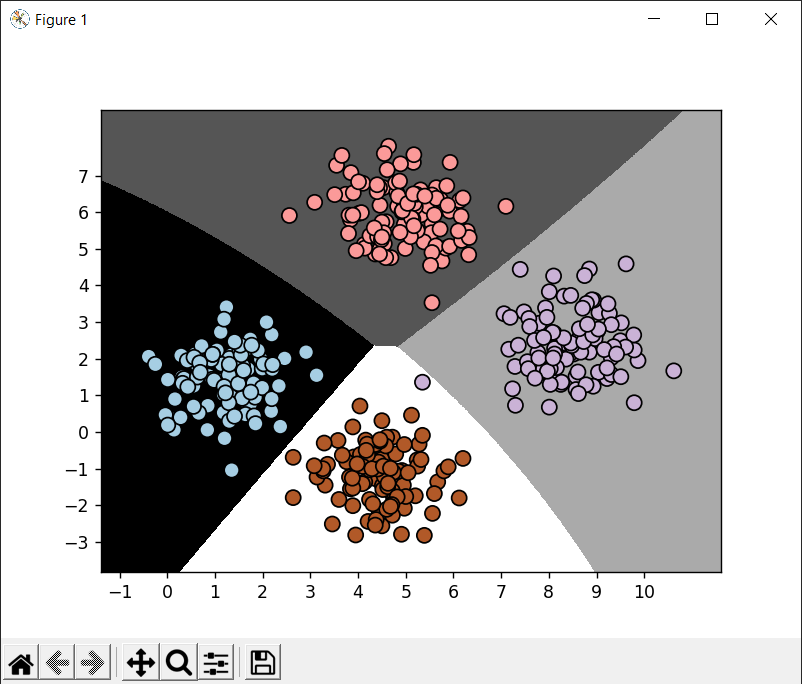


Рис. 5. Класифікація наївним байєсовським класифікатором

Точність методу 99,75%

**Завдання №6:**

Розбиття тестових даних:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.model\_selection import train\_test\_split  
  
from utils import visualize\_classifier  
  
# Input file containing data  
input\_file = 'data\_multivar\_nb.txt'  
  
# Load data from input file  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
# Create Naive Bayes classifier  
classifier = GaussianNB()  
  
# Train the classifier  
classifier.fit(X, y)  
  
# Predict the values for training data  
y\_pred = classifier.predict(X)  
  
# Compute accuracy  
accuracy = 100.0 \* (y == y\_pred).sum() / X.shape[0]  
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")  
  
# Visualize the performance of the classifier  
visualize\_classifier(classifier, X, y)  
  
###############################################  
# Cross validation  
  
# Split data into training and test data  
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)  
  
# compute accuracy of the classifier  
accuracy = 100.0 \* (y\_test == y\_test\_pred).sum() / X\_test.shape[0]  
print("Accuracy of the new classifier =", round(accuracy, 2), "%")  
  
# Visualize the performance of the classifier  
visualize\_classifier(classifier\_new, X\_test, y\_test)  
  
###############################################  
# Scoring functions  
  
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)) + "%")

Результат першого прогону:

