**ЛАБОРАТОРНА РОБОТА № 4**

**ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ**

Мета роботи: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні та створити рекомендаційні системи.

ЧАСИНА 1. ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

**1. ТЕОРЕТИЧНІ ВІДОМОСТІ**

Теоретичні відомості подані на лекціях. Також доцільно вивчити матеріал поданий в літературі:

Джоши Пратик. Искусственный интеллект с примерами на Python. : Пер. с англ. - СПб. : ООО "Диалектика", 2019. - 448 с. - Парал. тит. англ. ISBN 978-5-907114-41-8 (рус.)

Можна використовувати Google Colab або Jupiter Notebook.

**2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ**

**Завдання 2.1. Створення класифікаторів на основі випадкових та гранично випадкових лісів**

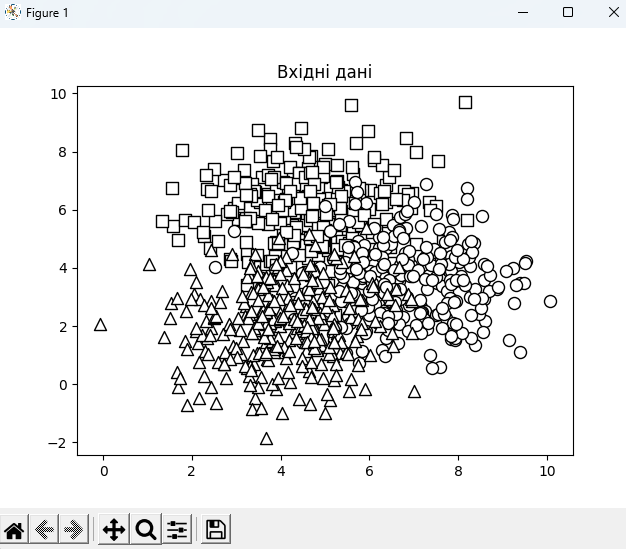


Рис 1. Вхідні дані.

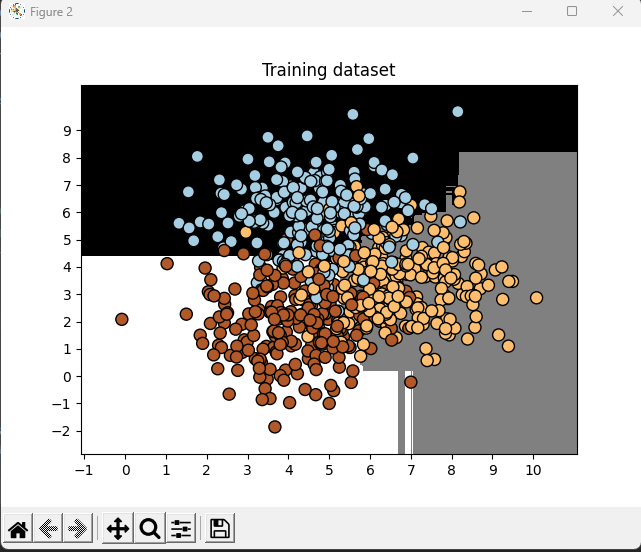


Рис 2. Training dataset

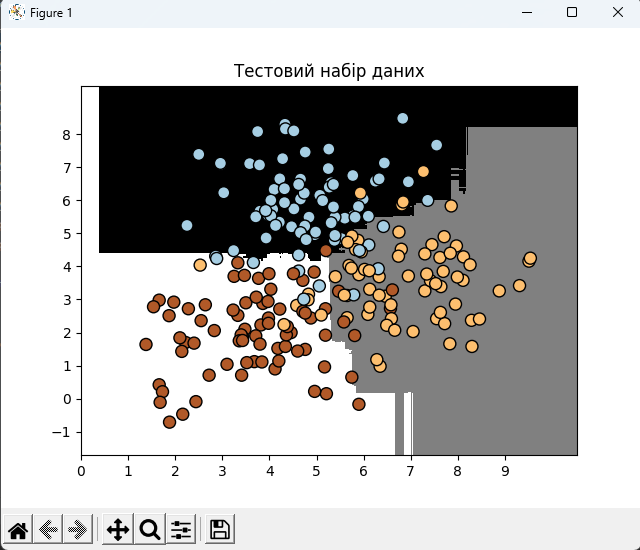


Рис 3. Тестовий набір даних

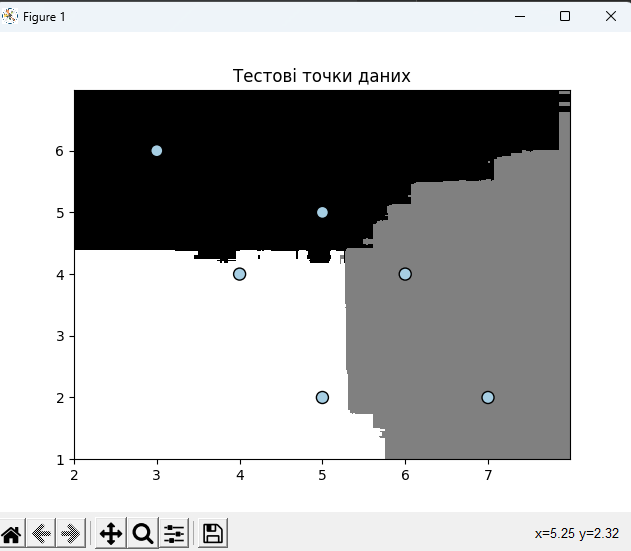


Рис 4. Тестові точки даних.

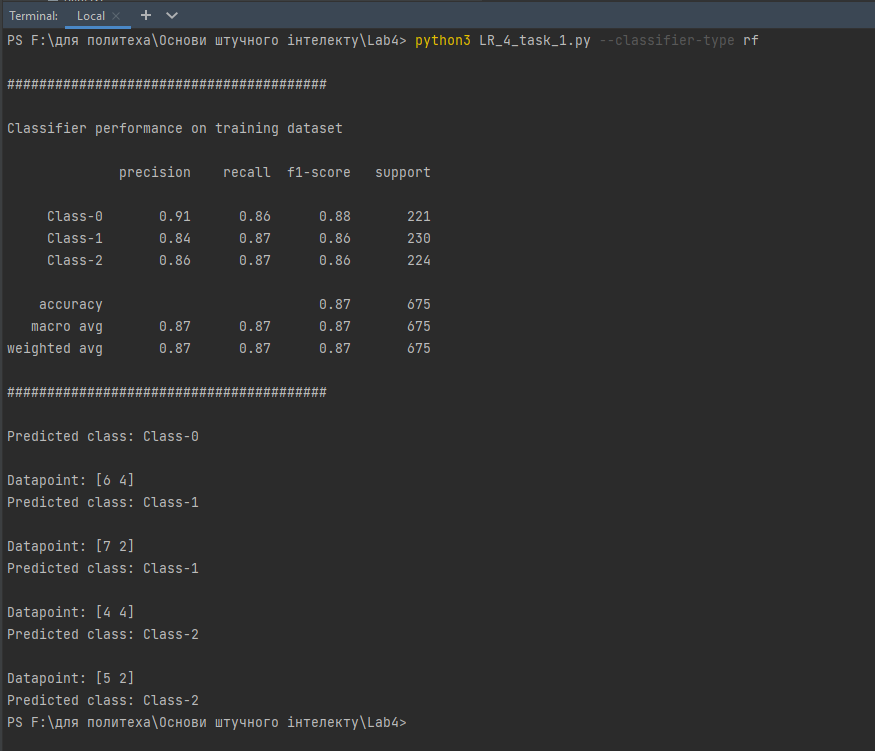


Рис 5. Результат перерахунку.

