ЛАБОРАТОРНА РОБОТА № 1

Варіант 17

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

Лістинг:

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("\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("\nl1 normalized data:\n", data\_normalized\_l1)

print("\nl2 normalized data:\n", data\_normalized\_l2)

Результат виконання:

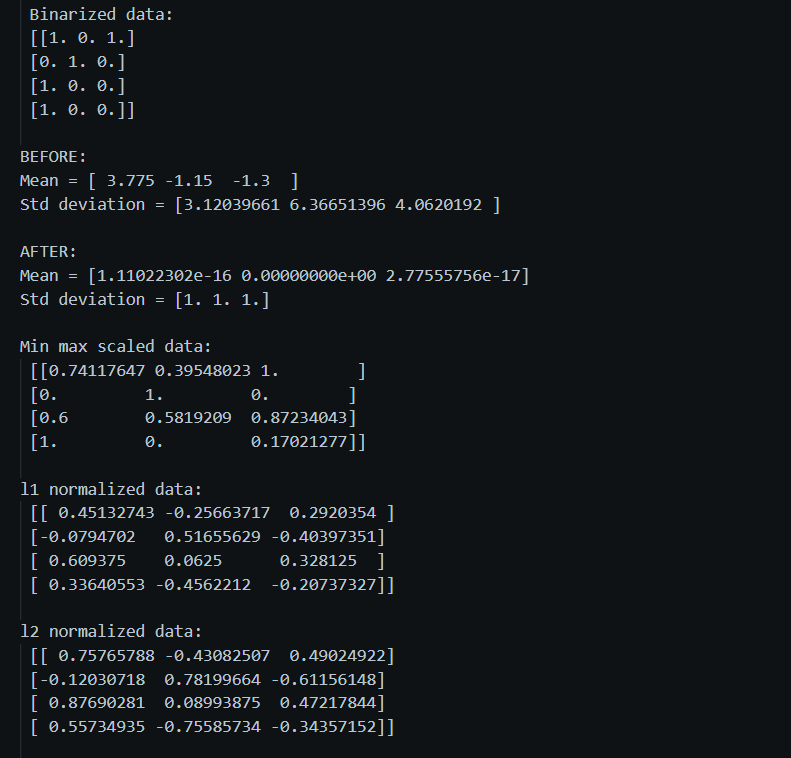


Рис. 1

Л1 нормалізація підходить для даних в яких є високий шанс викиду, оскільки вона прагне мінімізувати суму абсолютних значень коефіцієнтів. Це робить її менш чутливою до великих відхилень у порівнянні з Л2 нормалізацією, яка мінімізує квадратичну норму. При цьому Л2 нормалізаця завдяки цьому може зберігти більше ознак, також вона більш чутлива до викидів.

Загалом через свою нечутливість до викидів Л1 нормалізація вважається більш надійною.

Лістинг:

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:

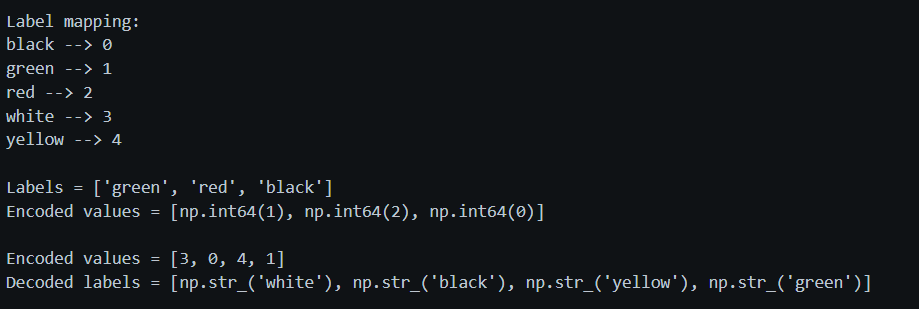
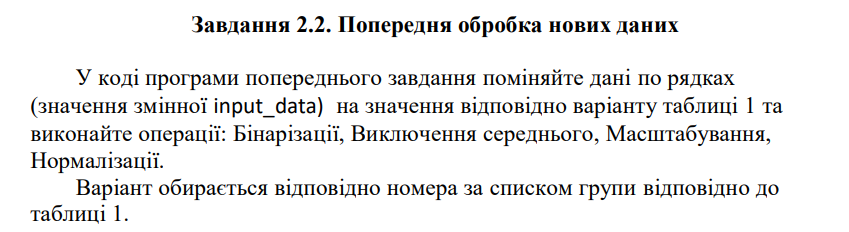
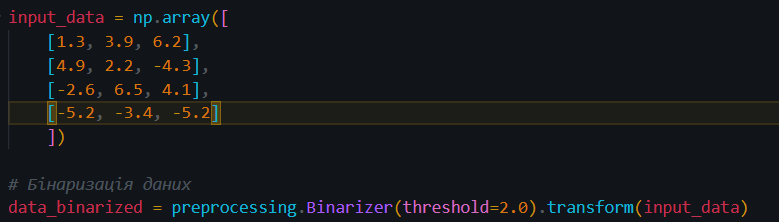


