**Лабораторна робота №1**

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

***Мета роботи:*** 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]])

**Завдання 2.1.1**. Бінаризація.

Лістинг програми:

# Бінаризація даних  
data\_binarized = preprocessing.Binarizer(threshold=2.1).transform(input\_data)  
print("\n Binarized data:\n", data\_binarized)

Результат виконання програми:

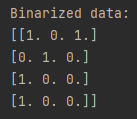


Рис. 2.1.1 – Результат виконання бінаризації.

**Завдання 2.1.2**. Виключення середнього.

Лістинг програми:

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))

Результат виконання програми:

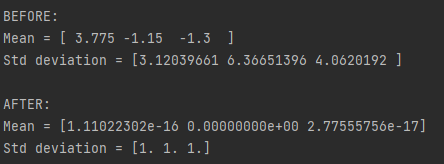


Рис. 2.1.2 – Результат виконання виключення середнього.

**Завдання 2.1.3**. Масштабування.

Лістинг програми:

# 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)

Результат виконання програми:

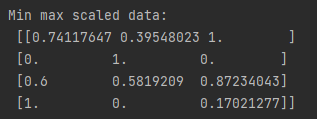
****

Рис. 2.1.3 – Результат виконання масштабування.

**Завдання 2.1.4**. Нормалізація.

Лістинг програми:

# 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:

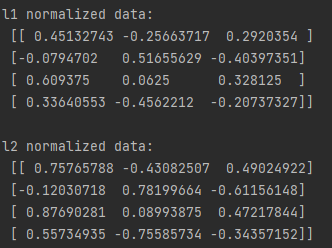


Рис. 2.1.4 – Результат виконання нормалізації.

**Чим відрізняються L1-нормалізація від L2-нормалізацієї?**

L1-нормалізація і L2-нормалізація є двома різними методами нормалізації даних, які використовуються для приведення даних до одиничної норми (норма вектора дорівнює 1). Вони відрізняються основним способом обчислення норми та впливом на дані.

Основна відмінність полягає в обчисленні: L1-норма враховує абсолютні значення компонента, тоді як L2-норма враховує їх квадрати. Це призводить до різниці впливу на великі та малі значення. L1-нормалізація може бути більш стійкою до великих викидів (outliers) через використання абсолютних значень, тоді як L2-нормалізація може бути більш чутливою до таких викидів через квадратичний вплив.

**Завдання 2.1.5**. Кодування міток.

Лістинг програми:

from sklearn import preprocessing  
  
# Надання позначок вхідних даних  
Input\_labels = ['red', 'Ыасk', 'red', 'green', 'Ьlack', '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', 'Ыасk']  
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:

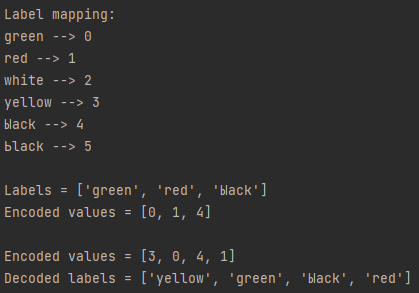


Рис. 2.1.5 – Результат виконання кодування міток.

**Завдання 2.2**. Попередня обробка нових даних.

|  |  |  |
| --- | --- | --- |
| **№ варіанту** | **Значення змінної** | **Поріг бінаризації** |
| 6 | 2.3, -1.6, 6.1, -2.4, -1.2, 4.3, 3.2, 5.5, -6.1, -4.4, 1.4, -1.2 | 2.1 |

Лістинг програми:

from sklearn import preprocessing  
  
# Надання позначок вхідних даних  
Input\_labels = [2.3, -1.6, 6.1, -2.4, -1.2, 4.3, 3.2, 5.5, -6.1, -4.4, 1.4, -1.2]  
  
# Бінарізація  
binarizer = preprocessing.Binarizer(threshold=2.1)  
input\_data\_binarized = binarizer.transform([Input\_labels])  
print("Binarized data:\n", input\_data\_binarized)  
  
# Виключення середнього  
input\_data\_mean\_removed = preprocessing.scale(Input\_labels)  
print("\nMean removed data:\n", input\_data\_mean\_removed)  
  
# Масштабування  
min\_max\_scaler = preprocessing.MinMaxScaler()  
input\_data\_scaled = min\_max\_scaler.fit\_transform([Input\_labels])  
print("\nScaled data:\n", input\_data\_scaled)  
  