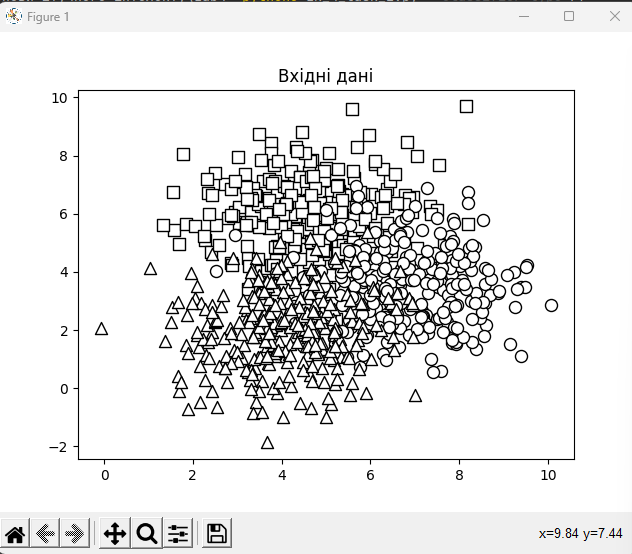


Рис 6. Вхідні дані.

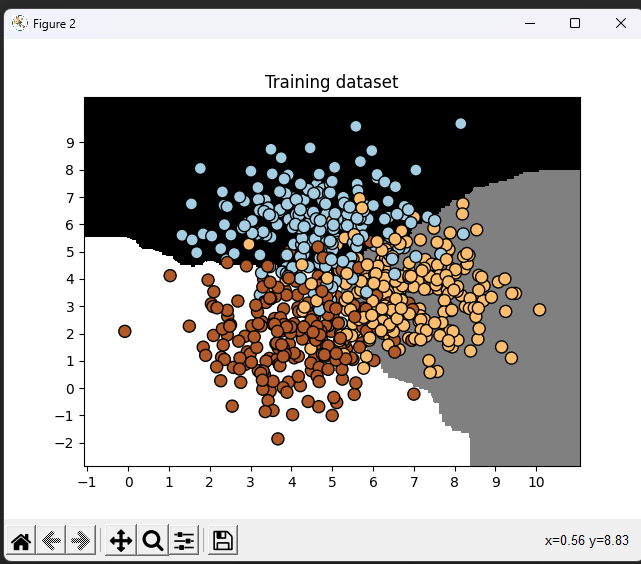


Рис 7. Training dataset.

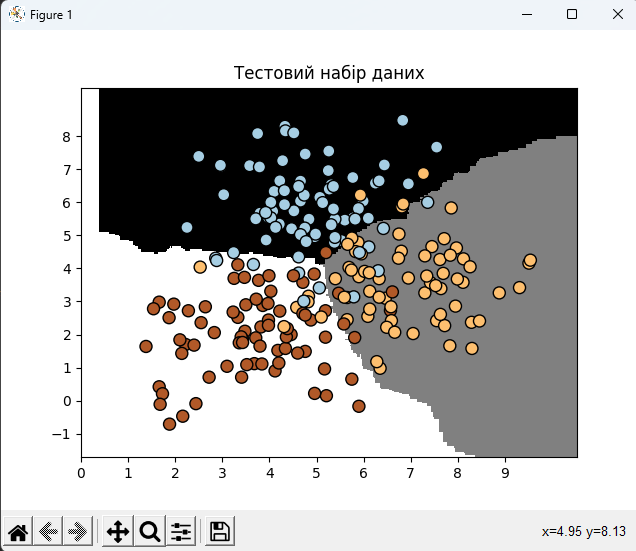


Рис 8. Тестовий набір даних.

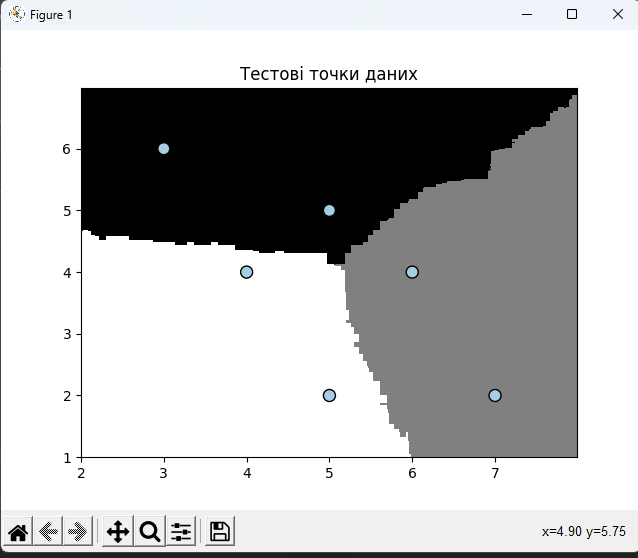
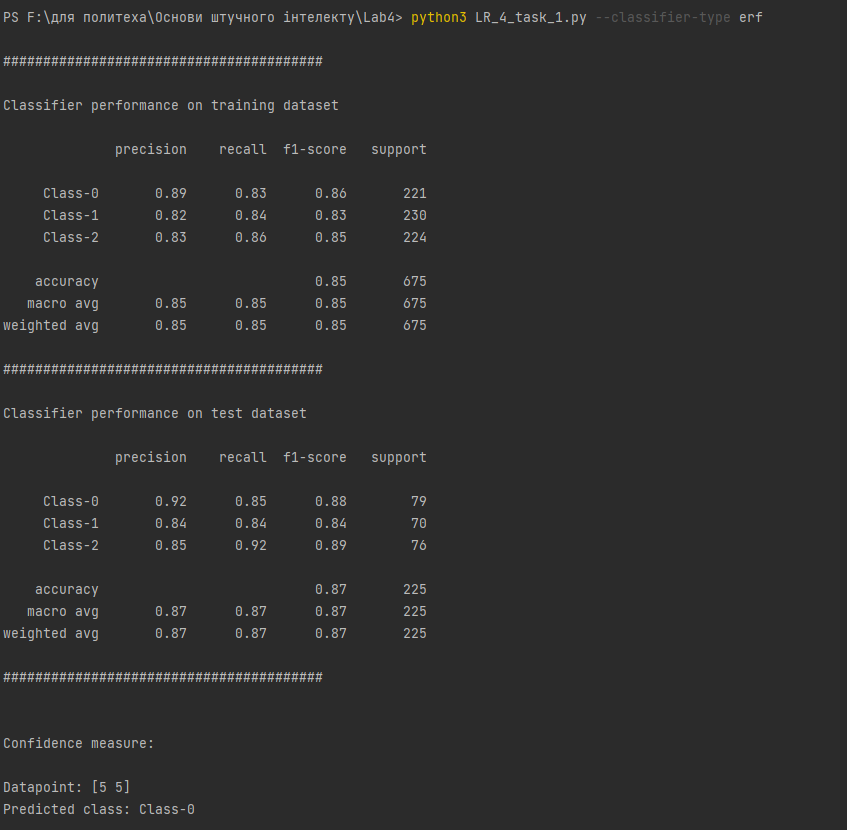


Рис 8. Тестові точки даних.

Рис 9. Результат роботи програми.