Рис 2.





Налаштування згідно варіанту:



Лістинг:

import numpy as np

from sklearn import preprocessing

input\_data = np.array([

    [1.3, 3.9, 6.2],

    [4.9, 2.2, -4.3],

    [-2.6, 6.5, 4.1],

    [-5.2, -3.4, -5.2]

    ])

*# Бінаризація даних*

data\_binarized = preprocessing.Binarizer(threshold=2.0).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("\nMin max scaled data:\n", data\_scaled\_minmax)

*# Нормалізація даних*

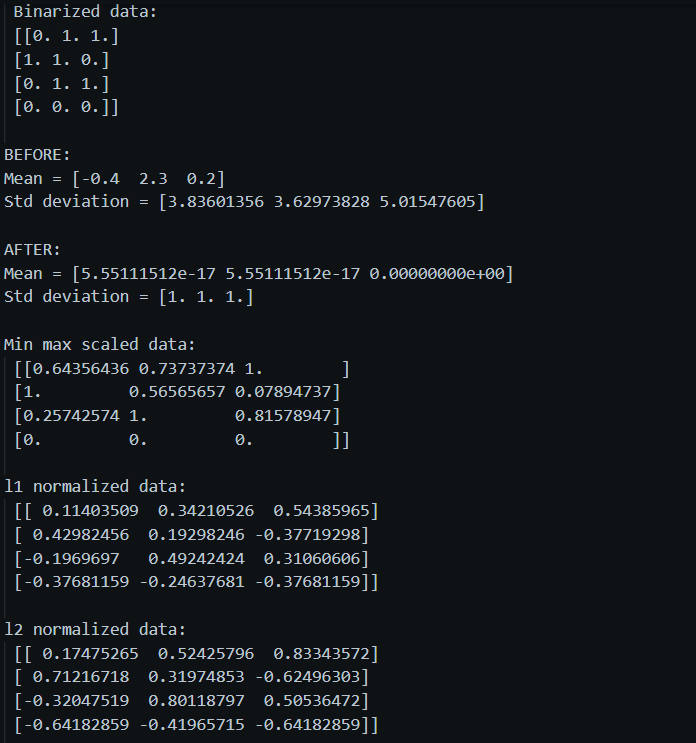
data\_normalized\_l1 = preprocessing.normalize(input\_data, norm='l1')

data\_normalized\_l2 = preprocessing.normalize(input\_data, norm='l2')

print("\nl1 normalized data:\n", data\_normalized\_l1)

print("\nl2 normalized data:\n", data\_normalized\_l2)

Результат виконання:

  
Рис. 3

Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор

Лістинг:

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:

