Міністерство освіти і науки України

Державний університет ,,Житомирська політехніка”

Кафедра: Фікт

Група: ІПЗ-20-1

# Лабораторна робота №1

«ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ

ДАНИХ»

Виконав: Денисюк Н. Г.

Прийняв: Голенко М. Ю.

**Мета роботи:** використовуючи спеціалізовані бібліотеки та мову

програмування Python дослідити попередню обробку та класифікацію даних

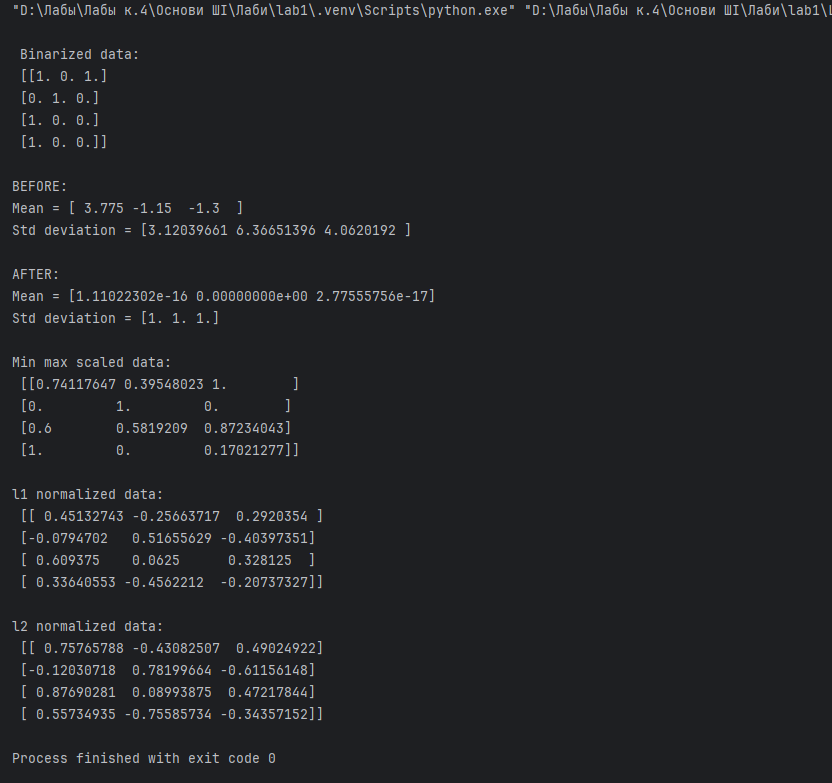
**Виконання роботи:**

**Завдання №2.1:** Попередня обробка даних

Код:

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))  
  
# Исключение среднего  
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:**



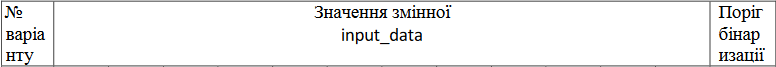
**Завдання №2.1.5:** Кодування мітокКод:

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:**

****

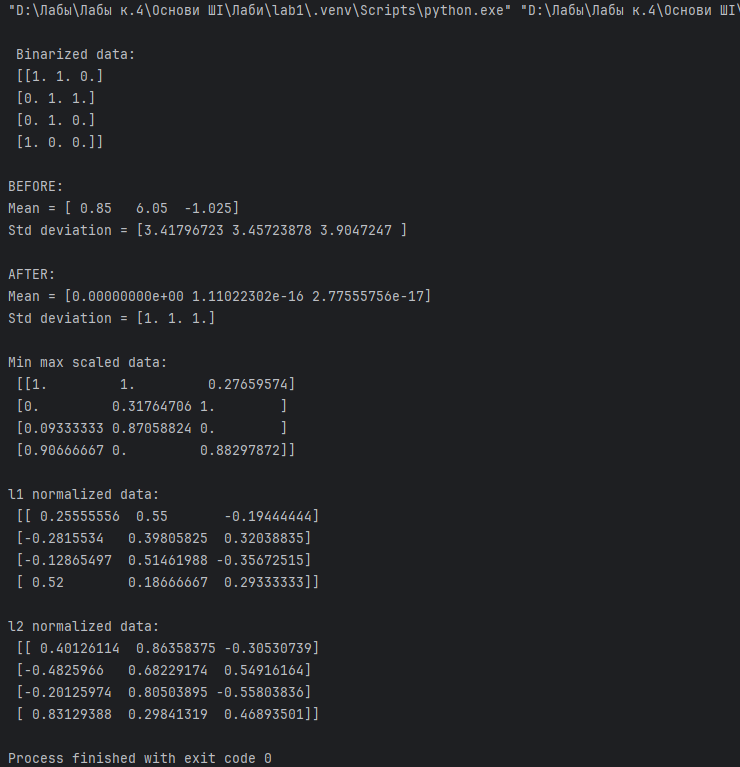
**Завдання №2.2:** Попередня обробка нових даних