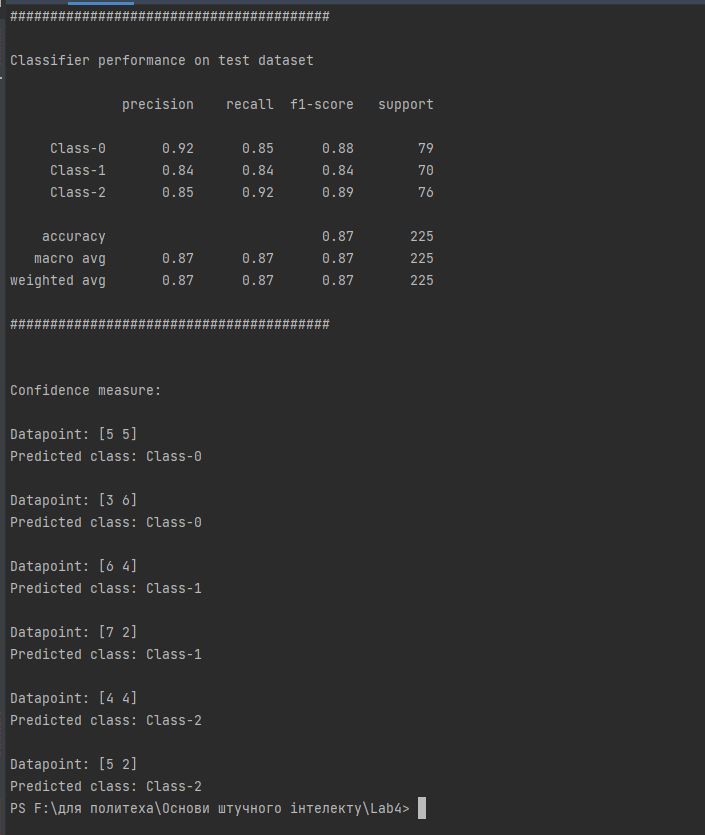


Рис 10. Продовження рис №9

**Завдання 2.2. Обробка дисбалансу класів**

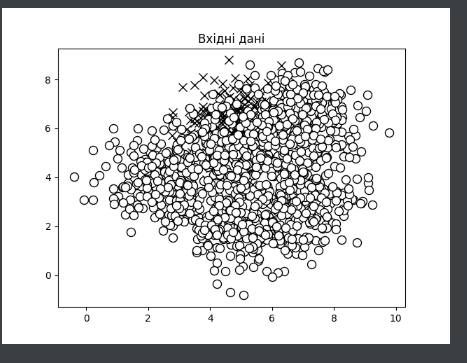


Рис 11. Вхідні дані.

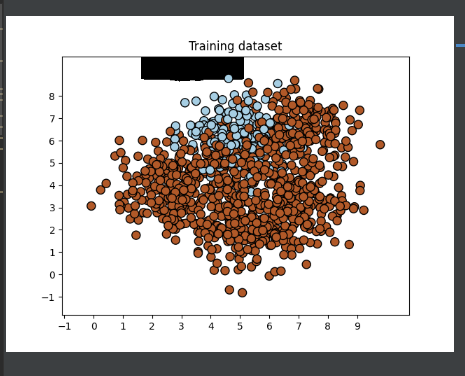


Рис 12. Training dataset

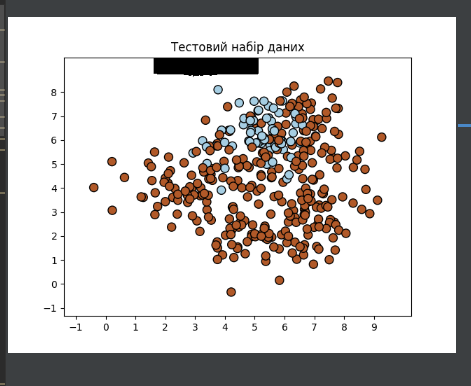


Рис 13. Тестовий набір даних

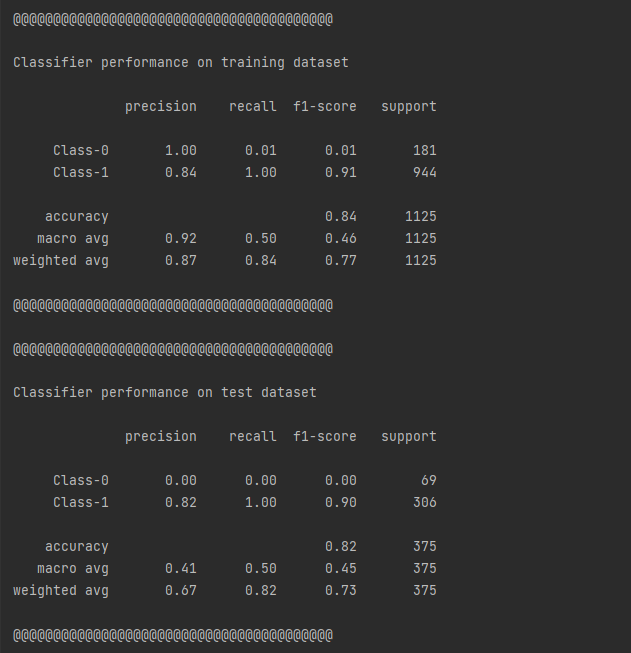


Рис 14. Результат роботи програми

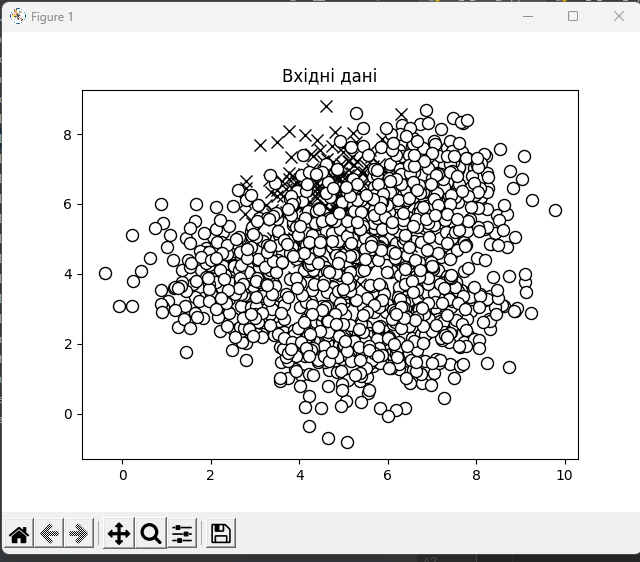


Рис 15. Вхідні дані.

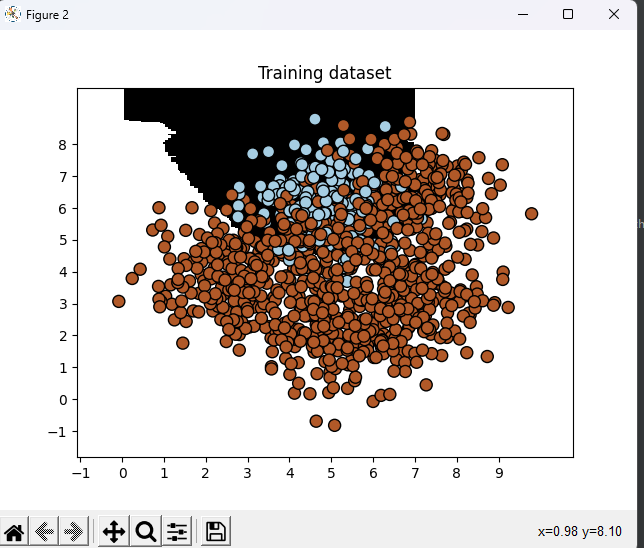


Рис 16. Training dataset.