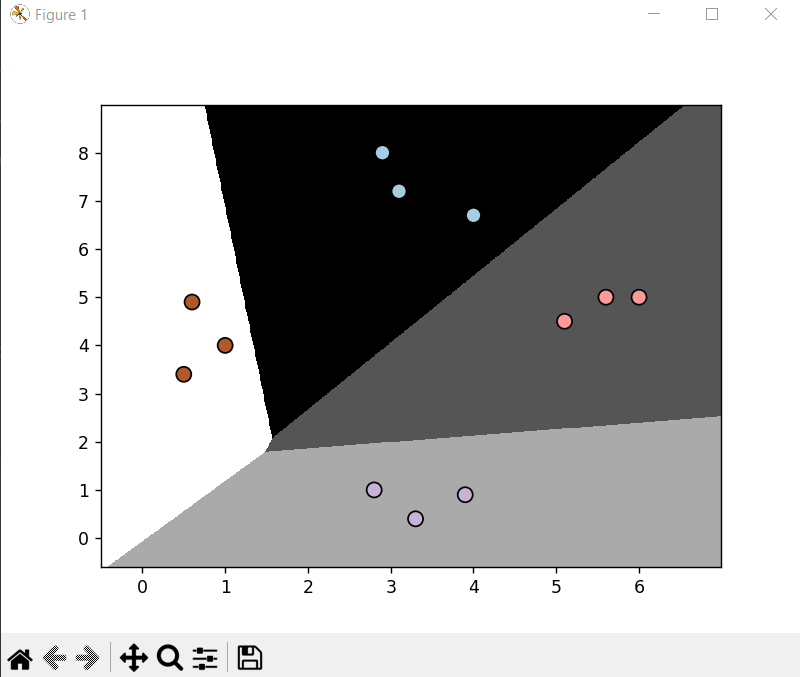


Рис. 4

Завдання 2.4. Класифікація наївним байєсовським класифікатором

Лістинг:

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)

Результат виконання:



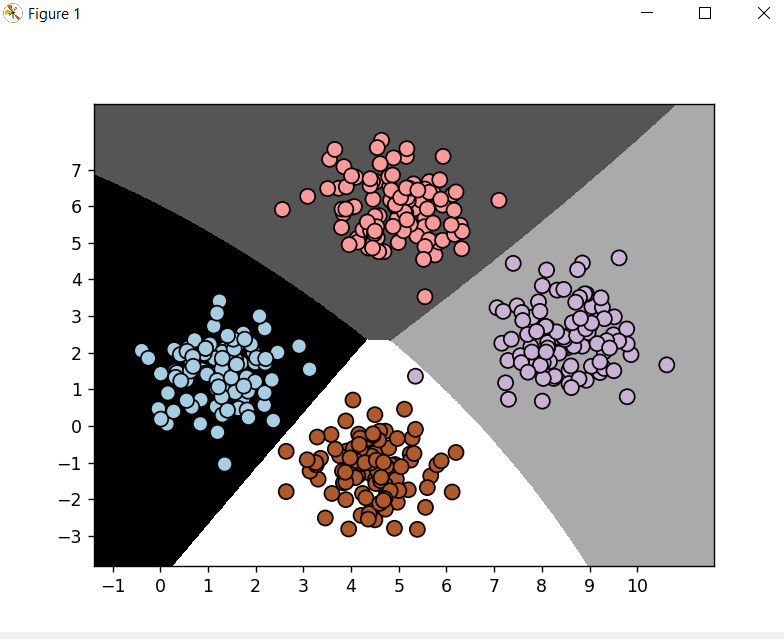


Рис. 5

Змінемо код: Розіб'ємо дані на навчальний та тестовий набори. Відповідно до значення параметра test\_size, зазначеного в рядку коду нижче, ми віднесемо 80% даних до тренування, а 20% - до тестування. Потім ми виконаємо тренування наївним байєсовським класифікатором на цих даних.

Лістинг:

import numpy as np

import matplotlib.pyplot as plt

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)) + "%")

Результат виконання:

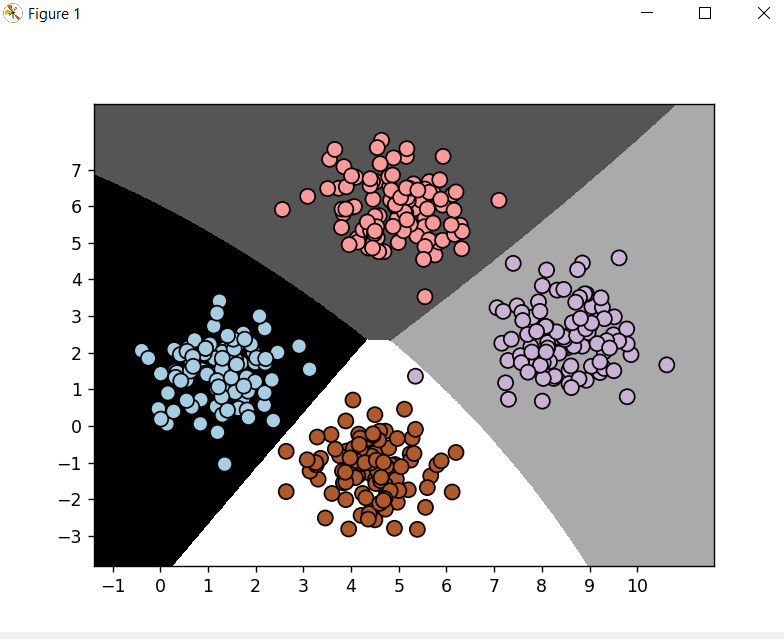
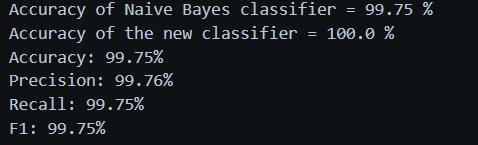


Рис. 6



Результат виконання:



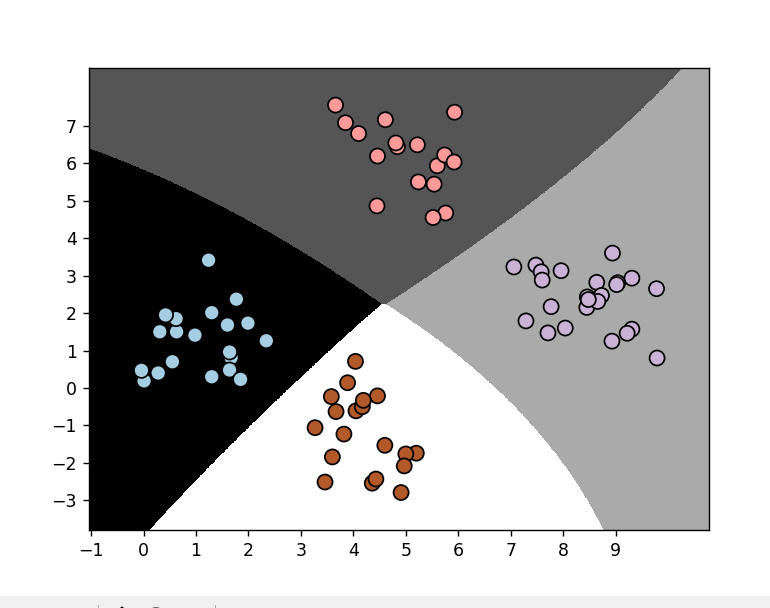


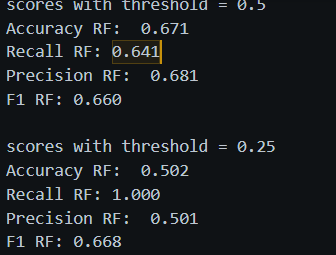
Рис. 7

На повному наборі даних межі будуть чіткішими, оскільки класифікатор "бачить" усі дані, в той час як на тестових даних межі можуть бути менш точними, оскільки вони не були використані для тренування.

При цьому на рисунку 6 видно, що одна точка вийшла за межі своєї класифікаційної області. У цьому випадку варто звернути увагу на те, чи є ця точка винятковою, чи це може бути сигналом для покращення моделі (наприклад, шляхом використання більшого об'єму даних або інших методів класифікації).

