**Практична робота № 1**

**Варіант 13**

**Попередня обробка та контрольована**

**Класифікація даних**

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

**Хід роботи:**

**Завдання 1.1:** Попередня обробка даних:

Як правило, при обробці ми маємо справу з великими обсягами необроблених вихідних даних. Алгоритми машинного навчання розраховані на те, що, перш ніж вони зможуть розпочати процес тренування, отримані дані будуть відформатовані певним чином. Щоб привести дані до форми, що прийнятна для алгоритмів машинного навчання, ми повинні попередньо підготувати їх і перетворити на потрібний формат.

**Зробіть висновок чим відрізняються L1-нормалізація від L2-нормалізації?**

Нормалізація L1 та L2 відрізняються точністю jотриманих після розрахунків суми значень. L2 має меншу точність та є менш надійним, у той час як L1 є манш універсальним та не простежувати неточність вхідних даних.

Лістинг файлу task-1.py

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(f"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))

#Виняток 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  
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  
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("l2 normalized data:\n", data\_normalized\_l2)  
  
# Кодування міток  
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(f"{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: ", 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: ", decoded\_list)

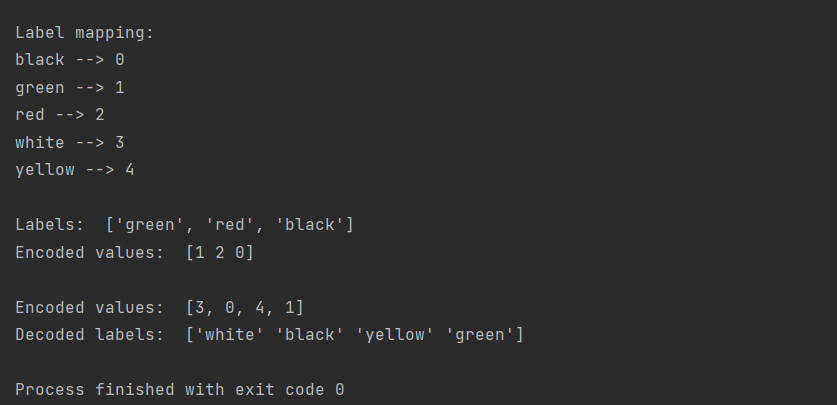
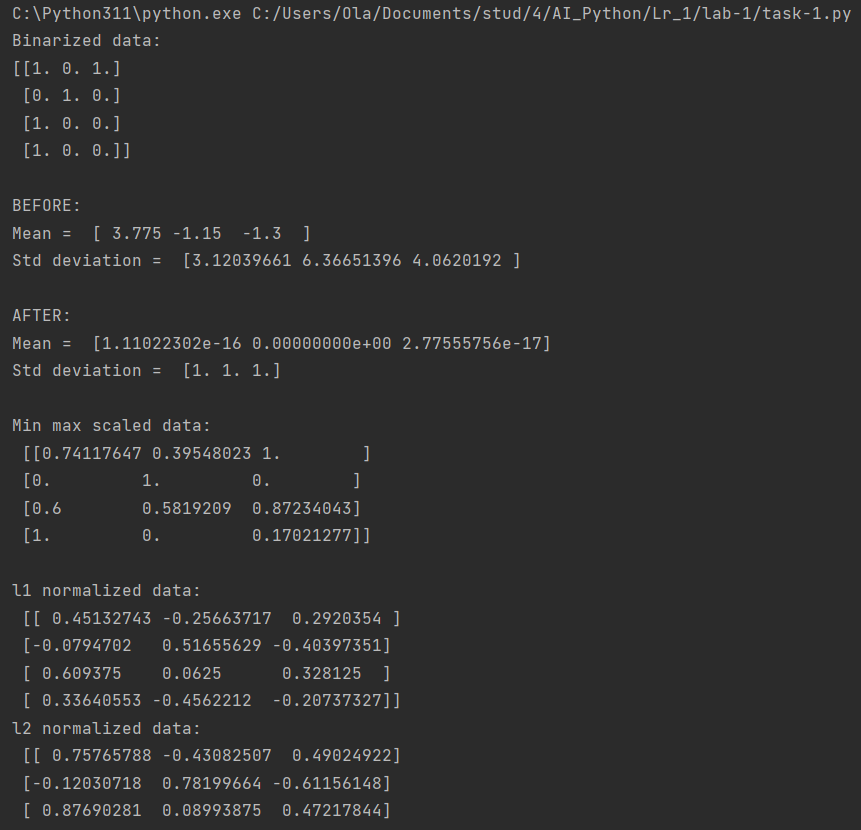
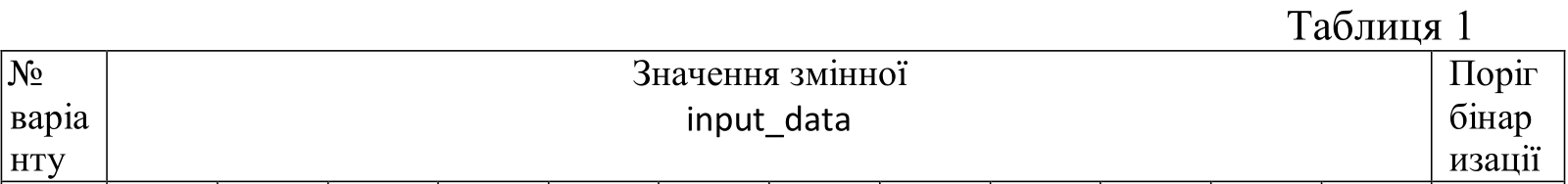