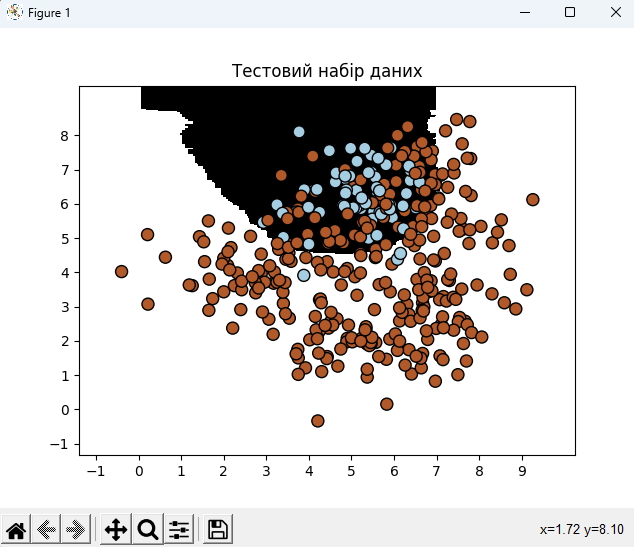


Рис 17. Тестовий набір даних

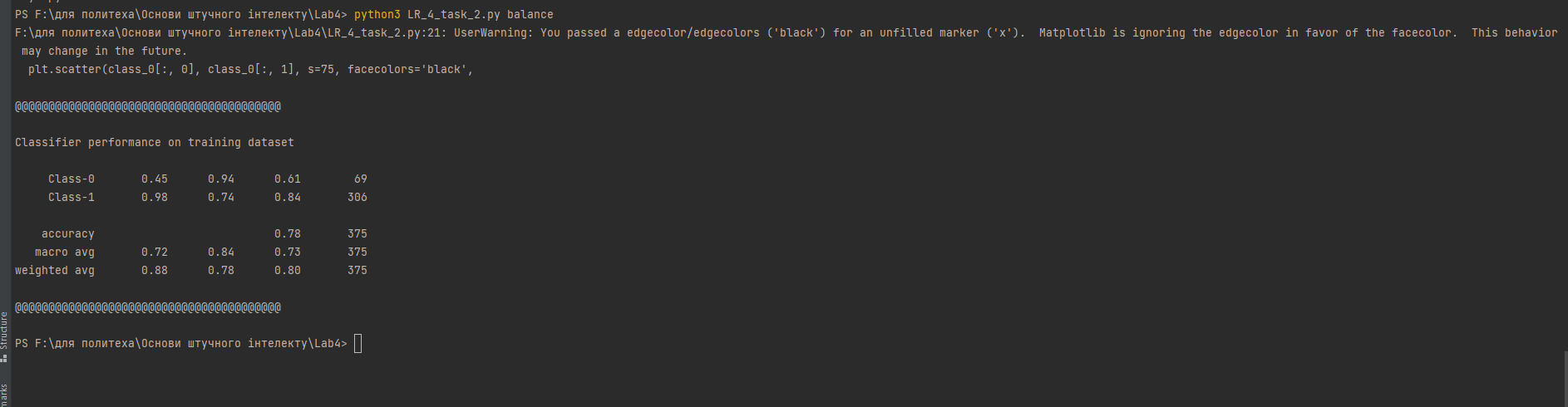


Рис 18. Результат роботи програми

**Завдання 2.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from sklearn.ensemble import ExtraTreesClassifier**

**from sklearn.metrics import classification\_report**

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

**from sklearn.model\_selection import GridSearchCV**

**from utilities import visualize\_classifier**

**input\_file = 'data\_random\_forests.txt'**

**data = np.loadtxt(input\_file, delimiter=',')**

**X, y = data[:, :-1], data[:, -1]**

**# Розбиття даних на три класи на підставі міток**

**class\_0 = np.array(X[y==0])**

**class\_1 = np.array(X[y==1])**

**class\_2 = np.array(X[y==2])**

**# Розбиття даних на навчальний та тестовий набори**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(**

**X, y, test\_size=0.25, random\_state=5)**

**# Визначення сітки значень параметрів**

**parameter\_grid = [ {'n\_estimators': [100], 'max\_depth': [2, 4, 7, 12, 16]},**

**{'max\_depth': [4], 'n\_estimators': [25, 50, 100, 250]}**

**]**

**metrics = ['precision\_weighted', 'recall\_weighted']**

**for metric in metrics:**

**print("\n@@@@@ Searching optimal parameters for", metric)**

**classifier = GridSearchCV(**

**ExtraTreesClassifier(random\_state=0),**

**parameter\_grid, cv=5, scoring=metric)**

**classifier.fit(X\_train, y\_train)**

**print("\nGrid scores for the parameter grid:")**

**for params, avg\_score in zip(classifier.cv\_results\_['params'], classifier.cv\_results\_['mean\_test\_score']):**

**print(params, '-->', round(avg\_score, 3))**

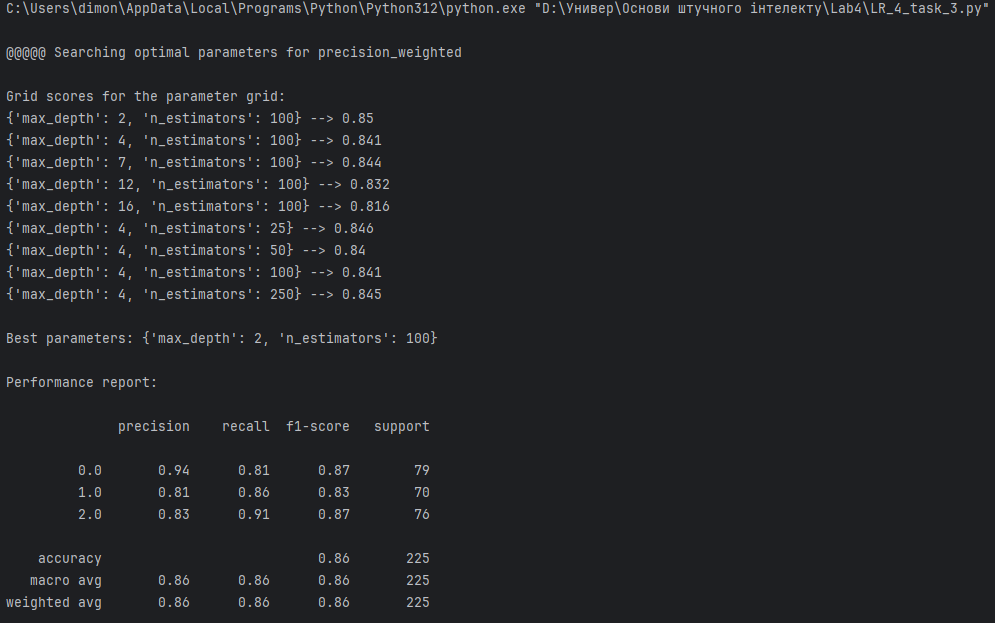
**print("\nBest parameters:", classifier.best\_params\_)**