Код:

import numpy as np  
from sklearn import preprocessing  
  
input\_data = np.array([[4.6, 9.9, -3.5], [-2.9, 4.1, 3.3],  
 [-2.2, 8.8, -6.1], [3.9, 1.4, 2.2]])  
  
# Бінаризація даних  
data\_binarized = preprocessing.Binarizer(threshold=2.2).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))  
  
# Исключение среднего  
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:**

****

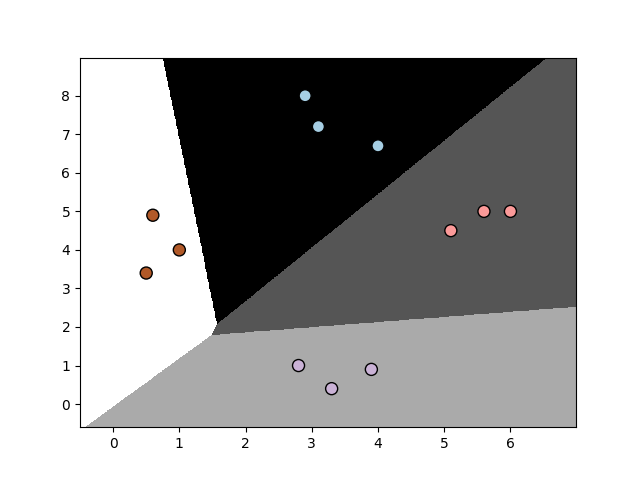
**Завдання №2.3:** Класифікація логістичною регресією або логістичний

класифікатор

Код:

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:**

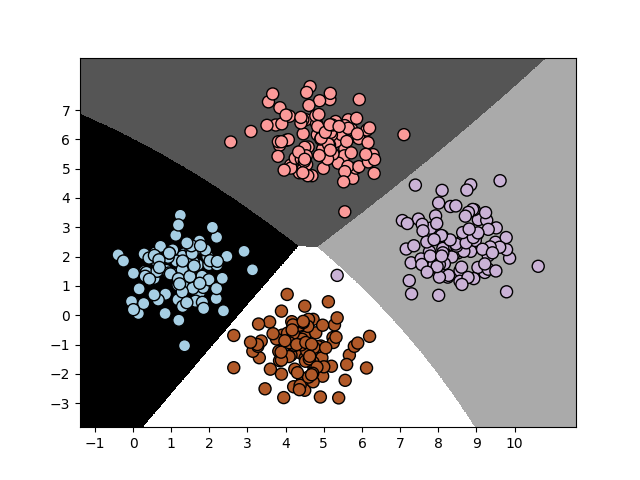


**Завдання №2.4:** Класифікація наївним байєсовським класифікатором

Код:

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)

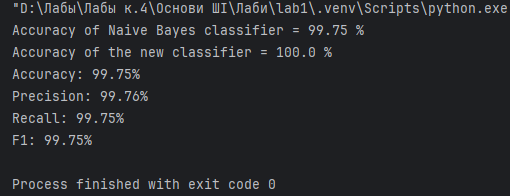
**Результат виконання:**

****

Код:

import numpy as np  
import matplotlib.pyplot as plt  
  
from sklearn.naive\_bayes import GaussianNB  
import sklearn.model\_selection as model\_selection  
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)  
  