# Нормалізація  
input\_data\_normalized\_l1 = preprocessing.normalize([Input\_labels], norm='l1')  
input\_data\_normalized\_l2 = preprocessing.normalize([Input\_labels], norm='l2')  
print("\nNormalized data l1:\n", input\_data\_normalized\_l1)  
print("\nNormalized data l2:\n", input\_data\_normalized\_l2)

Результат виконання:

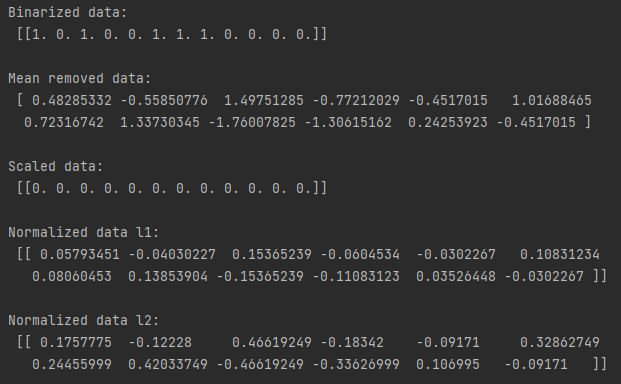


Рис. 2.2.1 – Результат виконання попередньої обробки нових даних.

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

Лістинг програми:

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:

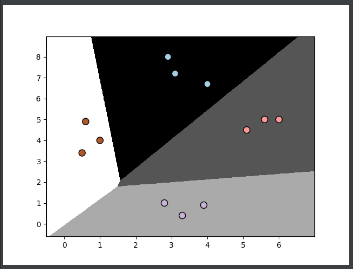


Рис. 2.3.1 – Результат виконання класифікації логістичною регресією або логістичного класифікатора.

**Завдання 2.4**. Класифікація наївним байєсовським класифікатором.

Лістинг програми:

import numpy as np  
from sklearn.naive\_bayes import GaussianNB  
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 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(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)) + "%")  
  
# Новий прогін  
# Розбивка даних на навчальний та тестовий набори  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_test, y\_test, 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)

Результат виконання:

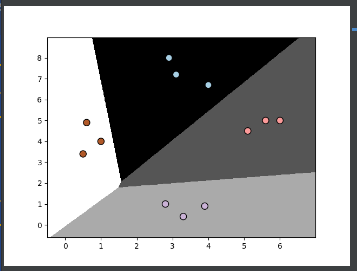


Рис. 2.4.1 – Результат виконання класифікації наївним байєсовським класифікатором (1).

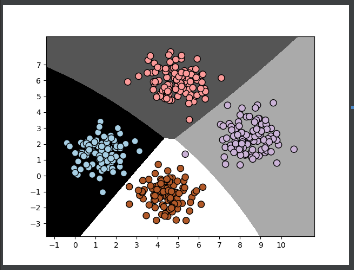


Рис. 2.4.2 – Результат виконання класифікації наївним байєсовським класифікатором (2).

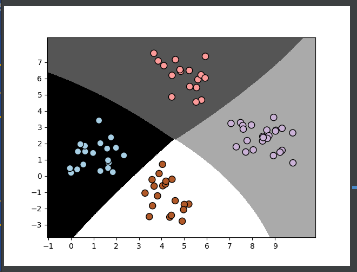


Рис. 2.4.3 – Результат виконання класифікації наївним байєсовським класифікатором (3).

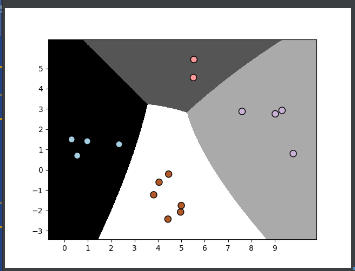


Рис. 2.4.4 – Результат виконання класифікації наївним байєсовським класифікатором (4).

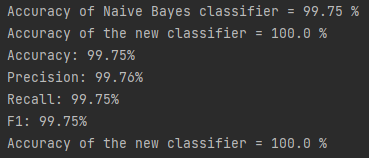


Рис. 2.4.5 – Результат виконання класифікації наївним байєсовським класифікатором (5).

***Висновок****:* Висновок полягає в тому, що розділення даних на тренувальні і тестові набори, а також використання крос-валідації допомагає уникнути перенавчання та дозволяє отримати більш надійні оцінки якості класифікатора. У даному випадку, класифікатор наївного байєса виявився придатним для задачі класифікації з прийнятною точністю на нових даних.