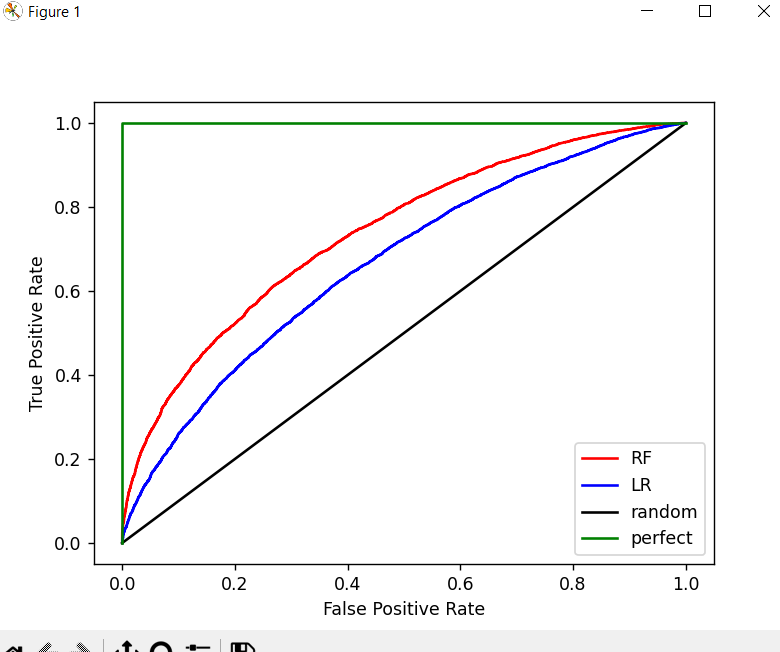
Завдання 2.5. Вивчити метрики якості класифікації





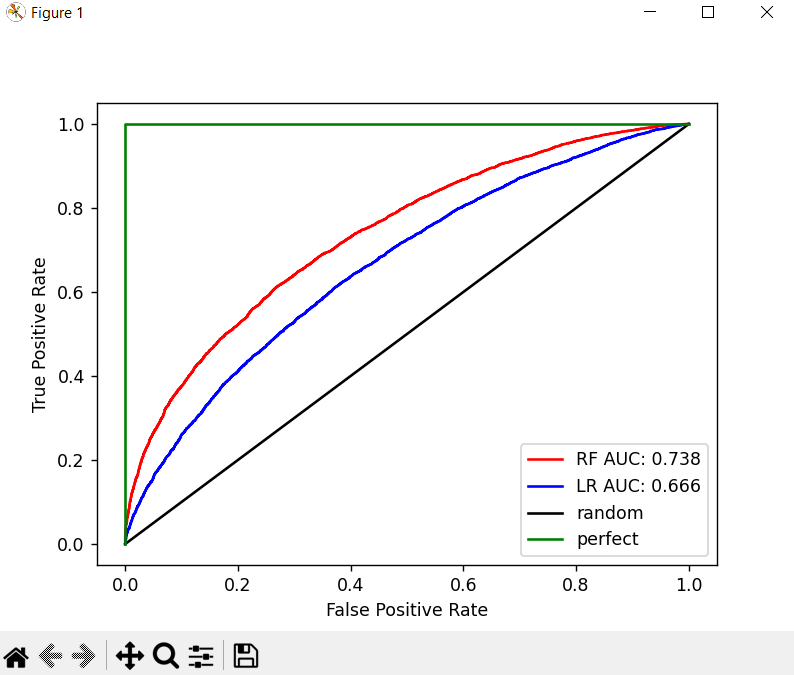
Зниження порогу до 0.25 може підвищити чутливість (recall), але призводить до зниження точності (precision) та загальної точності (accuracy). Це може бути корисно в ситуаціях, де важливіше виявлення всіх позитивних випадків, наприклад, у медичних тестах, але призводить до збільшення хибних позитивів.

Побудуйте криву ROC для кожної моделі:



Для аналізу продуктивності ми використовуватимемо метрику площі під кривою.

Результат:



Зробіть висновки яка з двох моделей (RF та LR) краща і чому. Висновки занесіть у звіт:

Думаю що вибір моделі має ґрунтуватися на конкретних вимогах задачі. Якщо важливі точність і стійкість до викидів, краще обрати RF. Якщо ж потрібно зосередитися на простоті та зрозумілості, LR може бути більш доречною. Тобто на мою думку немає об'єктивно кращої моделі, кожна використовуєтсья для свєї цілі.