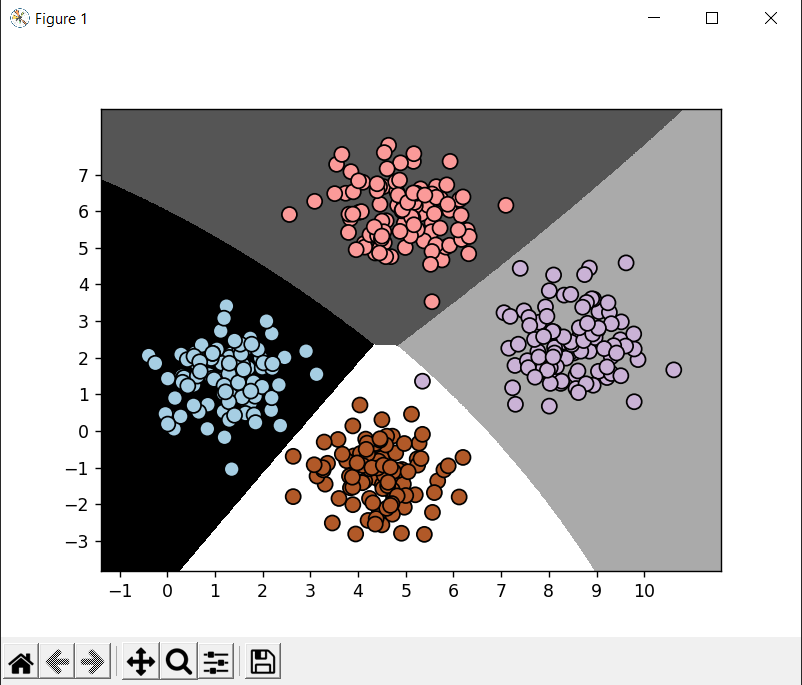


Рис. 6. Результат першого прогону класифікації наївним байєсовським класифікатором

Результат другого прогону:

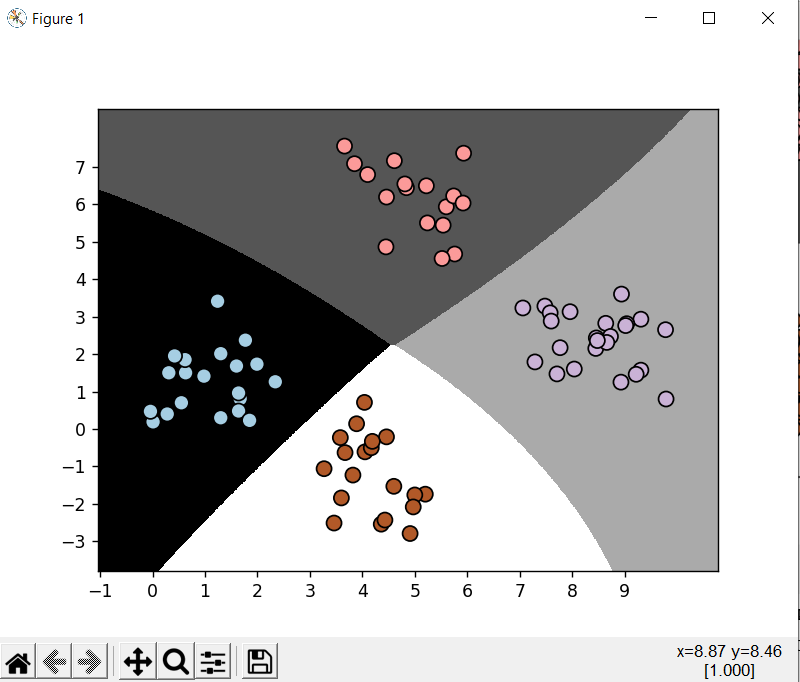


Рис. 7. Результат другого прогону класифікації наївним байєсовським класифікатором

На першому зображенні ми виявляємо некоректне категоризування, в той час як на другому зображенні такої помилки не спостерігається. Це підкріплюється статистичними даними: точність першого аналізу складає 99,75%, тоді як у другому випадку вона дорівнює 100%. Таким чином, результати другого аналізу виявилися більш ефективними порівняно з першим.

**Завдання №6:**

Власні функції для перевірки confusion\_matrix:

import pandas as pd  
  
df = pd.read\_csv('data\_metrics.csv')  
thresh = 0.5  
df['predicted\_RF'] = (df.model\_RF >= 0.5).astype('int')  
df['predicted\_LR'] = (df.model\_LR >= 0.5).astype('int')  
  
  
def hordeiev\_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 hordeiev\_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 hordeiev\_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 hordeiev\_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:', hordeiev\_find\_TP(df.actual\_label.values, df.predicted\_RF.values))  
print('FN:', hordeiev\_find\_FN(df.actual\_label.values, df.predicted\_RF.values))  
print('FP:', hordeiev\_find\_FP(df.actual\_label.values, df.predicted\_RF.values))  
print('TN:', hordeiev\_find\_TN(df.actual\_label.values, df.predicted\_RF.values))

Власна функція, яка дублює accuracy\_score:

def hordeiev\_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)  
  
  
print('Accuracy RF: %.3f'%(hordeiev\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Accuracy LR: %.3f'%(hordeiev\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values)))

Результати збіглись: RF - 0.67, LF - 0.62.

Власна функція, яка дублює recall\_score:

def hordeiev\_recall\_score(y\_true, y\_pred):  
 # calculates the fraction of samples  
 TP,FN,FP,TN = find\_conf\_matrix\_values(y\_true,y\_pred)  
 return TP / (TP + FN)  
  
  
print('Recall RF: %.3f'%(hordeiev\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall LR: %.3f'%(hordeiev\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values)))

Результати збіглись: RF - 0.641, LF - 0.616.