# Розбивка даних на навчальний та тестовий набори  
X\_train, X\_test, y\_train, y\_test = model\_selection.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 == y\_pred).sum() / X.shape[0]  
print("Accuracy of Naive Bayes classifier =", round(accuracy,  
2), "%")  
  
# 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 = model\_selection.cross\_val\_score(classifier,  
 X, y, scoring='accuracy', cv=num\_folds)  
  
print("Accuracy: " + str(round(100 \* accuracy\_values.mean(), 2)) + "%")  
  
precision\_values = model\_selection.cross\_val\_score(classifier,  
 X, y, scoring='precision\_weighted', cv=num\_folds)  
  
print("Precision: " + str(round(100 \* precision\_values.mean(),  
 2)) + "%")  
  
recall\_values = model\_selection.cross\_val\_score(classifier,  
 X, y, scoring='recall\_weighted', cv=num\_folds)  
  
print("Recall: " + str(round(100 \* recall\_values.mean(), 2)) +  
 "%")  
  
f1\_values = model\_selection.cross\_val\_score(classifier,  
 X, y, scoring='f1\_weighted', cv=num\_folds)  
  
print("F1: " + str(round(100 \* f1\_values.mean(), 2)) + "%")

**Результат виконання:**

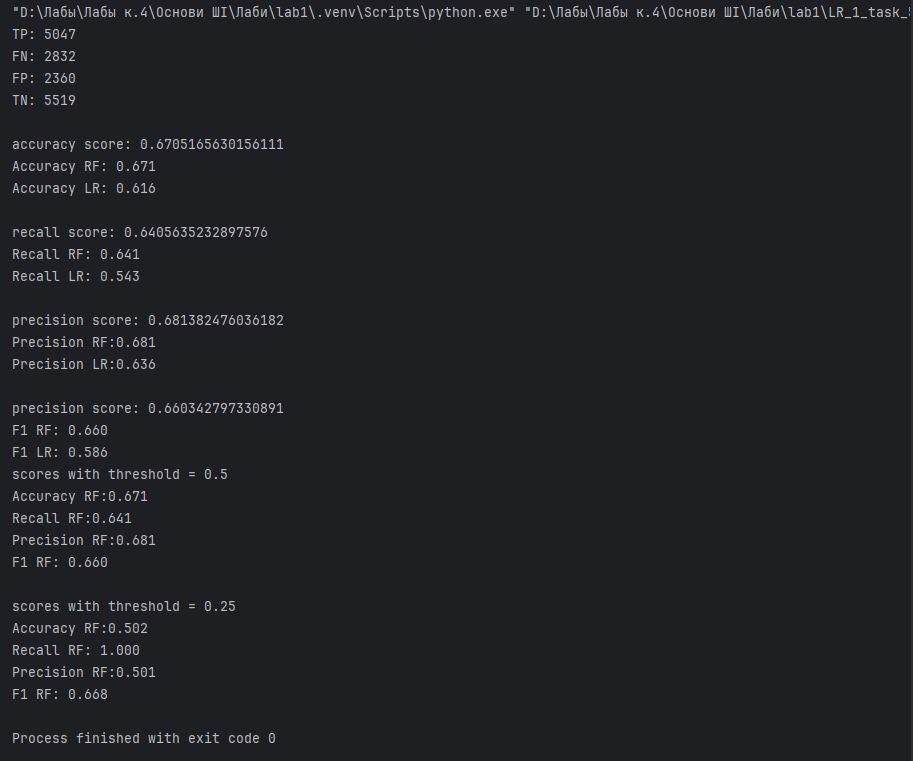
**** ****

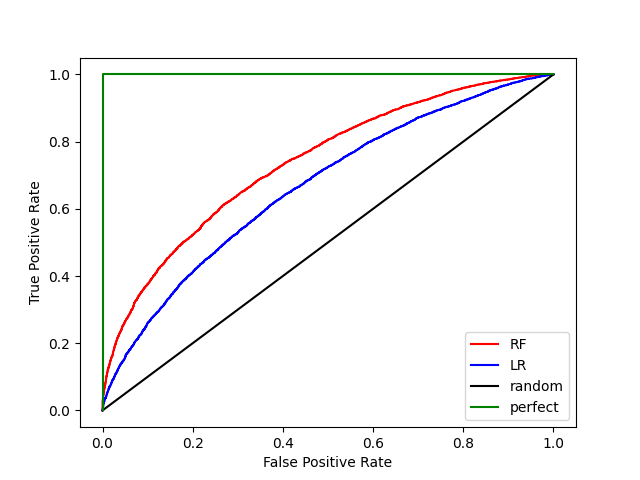
**Завдання №2.5:** Вивчити метрики якості класифікації