**y\_pred = classifier.predict(X\_test)**

**print("\nPerformance report:\n")**

**print(classification\_report(y\_test, y\_pred))**

Рис 19. Результат роботи програми



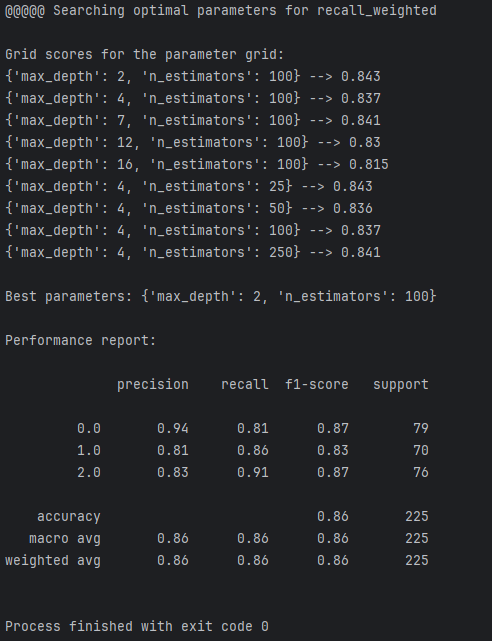
****

Рис 20. Продовження Рис №19

**Завдання 2.4. Обчислення відносної важливості ознак**

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeRegressor  
from sklearn.ensemble import AdaBoostRegressor  
from sklearn import datasets  
from sklearn.metrics import mean\_squared\_error, explained\_variance\_score  
from sklearn.utils import shuffle  
  
# Load housing data  
housing\_data = datasets.load\_boston()  
  
# Shuffle the data  
X, y = shuffle(housing\_data.data, housing\_data.target, random\_state=7)  
  
# Split data into training and testing datasets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.2, random\_state=7)  
  
# AdaBoost Regressor model  
regressor = AdaBoostRegressor(DecisionTreeRegressor(max\_depth=4),  
 n\_estimators=400, random\_state=7)  
regressor.fit(X\_train, y\_train)  
  
# Evaluate performance of AdaBoost regressor  
y\_pred = regressor.predict(X\_test)  
mse = mean\_squared\_error(y\_test, y\_pred)  
evs = explained\_variance\_score(y\_test, y\_pred )  
print("\nADABOOST REGRESSOR")  
print("Mean squared error =", round(mse, 2))  
print("Explained variance score =", round(evs, 2))  
  
# Extract feature importances  
feature\_importances = regressor.feature\_importances\_  
feature\_names = housing\_data.feature\_names  
  
# Normalize the importance values  
feature\_importances = 100.0 \* (feature\_importances / max(feature\_importances))  
  
# Sort the values and flip them  
index\_sorted = np.flipud(np.argsort(feature\_importances))  
  
# Arrange the X ticks  
pos = np.arange(index\_sorted.shape[0]) + 0.5  
  
# Plot the bar graph  
plt.figure()  
plt.bar(pos, feature\_importances[index\_sorted], align='center')  
plt.xticks(pos, feature\_names[index\_sorted])  
plt.ylabel('Relative Importance')  
plt.title('Оцінка важливості ознак з використанням регресора')  
plt.show()

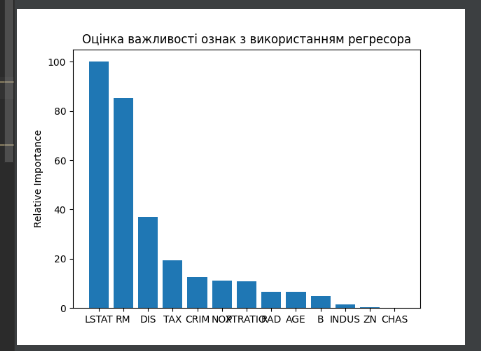


Рис 21. Графік роботи програми

**Завдання 2.5. Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from sklearn.metrics import classification\_report, mean\_absolute\_error**

**from sklearn import preprocessing**

**from sklearn.ensemble import ExtraTreesRegressor**

**from sklearn.metrics import classification\_report**

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

**input\_file = 'traffic\_data.txt'**

**data = []**

**with open(input\_file, 'r') as f:**

**for line in f.readlines():**

**items = line[:-1].split(',')**

**data.append(items)**

**data = np.array(data)**

**label\_encoder = []**

**X\_encoded = np.empty(data.shape)**

**for i, item in enumerate(data[0]):**

**if item.isdigit():**

**X\_encoded[:, i] = data[:, i]**

**else:**

**label\_encoder.append(preprocessing.LabelEncoder())**

**X\_encoded[:, i] = label\_encoder[-1].fit\_transform(data[:, i])**

**X = X\_encoded[:, :-1].astype(int)**

**y = X\_encoded[:, -1].astype(int)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(**

**X, y, test\_size=0.25, random\_state=5)**

**params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0}**

**regressor = ExtraTreesRegressor(\*\*params)**

**regressor.fit(X\_train, y\_train)**

**y\_pred = regressor.predict(X\_test)**

**print("Mean absolute error:", round(mean\_absolute\_error(y\_test, y\_pred), 2))**

**test\_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']**

**test\_datapoint\_encoded = [-1] \* len(test\_datapoint)**

**count = 0**

**for i, item in enumerate(test\_datapoint):**

**if item.isdigit():**

**test\_datapoint\_encoded[i] = int(test\_datapoint[i])**

**else:**

**test\_datapoint\_encoded[i] = int(label\_encoder[i].transform([test\_datapoint[i]]))**

**count = count + 1**

**test\_datapoint\_encoded = np.array(test\_datapoint\_encoded)**

**print("Predicted traffic:", int(regressor.predict([test\_datapoint\_encoded])[0]))**

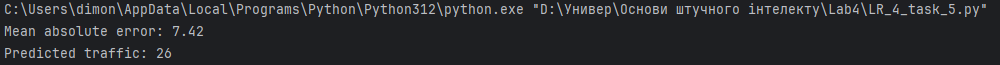
****

Рис 22. Результат роботи програми

**ЧАСТИНА 2. СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ**

**1. ТЕОРЕТИЧНІ ВІДОМОСТІ**

Системи рекомендацій Персоналізація користувацького досвіду була пріоритетом і стала новою мантрою в галузях, орієнтованих на споживача. Можливо, ви помітили, як компанії електронної комерції розміщують персоналізовану рекламу для вас, пропонуючи, що купувати, які новини читати, яке відео дивитися, де/що їсти та з ким вам може бути цікаво спілкуватися (друзі/професіонали) у соціальних мережах. медіа сайти. Системи рекомендацій — це основна система фільтрації інформації, розроблена для прогнозування переваг користувачів і допомоги рекомендувати правильні елементи для створення специфічного для користувача досвіду персоналізації.

**2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ**

**Завдання 2.6. Створення навчального конвеєра (конвеєра машинного навчання)**

**Необхідно створити конвеєр, призначений для вибору найбільш важливих ознак з вхідних даних і їх подальшої класифікації з використанням класифікатора на основі гранично випадкового лісу.**

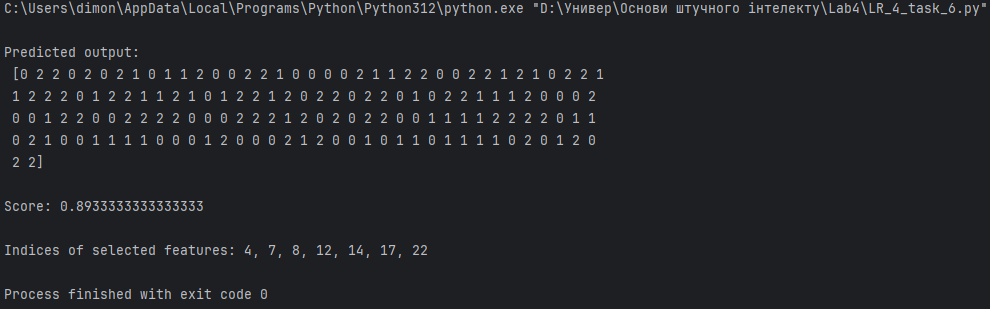


Рис 23. Результат роботи програми.