Власна функція, яка дублює precision\_score:

def hordeiev\_precision\_score(y\_true, y\_pred):  
 # calculates the fraction of samples  
 TP,FN,FP,TN = find\_conf\_matrix\_values(y\_true,y\_pred)  
 return TP / (TP + FP)  
  
  
print('Precision RF: %.3f'%(hordeiev\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision LR: %.3f'%(hordeiev\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values)))

Результати збіглись: RF - 0.681, LF - 0.616.

Власна функція, яка дублює f1\_score:

def hordeiev\_f1\_score(y\_true, y\_pred):  
 # calculates the fraction of samples  
 recall = hordeiev\_recall\_score(y\_true, y\_pred)  
 precision = hordeiev\_precision\_score(y\_true, y\_pred)  
 return 2 \* (precision \* recall) / (precision + recall)  
  
  
print('F1 RF: %.3f'%(hordeiev\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 LR: %.3f'%(hordeiev\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values)))

Результати збіглись: RF - 0.66, LF - 0.616.

Результати порогів 0.5 та 0.25:

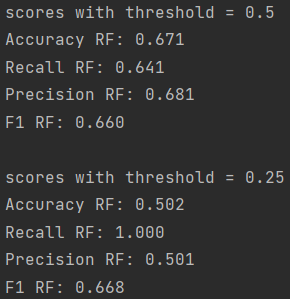


Рис. 8. Результат метрик з різними порогами

Отже, з порогом 0.5 краще такі метрики: accuracy та precision; а з порогом 0.25 – recall та f1.

Переглянемо графіки двох моделей:

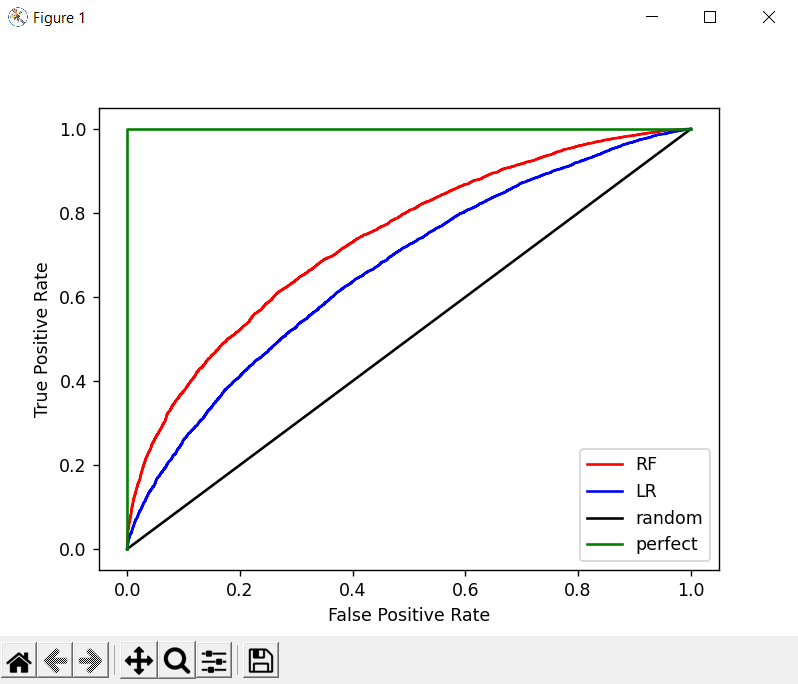


Рис. 9. ROC-криві двох моделей

На малюнку 9 спостерігаємо, що близькість до зеленої лінії означає кращі результати, тоді як наближення до чорної лінії свідчить про гірші показники. Ураховуючи, що траєкторія моделі RF ближча до зеленої лінії, можна зробити висновок, що модель RF демонструє кращі результати.