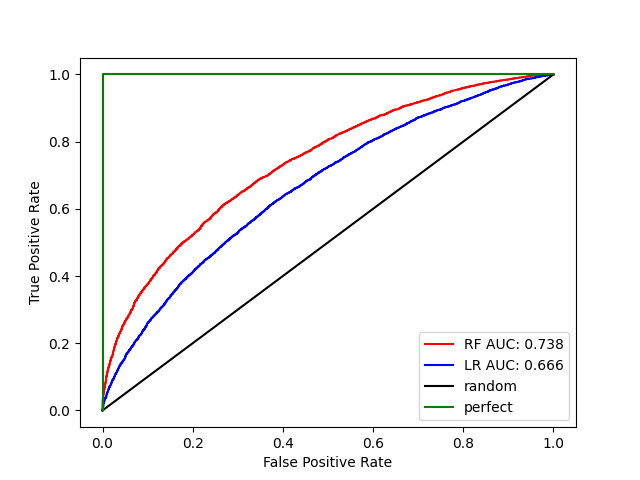
Код:

import pandas as pd  
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()  
  
confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)  
  
  
def find\_TP(y\_true, y\_pred):  
 # counts the number of false negatives (y\_true = 1, y\_pred = 0)  
 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))  
  
  
import numpy as np  
  
  
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 denysiuk\_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]])  
  
  
denysiuk\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)  
  
assert np.array\_equal(denysiuk\_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(denysiuk\_confusion\_matrix(df.actual\_label.values,  
 df.predicted\_LR.values), confusion\_matrix(df.actual\_label.values,  
 df.predicted\_LR.values)), 'denysiuk\_confusion\_matrix() is not correct for LR'  
  
  
from sklearn.metrics import accuracy\_score  
  
  
print('\naccuracy score:', accuracy\_score(df.actual\_label.values, df.predicted\_RF.values))  
  
def denysiuk\_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 denysiuk\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values) == accuracy\_score(df.actual\_label.values, df.predicted\_RF.values),\  
 'denysiuk\_accuracy\_score failed on RF'  
assert denysiuk\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values) == accuracy\_score(df.actual\_label.values, df.predicted\_LR.values),\  
 'denysiuk\_accuracy\_score failed on LR'  
print('Accuracy RF: %.3f'%(denysiuk\_accuracy\_score(df.actual\_label.values,  
 df.predicted\_RF.values)))  
print('Accuracy LR: %.3f'%(denysiuk\_accuracy\_score(df.actual\_label.values,  
 df.predicted\_LR.values)))  
  
  
from sklearn.metrics import recall\_score  
print('\nrecall score:', recall\_score(df.actual\_label.values, df.predicted\_RF.values))  
  
  
def denysiuk\_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 denysiuk\_recall\_score(df.actual\_label.values, df.predicted\_RF.values) == recall\_score(df.actual\_label.values, df.predicted\_RF.values),\  
 'denysiuk\_recall\_score failed on RF'  
assert denysiuk\_recall\_score(df.actual\_label.values, df.predicted\_LR.values) == recall\_score(df.actual\_label.values, df.predicted\_LR.values),\  
 'denysiuk\_recall\_score failed on LR'  
print('Recall RF: %.3f'%(denysiuk\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall LR: %.3f'%(denysiuk\_recall\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
  
from sklearn.metrics import precision\_score  
print('\nprecision score:', precision\_score(df.actual\_label.values, df.predicted\_RF.values))  
  
  
def denysiuk\_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 denysiuk\_precision\_score(df.actual\_label.values, df.predicted\_RF.values) == precision\_score(df.actual\_label.values, df.predicted\_RF.values),\  
 'denysiuk\_precision\_score failed on RF'  
assert denysiuk\_precision\_score(df.actual\_label.values, df.predicted\_LR.values) == precision\_score(df.actual\_label.values, df.predicted\_LR.values),\  
 'denysiuk\_precision\_score failed on LR'  
print('Precision RF:%.3f'%(denysiuk\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision LR:%.3f'%(denysiuk\_precision\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
  