**Висновок**: Перший перелік містить передбачені класифікаційні мітки для наших даних. Значення Score вказує на точність або якість нашої моделі чи алгоритму класифікації. Чим вище це значення, тим краще. У нашому випадку воно підтверджує, що наша модель правильно класифікувала близько 88,67% прикладів.

**Завдання 2.7. Пошук найближчих сусідів**

Для формування ефективних рекомендацій у рекомендаційних системах використовується поняття найближчих сусідів (nearest neighbours), суть якого полягає у знаходженні тих точок заданого набору, які розташовані на найближчих відстанях від зазначеної. Такий підхід часто застосовується для 24 створення систем, що класифікують точку даних на підставі її близькості до різних класів. Здійсніть пошук найближчих сусідів заданої точки даних.

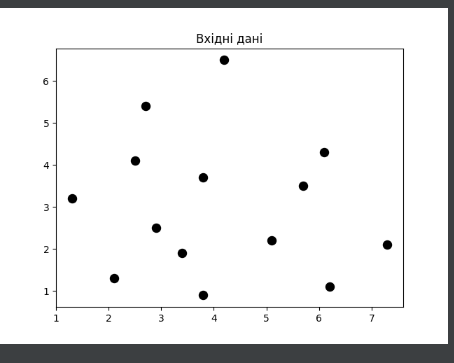


Рис 24. Вхідні дані

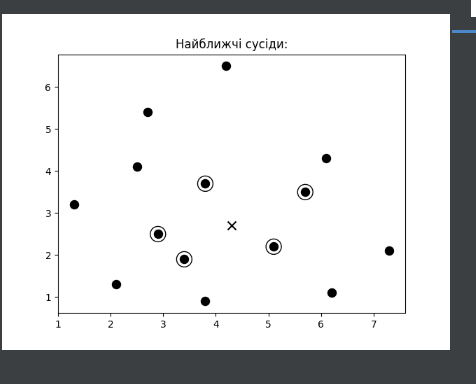


Рис 25. Найближчі сусіди

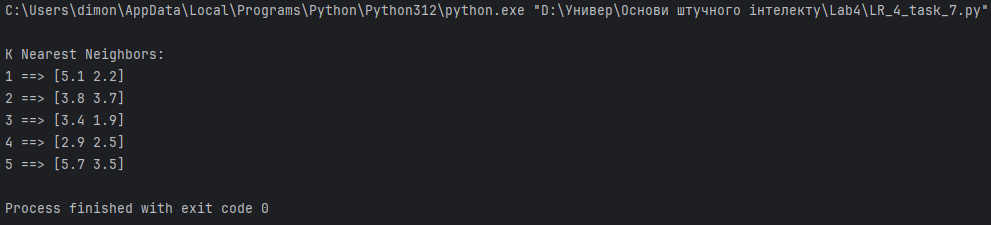


Рис 26. Результат роботи програми.

**Висновок:**

Перший графік відображає вхідні дані у двовимірному просторі, де кожна точка позначена чорним колом (маркер 'о'). Ця візуалізація надає інформацію про розташування всіх точок у масиві X.

Другий графік показує найближчі сусіди для тестової точки даних, яку позначено хрестиком (маркер 'х') та чорним кольором. Точки, які є найближчими сусідами до тестової точки, виділені чорними колами з відкритими контурами (facecolors='none'). Цей графік ілюструє результати застосування методу k-найближчих сусідів до тестової точки.

У вікні терміналу представлена інформація щодо k найближчих сусідів тестової точки даних. Це список п'яти найближчих сусідів до тестової точки, впорядкованих за відстанню від неї. Кожен рядок містить порядковий номер найближчого сусіда та його координати.

**Завдання 2.8. Створити класифікатор методом k найближчих сусідів**

Класифікатор на основі k найближчих сусідів – це модель класифікації, в якій задана точка класифікується з використанням алгоритму найближчих сусідів. Для визначення категорії вхідної точки, даний алгоритм знаходить у навчальному наборі k точок, що є найближчими по відношенню до заданої. Після цього призначений точці даних клас визначається "голосуванням". Ми переглядаємо класи k елементів отриманим списком і вибираємо з них той клас, якому відповідає найбільша кількість "голосів". Значення k залежить від конкретного завдання.

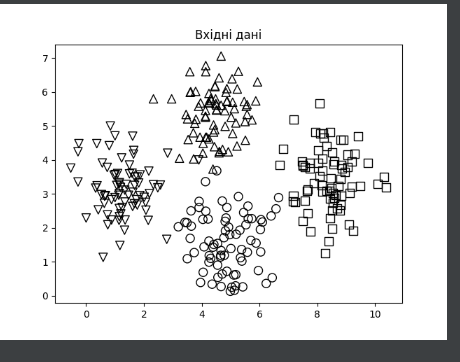


Рис 27. Вхідні дані.

"Вхідні дані"

Графік відтворює вхідні дані з файлу data.txt, де кожна точка визначена на графіку в залежності від двох ознак. Різні маркери використовуються для позначення точок залежно від їх класу, який передбачений за значенням останньої колонки у файлі.

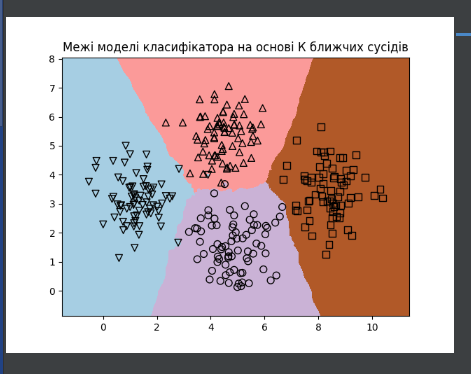


Рис 28. Межі моделі класифікатора на основі K ближчих сусідів.

Графік відображає передбачені границі рішення класифікаційної моделі, яка базується на методі k-найближчих сусідів, для всього простору ознак. Різні кольори позначають різні класи.

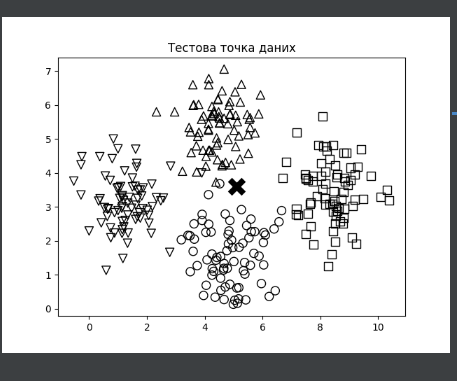


Рис 29. Тестова точка даних.

На графіку видно тестову точку даних, позначену символом "x", яку ми прагнемо класифікувати. Цей елемент відзначений окремо для визначення його положення в контексті класифікації.

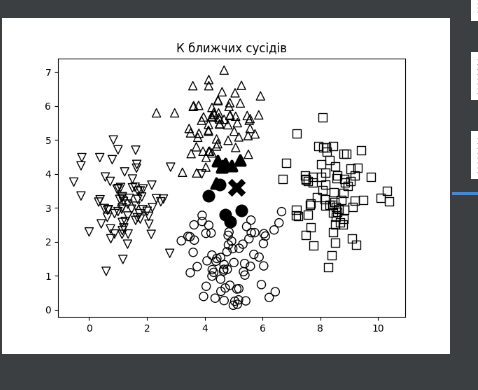


Рис 30. K ближчих сусідів

На графіку представлено K найближчих сусідів для тестової точки даних, яку відзначено чорним хрестом. Ці найближчі сусіди виділені чорними колами, що дозволяє візуально оцінити їх розташування та взаємодію з тестовою точкою. Це ілюструє процес класифікації та допомагає зрозуміти, як алгоритм k-найближчих сусідів приймає рішення щодо класифікації даної точки.