Лістинг:

from sklearn.metrics import f1\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

import pandas as pd

import numpy as np

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("Matrix sklearn.metrics:", confusion\_matrix(

    df.actual\_label.values, df.predicted\_RF.values))

def find\_TP(y\_true, y\_pred):

*# Кількість істинних позитивних (y\_true = 1, y\_pred = 1)*

    return sum((y\_true == 1) & (y\_pred == 1))

def find\_FN(y\_true, y\_pred):

*# Кількість хибних негативних (y\_true = 1, y\_pred = 0)*

    return sum((y\_true == 1) & (y\_pred == 0))

def find\_FP(y\_true, y\_pred):

*# Кількість хибних позитивних (y\_true = 0, y\_pred = 1)*

    return sum((y\_true == 0) & (y\_pred == 1))

def find\_TN(y\_true, y\_pred):

*# Кількість істинних негативних (y\_true = 0, y\_pred = 0)*

    return sum((y\_true == 0) & (y\_pred == 0))

*# Функція для обчислення усіх 4 значень*

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

*# Функція для створення матриці плутанини*

def pavlenko\_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]])

print("Matrix Pavlenko:", pavlenko\_confusion\_matrix(

    df.actual\_label.values, df.predicted\_RF.values))

print("------------------------------------------------")

assert np.array\_equal(pavlenko\_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(pavlenko\_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 RF sklearn.metrics:", accuracy\_score(

    df.actual\_label.values, df.predicted\_RF.values))

print("accuracy\_score LR sklearn.metrics:", accuracy\_score(

    df.actual\_label.values, df.predicted\_LR.values))

print("")

def pavlenko\_accuracy\_score(y\_true, y\_pred):

    TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)

    accuracy = (TP + TN) / (TP + TN + FP + FN)

    return accuracy

*# Перевірка точності для моделі RF*

assert pavlenko\_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'

*# Перевірка точності для моделі LR*

assert pavlenko\_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'

*# Вивід точності*

print('Accuracy RF Pavlenko:', pavlenko\_accuracy\_score(

    df.actual\_label.values, df.predicted\_RF.values))

print('Accuracy LR Pavlenko:', pavlenko\_accuracy\_score(

    df.actual\_label.values, df.predicted\_LR.values))

print("------------------------------------------------")

print("recall\_score RF sklearn.metrics:", recall\_score(

    df.actual\_label.values, df.predicted\_RF.values))

print("recall\_score LR sklearn.metrics:", recall\_score(

    df.actual\_label.values, df.predicted\_LR.values))

print("")

def pavlenko\_recall\_score(y\_true, y\_pred):

    TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)

    recall = TP / (TP + FN) if (TP + FN) > 0 else 0  *# Уникнути ділення на нуль*

    return recall

*# Перевірка recall для моделі RF*

assert pavlenko\_recall\_score(df.actual\_label.values, df.predicted\_RF.values) == recall\_score(

    df.actual\_label.values, df.predicted\_RF.values), 'my\_recall\_score failed on RF'

*# Перевірка recall для моделі LR*

assert pavlenko\_recall\_score(df.actual\_label.values, df.predicted\_LR.values) == recall\_score(

    df.actual\_label.values, df.predicted\_LR.values), 'my\_recall\_score failed on LR'

*# Вивід recall*

print('Recall RF Pavlenko: ', pavlenko\_recall\_score(

    df.actual\_label.values, df.predicted\_RF.values))

print('Recall LR Pavlenko: ', pavlenko\_recall\_score(

    df.actual\_label.values, df.predicted\_LR.values))

print("------------------------------------------------")

print("precision\_score RF sklearn.metrics:", precision\_score(

    df.actual\_label.values, df.predicted\_RF.values))

print("precision\_score LR sklearn.metrics:", precision\_score(

    df.actual\_label.values, df.predicted\_LR.values))

print("")

def pavlenko\_precision\_score(y\_true, y\_pred):

    TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)

*# Уникнути ділення на нуль*

    precision = TP / (TP + FP) if (TP + FP) > 0 else 0

    return precision

*# Перевірка precision для моделі RF*

assert pavlenko\_precision\_score(df.actual\_label.values, df.predicted\_RF.values) == precision\_score(

    df.actual\_label.values, df.predicted\_RF.values), 'my\_precision\_score failed on RF'

*# Перевірка precision для моделі LR*

assert pavlenko\_precision\_score(df.actual\_label.values, df.predicted\_LR.values) == precision\_score(

    df.actual\_label.values, df.predicted\_LR.values), 'my\_precision\_score failed on LR'