Рис.1.1. task-1.py

**Завдання 1.2:** У коді програми попереднього завдання поміняйте дані по рядках (значення змінної input\_data) на значення відповідно варіанту таблиці 1 та виконайте операції: Бінарізації, Виключення середнього, Масштабування, Нормалізації.

Лістинг файлу task-2.py

import numpy as np  
from sklearn import preprocessing  
  
input\_data = np.array([  
 [-2.3, -1.6, -6.1],  
 [-2.4, -1.2, 4.3],  
 [3.2, 3.1, 6.1],  
 [-4.4, 1.4, -1.2]  
])  
  
# Бінаризація даних  
data\_binarized = preprocessing.Binarizer(threshold=2.1).transform(input\_data)  
print(f"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))  
  
# Виняток 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  
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  
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("l2 normalized data:\n", data\_normalized\_l2)

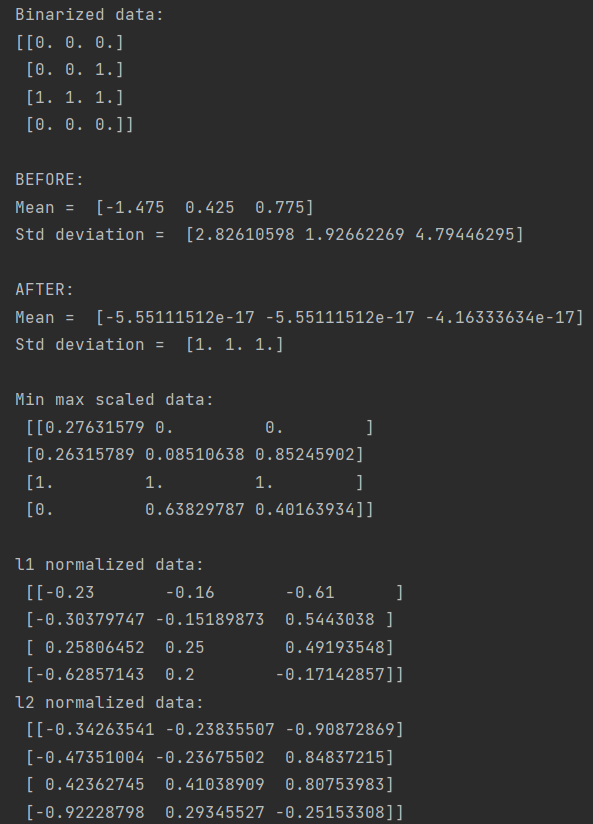


Рис.1.2. task-2.py

**Завдання 1.3:** Класифікація логістичною регресією або логістичний класифікатор.

Логістична регресія – це методика, що використовується для пояснення відносин між вхідними та вихідними змінними. Вхідні змінні вважаються незалежними, вихідні – залежними. Залежна змінна може мати лише фіксований набір значень.