**Завдання 2.5**. Вивчити метрики якості класифікації.

Лістинг програми:

import pandas as pd  
import numpy as np  
from sklearn.metrics import confusion\_matrix, roc\_curve  
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\_auc\_score  
import matplotlib.pyplot as plt  
  
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()  
  
  
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))  
  
  
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):  
 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 grunytsky\_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]])  
  
  
grunytsky\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)  
  
assert np.array\_equal(grunytsky\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values),  
 confusion\_matrix(df.actual\_label.values,  
 df.predicted\_RF.values)), 'grunytsky\_confusion\_matrix() is not correct for RF'  
assert np.array\_equal(grunytsky\_confusion\_matrix(df.actual\_label.values, df.predicted\_LR.values),  
 confusion\_matrix(df.actual\_label.values,  
 df.predicted\_LR.values)), 'grunytsky\_confusion\_matrix() is not correct for LR'  
  
  
def grunytsky\_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 grunytsky\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values) == accuracy\_score(  
 df.actual\_label.values, df.predicted\_RF.values), 'grunytsky\_accuracy\_score failed on RF'  
assert grunytsky\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values) == accuracy\_score(  
 df.actual\_label.values, df.predicted\_LR.values), 'grunytsky\_accuracy\_score failed on LR'  
print('Accuracy RF: %.3f' % (grunytsky\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Accuracy LR: %.3f' % (grunytsky\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
  
def grunytsky\_recall\_score(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return TP / (TP + FN)  
  
  
assert grunytsky\_recall\_score(df.actual\_label.values, df.predicted\_RF.values) == recall\_score(df.actual\_label.values,  
 df.predicted\_RF.values), 'grunytsky\_recall\_score failed on RF'  
assert grunytsky\_recall\_score(df.actual\_label.values, df.predicted\_LR.values) == recall\_score(df.actual\_label.values,  
 df.predicted\_LR.values), 'grunytsky\_recall\_score failed on LR'  
print('Recall RF: %.3f' % (grunytsky\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall LR: %.3f' % (grunytsky\_recall\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
  
def grunytsky\_precision\_score(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return TP / (TP + FP)  
  
  
assert grunytsky\_precision\_score(df.actual\_label.values, df.predicted\_RF.values) == precision\_score(  
 df.actual\_label.values, df.predicted\_RF.values), 'grunytsky\_precision\_score failed on RF'  
assert grunytsky\_precision\_score(df.actual\_label.values, df.predicted\_LR.values) == precision\_score(  
 df.actual\_label.values, df.predicted\_LR.values), 'grunytsky\_precision\_score failed on LR'  
  
print('Precision RF: %.3f' % (grunytsky\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision LR: %.3f' % (grunytsky\_precision\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
  
def grunytsky\_f1\_score(y\_true, y\_pred):  
 recall = grunytsky\_recall\_score(y\_true, y\_pred)  
 precision = grunytsky\_precision\_score(y\_true, y\_pred)  
 return (2 \* (precision \* recall)) / (precision + recall)  
  
  
assert grunytsky\_f1\_score(df.actual\_label.values, df.predicted\_RF.values) == f1\_score(df.actual\_label.values,  
 df.predicted\_RF.values), 'grunytsky\_f1\_score failed on RF'  
assert grunytsky\_f1\_score(df.actual\_label.values, df.predicted\_LR.values) == f1\_score(df.actual\_label.values,  
 df.predicted\_LR.values), 'grunytsky\_f1\_score failed on LR'  
print('F1 RF: %.3f' % (grunytsky\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 LR: %.3f' % (grunytsky\_f1\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
print('scores with threshold = 0.5')  
print('Accuracy RF: %.3f' % (grunytsky\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall RF: %.3f' % (grunytsky\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision RF: %.3f' % (grunytsky\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 RF: %.3f' % (grunytsky\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('')  
print('scores with threshold = 0.25')  
print('Accuracy RF: %.3f' % (  
 grunytsky\_accuracy\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('Recall RF: %.3f' % (grunytsky\_recall\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('Precision RF: %.3f' % (  
 grunytsky\_precision\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('F1 RF: %.3f' % (grunytsky\_f1\_score(df.actual\_label.values, (df.model\_RF >= 0.25).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:

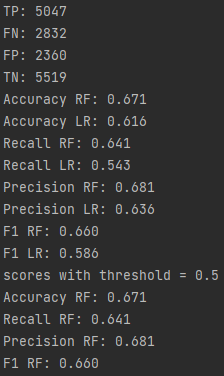


Рис. 2.5.1 – Результат виконання завдання (1).