*# Вивід precision*

print('Precision RF Pavlenko: ', pavlenko\_precision\_score(

    df.actual\_label.values, df.predicted\_RF.values))

print('Precision LR Pavlenko: ', pavlenko\_precision\_score(

    df.actual\_label.values, df.predicted\_LR.values))

print("------------------------------------------------")

print("f1\_score RF sklearn.metrics:", f1\_score(

    df.actual\_label.values, df.predicted\_RF.values))

print("f1\_score LR sklearn.metrics:", f1\_score(

    df.actual\_label.values, df.predicted\_LR.values))

print("")

def pavlenko\_f1\_score(y\_true, y\_pred):

    recall = pavlenko\_recall\_score(y\_true, y\_pred)

    precision = pavlenko\_precision\_score(y\_true, y\_pred)

    f1 = (2 \* precision \* recall) / (precision + recall) if (precision +

          recall) > 0 else 0  *# Уникнути ділення на нуль*

    return f1

*# Перевірка F1 для моделі RF*

assert pavlenko\_f1\_score(df.actual\_label.values, df.predicted\_RF.values) == f1\_score(

    df.actual\_label.values, df.predicted\_RF.values), 'my\_f1\_score failed on RF'

*# Перевірка F1 для моделі LR*

*# assert pavlenko\_f1\_score(df.actual\_label.values, df.predicted\_LR.values) == f1\_score(df.actual\_label.values, df.predicted\_LR.values), 'my\_f1\_score failed on LR'*

*# Вивід F1*

print('F1 RF Pavlenko: ', pavlenko\_f1\_score(

    df.actual\_label.values, df.predicted\_RF.values))

print('F1 LR Pavlenko: ', pavlenko\_f1\_score(

    df.actual\_label.values, df.predicted\_LR.values))

f1\_my = pavlenko\_f1\_score(df.actual\_label.values, df.predicted\_LR.values)

f1\_sklearn = f1\_score(df.actual\_label.values, df.predicted\_LR.values)

print(f'My F1: {f1\_my}, Sklearn F1: {f1\_sklearn}')

print("------------------------------------------------")

print('scores with threshold = 0.5')

print('Accuracy RF: % .3f'% (pavlenko\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('Recall RF: %.3f' % (pavlenko\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('Precision RF: % .3f'% (pavlenko\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('F1 RF: %.3f' % (pavlenko\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))

print('')

print('scores with threshold = 0.25')

print('Accuracy RF: % .3f'% (pavlenko\_accuracy\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))

print('Recall RF: %.3f' % (pavlenko\_recall\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))

print('Precision RF: % .3f'% (pavlenko\_precision\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))

print('F1 RF: %.3f' % (pavlenko\_f1\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))

print("------------------------------------------------")

from sklearn.metrics import roc\_curve

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)

import matplotlib.pyplot as plt

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()

from sklearn.metrics import roc\_auc\_score

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')

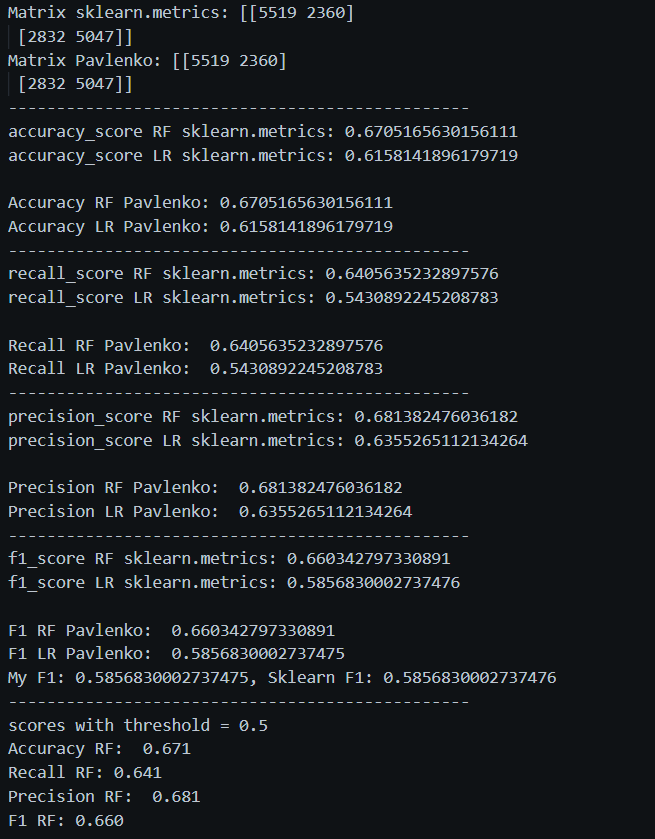
plt.legend()