from sklearn.metrics import f1\_score  
print('\nprecision score:', f1\_score(df.actual\_label.values, df.predicted\_RF.values))  
  
def denysiuk\_f1\_score(y\_true, y\_pred):  
 # calculates the F1 score  
 recall = denysiuk\_recall\_score(y\_true, y\_pred)  
 precision = denysiuk\_precision\_score(y\_true, y\_pred)  
 return 2 \* precision \* recall / (precision + recall)  
  
print('F1 RF: %.3f'%(denysiuk\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 LR: %.3f'%(denysiuk\_f1\_score(df.actual\_label.values, df.predicted\_LR.values)))  
  
  
print('scores with threshold = 0.5')  
print('Accuracy RF:%.3f'%(denysiuk\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Recall RF:%.3f'%(denysiuk\_recall\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('Precision RF:%.3f'%(denysiuk\_precision\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('F1 RF: %.3f'%(denysiuk\_f1\_score(df.actual\_label.values, df.predicted\_RF.values)))  
print('')  
print('scores with threshold = 0.25')  
print('Accuracy RF:%.3f'%(denysiuk\_accuracy\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('Recall RF: %.3f'%(denysiuk\_recall\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('Precision RF:%.3f'%(denysiuk\_precision\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
print('F1 RF: %.3f'%(denysiuk\_f1\_score(df.actual\_label.values, (df.model\_RF >= 0.25).astype('int').values)))  
  
  
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)  
  
  
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)  
  
  
import matplotlib.pyplot as plt  
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()

**Результати виконання:**

****





Висновки:

RF: Случайний ліс, зазвичай, добре підходить для роботи з даними високої розмірності та нелінійними взаємозв'язками між ознаками. Він навчається на низці дерев рішень і усереднює їх прогнози, що дозволяє забезпечити високу точність та стійкість до перенавчання.

LR: Логістична регресія, з іншого боку, є лінійним методом, який добре працює у випадку лінійних взаємозв'язків між ознаками. Це проста та швидка модель, яка легко інтерпретується.

**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.

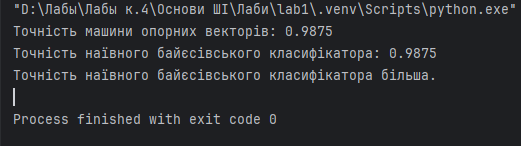
Порівняйте їх з показниками наївного байєсівського класифікатора.

Зробіть висновки яку модель класифікації краще обрати і чому.

Код:

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  
  
# Завантаження даних  
with open("data\_multivar\_nb.txt", "r") as file:  
 data = file.readlines()  
  
# Підготовка даних  
X = []  
y = []  
for line in data:  
 line = line.strip().split(',')  
 X.append([float(x) for x in line[:-1]])  
 y.append(int(line[-1]))  
  
# Розділення даних на навчальний та тестовий набор  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Навчання моделей  
svm\_model = SVC(kernel='linear')  
svm\_model.fit(X\_train, y\_train)  
  
nb\_model = GaussianNB()  
nb\_model.fit(X\_train, y\_train)  
  
# Прогнозування  
svm\_pred = svm\_model.predict(X\_test)  
nb\_pred = nb\_model.predict(X\_test)  
  
# Оцінка точності  
svm\_accuracy = accuracy\_score(y\_test, svm\_pred)  
nb\_accuracy = accuracy\_score(y\_test, nb\_pred)  
  
# Вивід результатів  
print("Точність машини опорних векторів:", svm\_accuracy)  
print("Точність наївного байєсівського класифікатора:", nb\_accuracy)  
  
if svm\_accuracy > nb\_accuracy:  
 print("Точність машини опорних векторів більша.")  
else:  
 print("Точність наївного байєсівського класифікатора більша.")

**Результат виконання:**



**Висновок:** використав спеціалізовані бібліотеки та мову

програмування Python, дослідив попередню обробку та класифікацію даних.