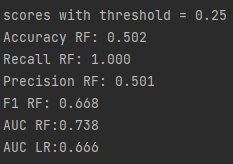


Рис. 2.5.2 – Результат виконання завдання (2).

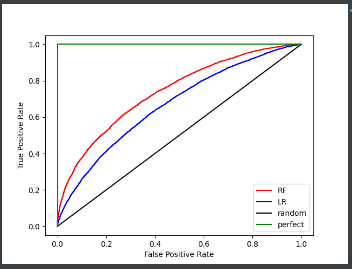


Рис. 2.5.3 – Крива ROC (1).

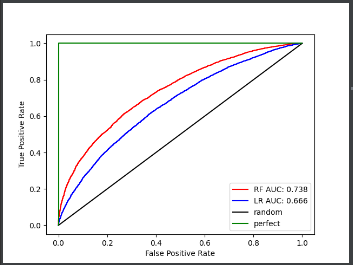


Рис. 2.5.4 – Крива ROC (2).

**Порівнявши результати для різних порогів, були зроблені такі висновки:**

* Вибір порогу відсічення має велике значення для балансу між точністю та чутливістю моделі.
* Поріг 0.5 є типовим і показує прийнятну якість класифікації для обох моделей.

**Яка з двох моделей краща (RF та LR)?**

* Оцінюючи яку з двох моделей, Random Forest (RF) або Logistic Regression (LR), краще використовувати, потрібно враховувати конкретну задачу та вимоги до моделі.

**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 pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.svm import SVC  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  
from sklearn.naive\_bayes import GaussianNB  
  
data = pd.read\_csv('data\_multivar\_nb.txt', header=None)  
  
X = data.iloc[:, :-1]  
y = data.iloc[:, -1]  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
svm\_model = SVC(kernel='linear')  
  
svm\_model.fit(X\_train, y\_train)  
  
y\_pred = svm\_model.predict(X\_test)  
  
accuracy = accuracy\_score(y\_test, y\_pred)  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
confusion = confusion\_matrix(y\_test, y\_pred)  
  
print("Accuracy:", accuracy)  
print("\nPrecision:", precision)  
print("\nRecall:", recall)  
print("\nF1 score:", f1)  
print("\nConfusion Matrix:\n", confusion)  
  
nb\_classifier = GaussianNB()  
nb\_classifier.fit(X\_train, y\_train)  
  
x\_pred = nb\_classifier.predict(X\_test)  
  
nb\_accuracy = accuracy\_score(y\_test, x\_pred)  
nb\_precision = precision\_score(y\_test, x\_pred, average='weighted')  
nb\_recall = recall\_score(y\_test, x\_pred, average='weighted')  
nb\_f1 = f1\_score(y\_test, x\_pred, average='weighted')  
nb\_confusion = confusion\_matrix(y\_test, x\_pred)  
  
print("\nIndicators of the naive Bayesian classifier")  
print("\nAccuracy:", nb\_accuracy)  
print("\nPrecision:", nb\_precision)  
print("\nRecall:", nb\_recall)  
print("\nF1 score:", nb\_f1)  
print("\nConfusion Matrix:\n", nb\_confusion)

Результат виконання:

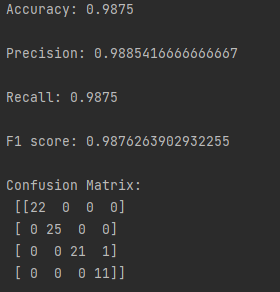


Рис. 2.6.1 – Результат виконання завдання (1).

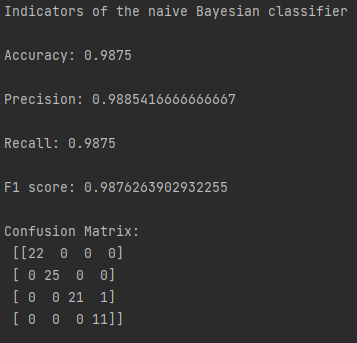


Рис. 2.6.2 – Результат виконання завдання (2).

**Висновок:** Модель Support Vector Machine (SVM) виявилася кращою для цієї задачі класифікації порівняно з наївним байєсовським класифікатором. SVM має більш високу точність та здатність до класифікації, що робить його більш підходящим для даного завдання.

***Посилання на репозиторій:*** <https://github.com/GrunytskyDmytro/Lab1_AI>

***Висновок по лабораторній роботі:*** в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив попередню обробку та класифікацію даних.