Лістинг файлу task-3.py

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)

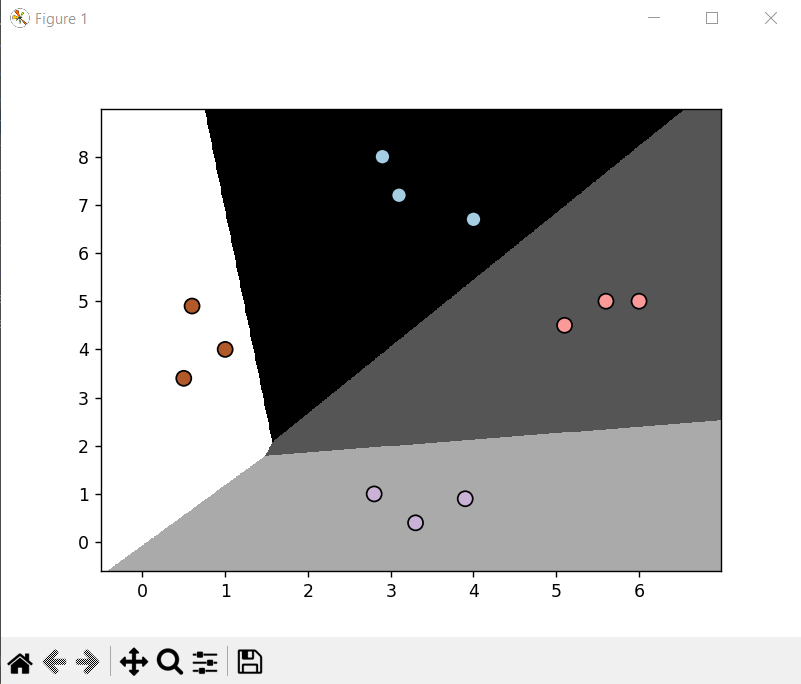


Рис.1.3. task-3.py

**Завдання 1.4:** Класифікація наївним байєсовським класифікатором. Наївний байєсовський : класифікатор - це простий класифікатор, заснований на використанні теореми Байєса, яка описує ймовірність події з урахуванням пов'язаних з нею умов.

Лістинг файлу task-4.py

# 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)) + "%")

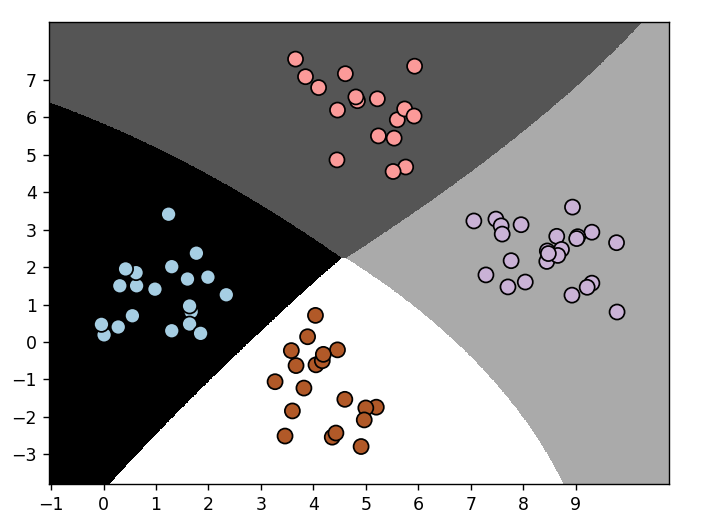
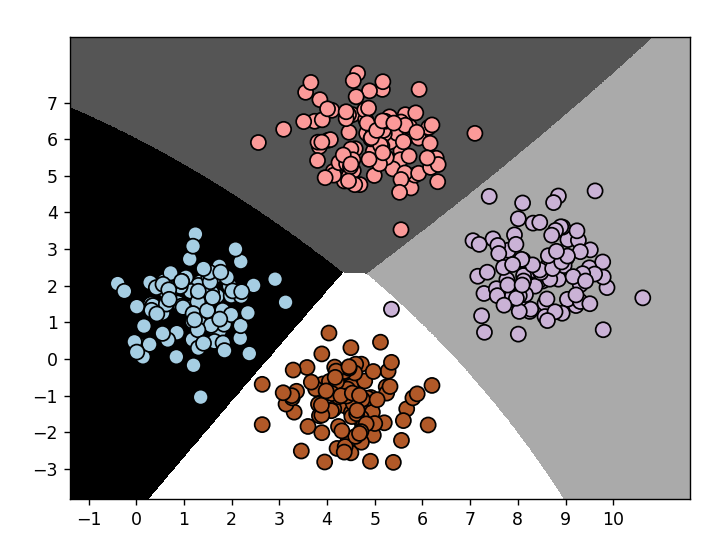