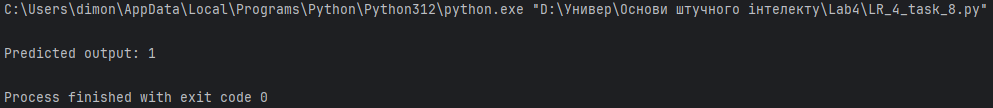


Рис 31. Результат роботи програми

**Завдання 2.9. Обчислення оцінок подібності.**

При побудові рекомендаційних систем дуже важливу роль відіграє вибір способу порівняння різних об'єктів, що входять до набору даних. Припустимо, що наш набір даних включає інформацію про користувачів та їх переваги. Для того, щоб щось рекомендувати, ми повинні розуміти, як порівнювати смаки різних людей. І тут першому плані виходять оцінки подібності (similaгity scores). Оцінка подібності дає уявлення про те у якій мірі два об’єкта можуть вважатися аналогічними один одному. Для цієї мети часто використовують оцінки двох типів: евклідові і по Пірсону. В основу евклідової оцінки (метрики) покладено відстань між двома точками даних. Якщо вам необхідно освіжити знання щодо того, що таке евклідова відстань, то загляньте в Вікіпедію (https://ru. wikipedia. org /wi ki /Евклідова\_метрика). Евклідова відстань не обмежена за величиною. Тому ми беремо відповідне значення та перетворимо його таким чином, щоб нове значення знаходилося в діапазоні від 0 до 1. Якщо евклідова відстань між двома об'єктами велика, то відповідна евклідова оцінка повинна мати невелику величину, оскільки низька оцінка вказує на малу міру подібності між об'єктами. Отже, евклідова відстань обернено пропорційно евклідовій оцінці.

Оцінка подібності за Пірсоном (Pearson score) – це математична міра кореляції двох об'єктів. Для її обчислення використовують коваріацію (covariance) двох об'єктів та їх індивідуальні стандартні відхилення (Standard deviations). Значення цієї оцінки можуть бути змінені в межах від -1 до +1. Оцінка +1 свідчить про високий рівень подібності об'єктів, тоді як оцінка -1 - великі відмінності з-поміж них. Оцінка 0 свідчить про відсутність кореляції між двома об'єктами.

David Smith та Brenda Peterson

David Smith та Samuel Miller

David Smith та Julie Hammel

David Smith та Clarissa Jackson

David Smith та Adam Cohen

David Smith та Chris Duncan

import argparse

import json

import numpy as np

def build\_arg\_parser():

parser = argparse.ArgumentParser(description='Compute similarity score')

parser.add\_argument('--user1', dest='user1', required=True,

help='First user')

parser.add\_argument('--user2', dest='user2', required=True,

help='Second user')

parser.add\_argument("--score-type", dest="score\_type", required=True,

choices=['Euclidean', 'Pearson'], help='Similarity metric to be used')

return parser

def euclidean\_score(dataset, user1, user2):

# Обчислення оцінки евклідова відстані між

# користувачами userl та user2

if user1 not in dataset:

raise TypeError('Cannot find ' + user1 + ' in the dataset')

if user2 not in dataset:

raise TypeError('Cannot find ' + user2 + ' in the dataset')

# Фільми, оцінені обома користувачами, userl та user2

common\_movies = {}

for item in dataset[user1]:

if item in dataset[user2]:

common\_movies[item] = 1

# За відсутності фільмів, оцінених обома користувачами,

# оцінка приймається рівною 0

if len(common\_movies) == 0:

return 0

squared\_diff = []

for item in dataset[user1]:

if item in dataset[user2]:

squared\_diff.append(np.square(dataset[user1][item]-dataset[user2][item]))

return 1 / (1 + np.sqrt(np.sum(squared\_diff)))

def pearson\_score(dataset, user1, user2):

if user1 not in dataset:

raise TypeError('Cannot find ' + user1 + ' in the dataset')

if user2 not in dataset:

raise TypeError('Cannot find ' + user2 + ' in the dataset')

# Фільми, оцінені обома користувачами, userl та user2

common\_movies = {}

for item in dataset[user1]:

if item in dataset[user2]:

common\_movies[item] = 1

num\_ratings = len(common\_movies)

# За відсутності фільмів, оцінених обома користувачами,

# оцінка приймається рівною 0

if num\_ratings == 0:

return 0

#Обчислення суми рейтингових оцінок усіх фільмів

# оцінених обома користувачами

user1\_sum = np.sum([dataset[user1][item] for item in common\_movies])

user2\_sum = np.sum([dataset[user2][item] for item in common\_movies])

# Обчислення Суми квадратів рейтингових оцінок всіх

# фільмів, оцінених обома користувачами

user1\_squared\_sum = np.sum([np.square(dataset[user1][item])for item in common\_movies])

user2\_squared\_sum = np.sum([np.square(dataset[user2][item])for item in common\_movies])

# Обчислення суми творів рейтингових оцінок всіх

# фільмів, оцінених обома користувачами

sum\_of\_products = np.sum([dataset[user1][item] \* dataset[user2][item] for item in common\_movies])

# Обчислення коефіцієнта кореляції Пірсона

Sxy = sum\_of\_products - (user1\_sum \* user2\_sum / num\_ratings)

Sxx = user1\_squared\_sum - np.square(user1\_sum) / num\_ratings

Syy = user2\_squared\_sum - np.square(user2\_sum) / num\_ratings

if Sxx \* Syy == 0:

return 0

return Sxy / np.sqrt(Sxx \* Syy)

if \_\_name\_\_=='\_\_main\_\_':

args = build\_arg\_parser().parse\_args()

user1 = args.user1

user2 = args.user2

score\_type = args.score\_type

ratings\_file = 'ratings.json'

with open(ratings\_file, 'r') as f:

data = json.loads(f.read())

if score\_type == 'Euclidean':

print("\nEuclidean score:")

print(euclidean\_score(data,user1,user2))

else:

print("\nPearson score:")

print(pearson\_score(data, user1, user2))

Рис 32. Результат порівняння оцінок користувачів.

**Висновок:** Загальною тенденцією може бути те, що вищі значення Пірсонового коефіцієнта можуть свідчити про більшу схожість між користувачами, у той час як відстань Євкліда може вказувати на більшу відстань або меншу схожість. Важливо враховувати, що результати можуть варіюватися залежно від конкретних даних та використовуваних метрик для обчислень.

**Завдання 2.10. Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації**

Термін колаборативна фільтрація (collaborative filtering) відноситься до процесу ідентифікації шаблонів поведінки об'єктів набору даних з метою прийняття рішень щодо нового об'єкта. У контексті рекомендаційних систем метод колаборативної фільтрації використовують для прогнозування уподобань нового користувача на підставі наявної інформації про уподобання інших користувачів з аналогічними смаками.

Складаючи прогнози для переваг індивідуальних користувачів, ми використовуємо спільну інформацію про уподобання інших користувачів. Саме тому цей метод фільтрації називається колаборативним.