**Завдання №7:**

Розробимо програму класифікації даних за допомогою машини опорних векторів та наївного наївного байєсівського класифікатора. Також порівняємо їх показники якості:

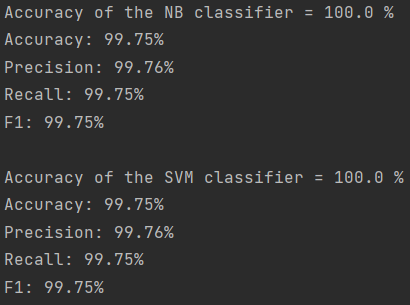
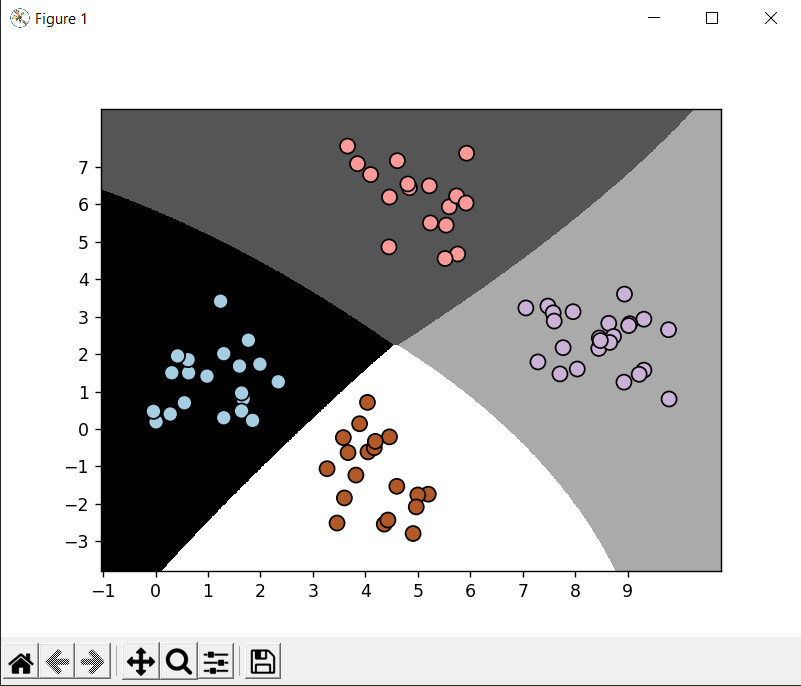


Рис. 9. Показники якості двох методів

Бачимо, що показники якості двох методів збіглись для заданих даних. Єдине, що відрізняється – графіки. Переглянемо їх:

  
Рис. 10. Графік класифікатора наївного байєсівського класифікатора

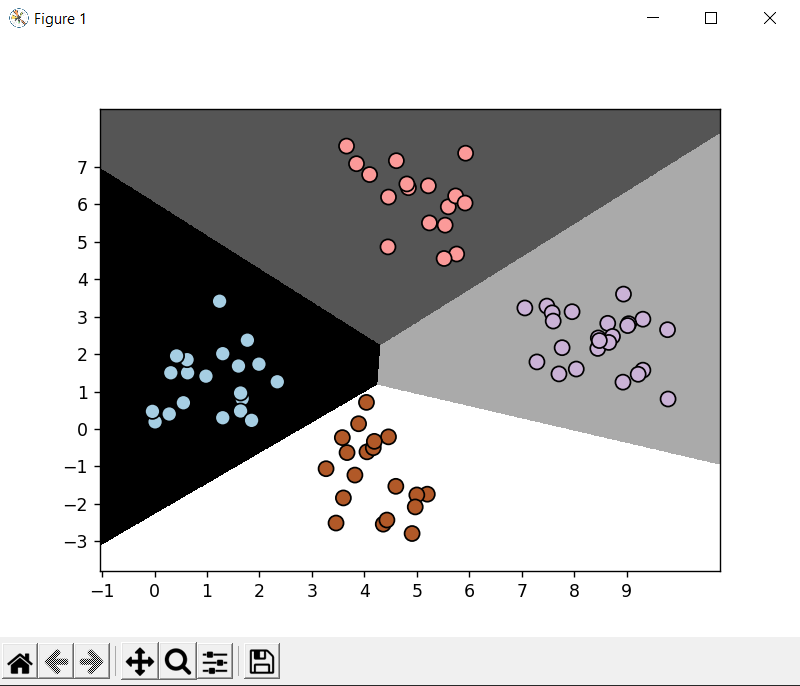


Рис. 10. Графік класифікатора за допомогою машини опорних векторів

**Висновок**: на цій лабораторній роботі я, використовуючи спеціалізовані бібліотеки та мову програмування Python, дослідив попередню обробку та класифікацію даних.