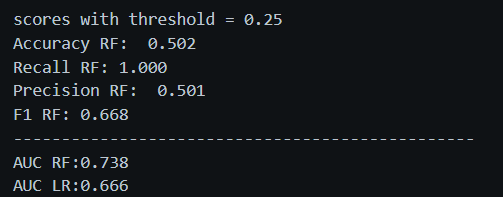
plt.xlabel('False Positive Rate')

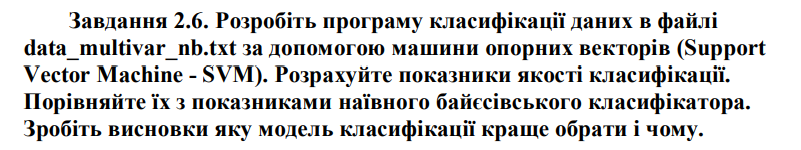
plt.ylabel('True Positive Rate')

plt.show()

Результат виконання (консоль):







Лістинг:

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

*# 1. Load the data*

data = pd.read\_csv('data\_multivar\_nb.txt', header=None)

X = data.iloc[:, :-1].values  *# Features*

y = data.iloc[:, -1].values  *# Class labels*

*# 2. Split the data into training and test sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

*# 3. Create and train models*

*# Support Vector Machine (SVM)*

svm\_model = SVC()

svm\_model.fit(X\_train, y\_train)

*# Naive Bayes Classifier (Gaussian Naive Bayes)*

nb\_model = GaussianNB()

nb\_model.fit(X\_train, y\_train)

*# 4. Make predictions on the test data*

y\_pred\_svm = svm\_model.predict(X\_test)

y\_pred\_nb = nb\_model.predict(X\_test)

*# 5. Evaluate classification performance*

*# Function to print the metrics*

def print\_metrics(y\_true, y\_pred, model\_name):

    print(f"Results for model: {model\_name}")

    print(f"Accuracy: {accuracy\_score(y\_true, y\_pred):.3f}")

    print(f"Precision: {precision\_score(y\_true, y\_pred, average='weighted'):.3f}")

    print(f"Recall: {recall\_score(y\_true, y\_pred, average='weighted'):.3f}")

    print(f"F1-Score: {f1\_score(y\_true, y\_pred, average='weighted'):.3f}")

    print(f"Confusion Matrix:\n {confusion\_matrix(y\_true, y\_pred)}\n")

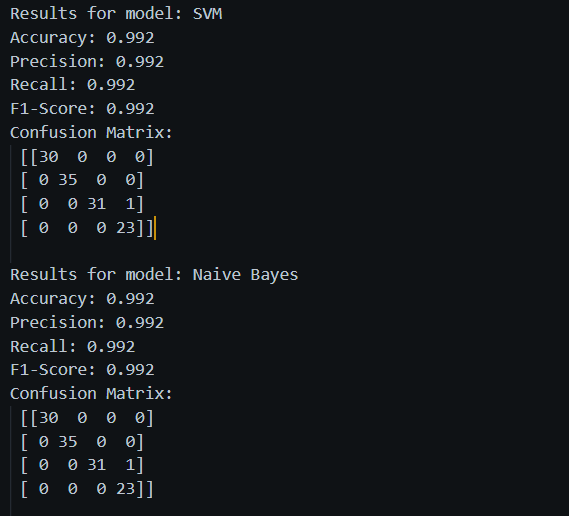
*# Evaluation for SVM*

print\_metrics(y\_test, y\_pred\_svm, "SVM")

*# Evaluation for Naive Bayes*

print\_metrics(y\_test, y\_pred\_nb, "Naive Bayes")

Результат виконання:



Обидві моделі показують однакові результати за всіма метриками: точність, повнота, F1-оцінка та матриця плутанини ідентичні. Це свідчить про те, що для даного набору даних як **SVM**, так і **Наївний Байєс** забезпечують однаково високу якість класифікації.

Git: https://github.com/PavlenkoOks/AI