Рис.1.4. task-4.py

**Порівняйте між собою результати висновок запишіть у звіт.**

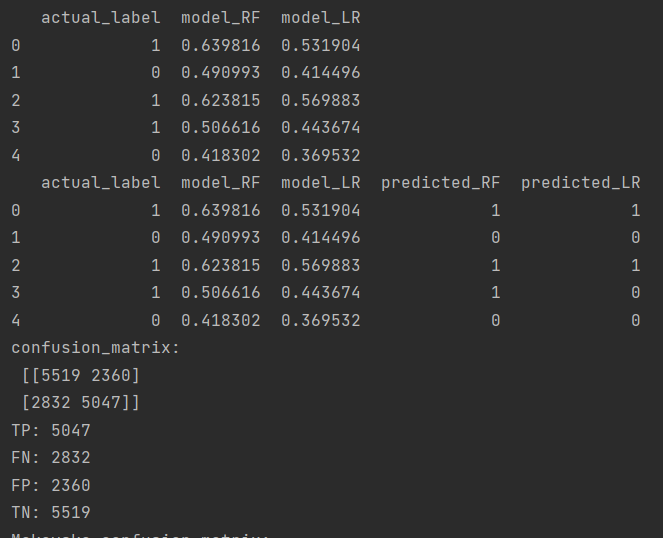
Після проведення декількох експериментів, можемо підсумувати, що розділення даних дає можливість отримати більш надійні данні, а використання функції для обчислення якості, точності та повноти даних дає можливість більш детально вказати результат.

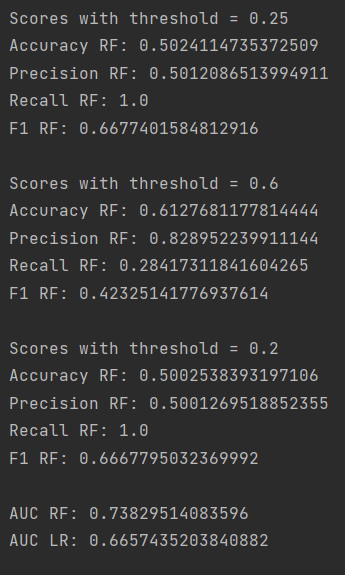
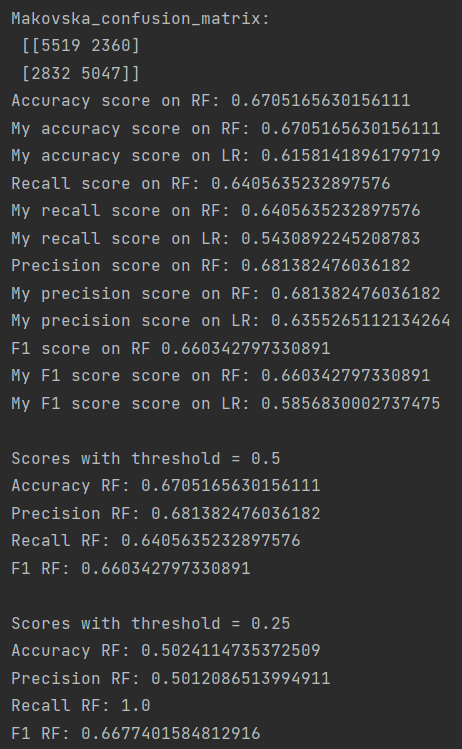
**Завдання 1.5:** Попередня обробка даних: Як правило, при обробці ми маємо справу з великими обсягами

Фрагмент лістингу файлу task-5.py

def 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 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))  
  
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

# Confusion  
def Makovska\_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]])  
  
# Accuracy  
def Makovska\_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)  
  
# Recall  
  
  
def Makovska\_recal\_score(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return TP / (TP + FN)  
  
def Makovska\_precision\_score(y\_true, y\_pred):  
 TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)  
 return TP / (TP + FP)  
  
# F1 score  
def Makovska\_f1\_score(y\_true, y\_pred):  
 precision = Makovska\_precision\_score(y\_true, y\_pred)  
 recall = Makovska\_recal\_score(y\_true, y\_pred)  
 return (2 \* (precision \* recall)) / (precision + recall)