В даному випадку основне припущення полягає в тому, що якщо дві людини дають однакові рейтингові оцінки деякому набору фільмів, то їх оцінки фільмів з невідомого набору також будуть приблизно однаковими. Знаходячи загальні оціночні судження щодо одних фільмів, ми можемо прогнозувати оцінки щодо інших фільмів. З попереднього завдання ми дізналися, як порівнювати між собою різних користувачів у межах одного набору даних. Ці методики оцінки подібності використовуються для пошуку користувачів зі схожими уподобаннями в нашому наборі даних. Як правило, колаборативне фільтрування застосовують, коли мають справу з наборами даних великого розміру. Методи такого типу можна використовувати в різних областях, включаючи фінансовий аналіз, онлайн-покупки, маркетингові дослідження, вивчення купівельних звичок і т.п.

import argparse

import json

import numpy as np

from compute\_scores import pearson\_score

def build\_arg\_parser():

parser = argparse.ArgumentParser(description='Find users who are similar to the input user')

parser.add\_argument('--user', dest='user', required=True,

help='Input user')

return parser

def find\_similar\_users(dataset, user, num\_users):

if user not in dataset:

raise TypeError('Cannot find ' + user + ' in the dataset')

# Обчислення оцінки подібності за Пірсоном між

# вказаним користувачем та всіма іншими

# користувачами в наборі даних

scores = np.array([[x, pearson\_score(dataset, user, x)] for x in dataset if x != user])

# Сортування оцінок за спаданням

scores\_sorted = np.argsort(scores[:, 1])[::-1]

# Вилучення оцінок перших 'num\_users' користувачів

top\_users = scores\_sorted[:num\_users]

return scores[top\_users]

if \_\_name\_\_=='\_\_main\_\_':

args = build\_arg\_parser().parse\_args()

user = args.user

ratings\_file = 'ratings.json'

with open(ratings\_file, 'r') as f:

data = json.loads(f.read())

print('\nUsers similar to ' + user + ':\n')

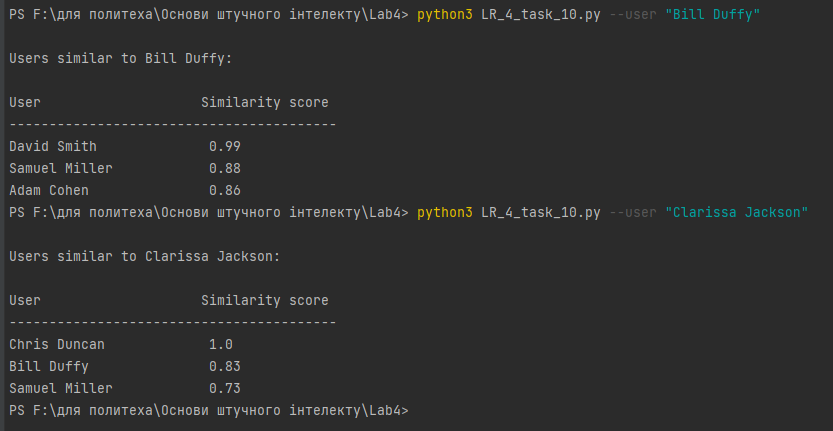
similar\_users = find\_similar\_users(data, user, 3)

print('User\t\t\tSimilarity score')

print('-'\*41)

for item in similar\_users:

print(item[0], '\t\t', round(float(item[1]), 2))



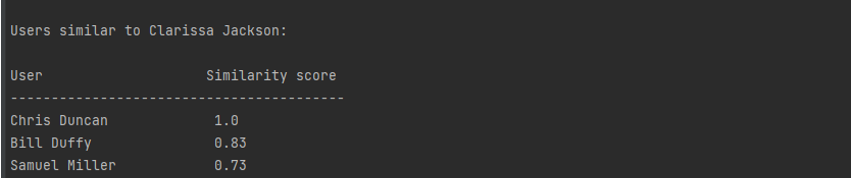


Рис 33. Результат. пошуку користувачів зі схожими уподобаннями

**Висновок**: Ці дані свідчать про подібність між користувачами на основі певних критеріїв чи взаємодій. Однак важливо відзначити, що спосіб визначення схожості може варіюватися залежно від методів аналізу та обраних критеріїв, і це не завжди вказує на структуровані чи однозначні відносини між користувачами.

**Завдання 2.11.Створення рекомендаційної системи фільмів**

Створіть рекомендаційну систему на основі даних, наданих у файлі ratings.json. У цьому файлі міститься інформація про користувачів та оцінки, дані ними різним фільмам. Щоб рекомендувати фільми конкретному користувачу, ми повинні знайти аналогічних користувачів у наборі даних та використовувати інформацію про їх переваги для формування відповідної рекомендації.

import argparse

import json

import numpy as np

from compute\_scores import pearson\_score

def build\_arg\_parser():

parser = argparse.ArgumentParser(description='Find the movie recommendations for the given user')

parser.add\_argument('--user', dest='user', required=True,

help='Input user')

return parser

# Отримати рекомендації щодо фільмів

# для вказаного користувача

def get\_recommendations(dataset, input\_user):

if input\_user not in dataset:

raise TypeError('Cannot find ' + input\_user + ' in the dataset')

overall\_scores = {}

similarity\_scores = {}

for user in [x for x in dataset if x != input\_user]:

similarity\_score = pearson\_score(dataset, input\_user, user)

if similarity\_score <= 0:

continue

filtered\_list = [x for x in dataset[user] if x not in \

dataset[input\_user] or dataset[input\_user][x] == 0]

for item in filtered\_list:

overall\_scores.update({item: dataset[user][item] \* similarity\_score})

similarity\_scores.update({item: similarity\_score})

if len(overall\_scores) == 0:

return ['No recommendations possible']

# Генерація рейтингів фільмів за допомогою їх нормалізації

movie\_scores = np.array([[score / similarity\_scores[item], item]

for item, score in overall\_scores.items()])

# Сортування за спаданням

movie\_scores = movie\_scores[np.argsort(movie\_scores[:, 0])[::-1]]

# Вилучення рекомендацій фільмів

movie\_recommendations = [movie for \_, movie in movie\_scores]

return movie\_recommendations

if \_\_name\_\_ == '\_\_main\_\_':

args = build\_arg\_parser().parse\_args()

user = args.user

ratings\_file = 'ratings.json'

with open(ratings\_file, 'r') as f:

data = json.loads(f.read())

print("\nMovie recommendations for " + user + ":")

movies = get\_recommendations(data, user)

for i, movie in enumerate(movies):

print(str(i + 1) + '. ' + movie)

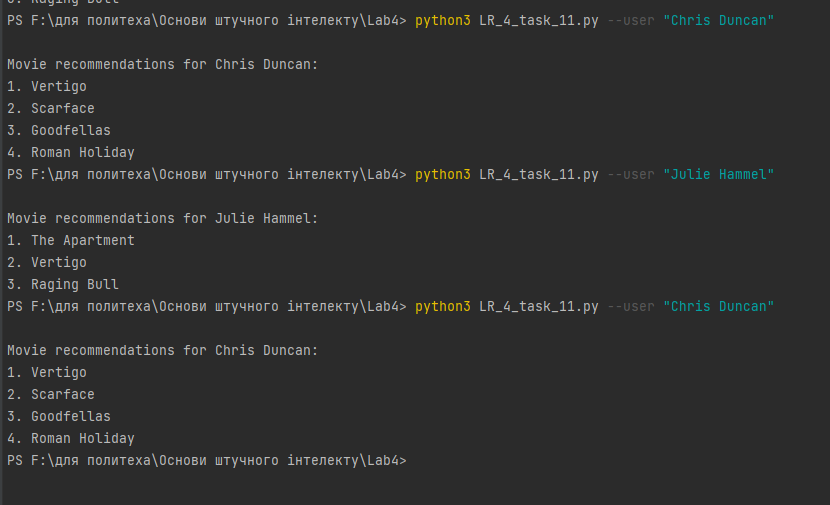


Рис 34. Результат рекомендації фільмів для користувача

Посилання на Git:

Висновок: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідив методи ансамблів у машинному навчанні та створити рекомендаційні системи.