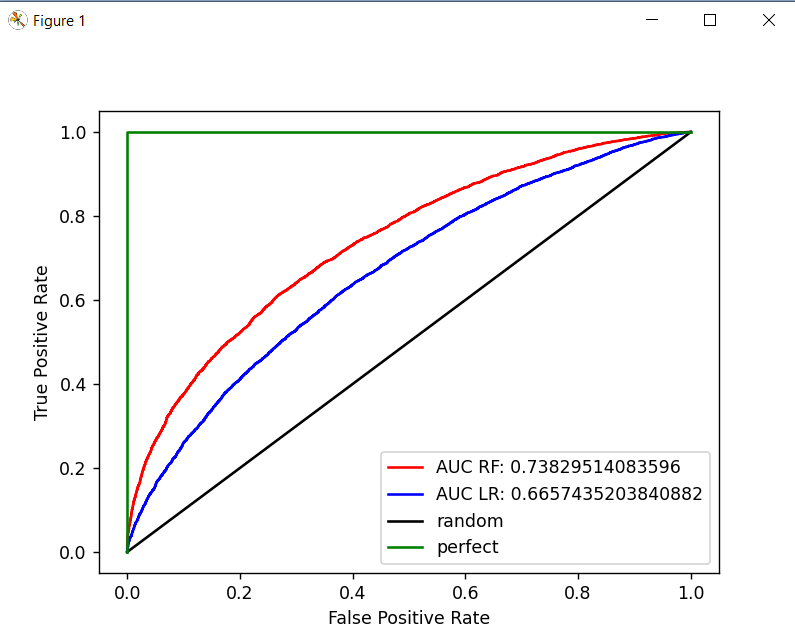


Рис.1.5. task-5.py

**Завдання 1.6:** Попередня обробка даних: Як правило, при обробці ми маємо справу з великими обсягами

Лістинг файлу task-2.py

input\_file = 'data\_multivar\_nb.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.21, random\_state=5)  
classifier = svm.SVC(decision\_function\_shape='ovr')  
classifier.fit(X\_train, y\_train)  
y\_test\_pred = classifier.predict(X\_test)  
  
visualize\_classifier(classifier, X\_test, y\_test)  
  
num\_folds = 3  
accuracy\_values = cross\_val\_score(classifier, X\_test, y\_test, scoring='accuracy', cv=num\_folds)  
print(f"Accuracy: {round(100 \* accuracy\_values.mean(), 3)}%")  
  
precision\_values = cross\_val\_score(classifier, X\_test, y\_test, scoring='precision\_weighted', cv=num\_folds)  
print(f"Precision: {round(100 \* precision\_values.mean(), 3)}%")  
  
recall\_values = cross\_val\_score(classifier, X\_test, y\_test, scoring='recall\_weighted', cv=num\_folds)  
print(f"Recall: {round(100 \* recall\_values.mean(), 3)}%")  
  
f1\_values = cross\_val\_score(classifier, X\_test, y\_test, scoring='f1\_weighted', cv=num\_folds)  
print(f"F1: {round(100 \* f1\_values.mean(), 3)}%")

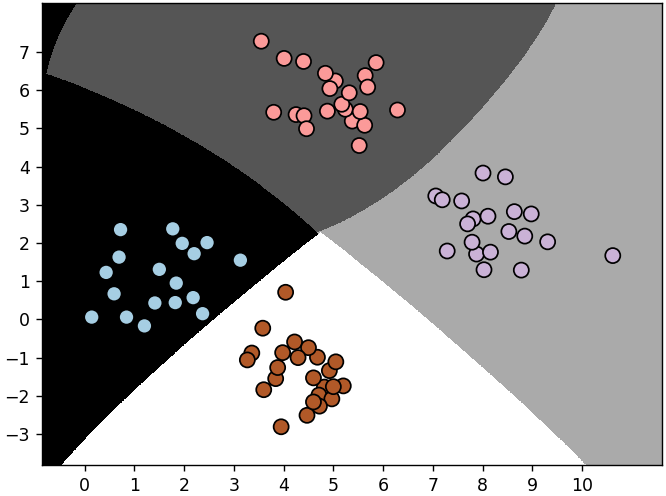


Рис.1.6. task-6.py

**<https://github.com/avrorilka/AI_Python>**

**Висновки:** в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python ми дослідити попередню обробку та класифікацію даних. Розглянули такі методи підготовки та перетворення даних як бінаризація даних, виключення середнього, масштабування, нормалізація, розглянули та проаналізували різницю між L1 та L2 нормалізаціями, та кодування міток. Навчилися застосовувати методику логічної регресії для класифікації логістичною регістрацією та класифікацію наївним байєсовським класифікатором. Також ми вивчили метрики якості класифікації.