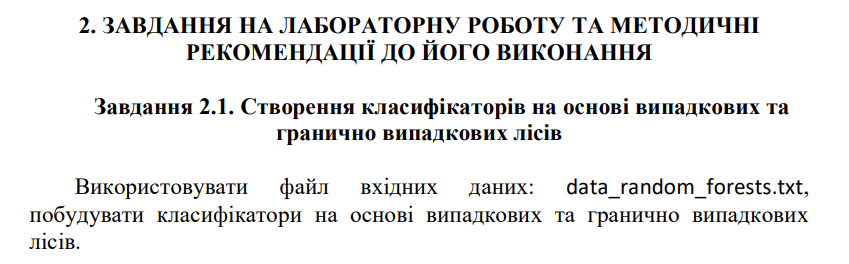
**ЛАБОРАТОРНА РОБОТА № 4**

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА

СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

**Мета роботи**: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанніта створити рекомендаційні системи.**Посилання на GitHub:** **https://github.com/enot1k666/labsOAI.git**

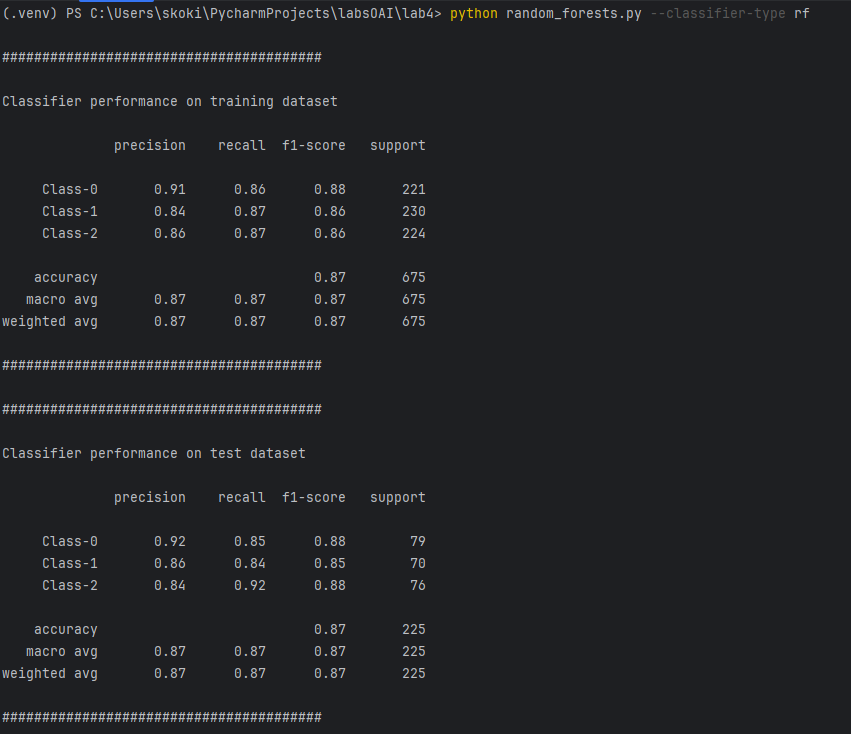


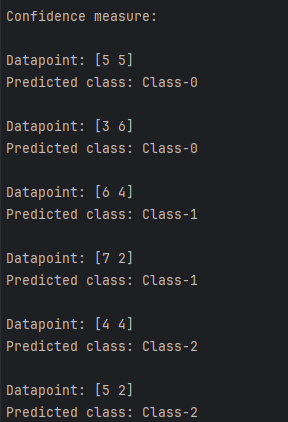
**Лістинг програми:**

import argparse  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report  
from utilities import visualize\_classifier  
  
  
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(description='Classify data using Ensemble Learning techniques')  
 parser.add\_argument('--classifier-type', dest='classifier\_type', required=True,  
 choices=['rf', 'erf'], help="Type of classifier to use; can be either 'rf' or 'erf'")  
 return parser  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 args = build\_arg\_parser().parse\_args()  
 classifier\_type = args.classifier\_type  
  
 input\_file = 'data\_random\_forests.txt'  
 data = np.loadtxt(input\_file, delimiter=',')  
 X, y = data[:, :-1], data[:, -1]  
  
 class\_0 = np.array(X[y == 0])  
 class\_1 = np.array(X[y == 1])  
 class\_2 = np.array(X[y == 2])  
  
 plt.figure()  
 plt.scatter(class\_0[:, 0], class\_0[:, 1], s=75, facecolors='white',  
 edgecolors='black', linewidth=1, marker='s')

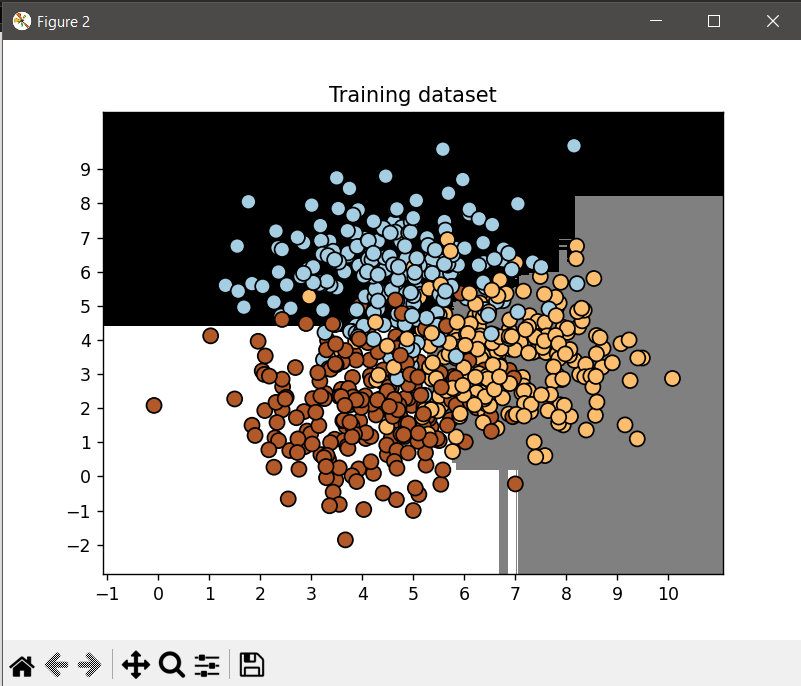
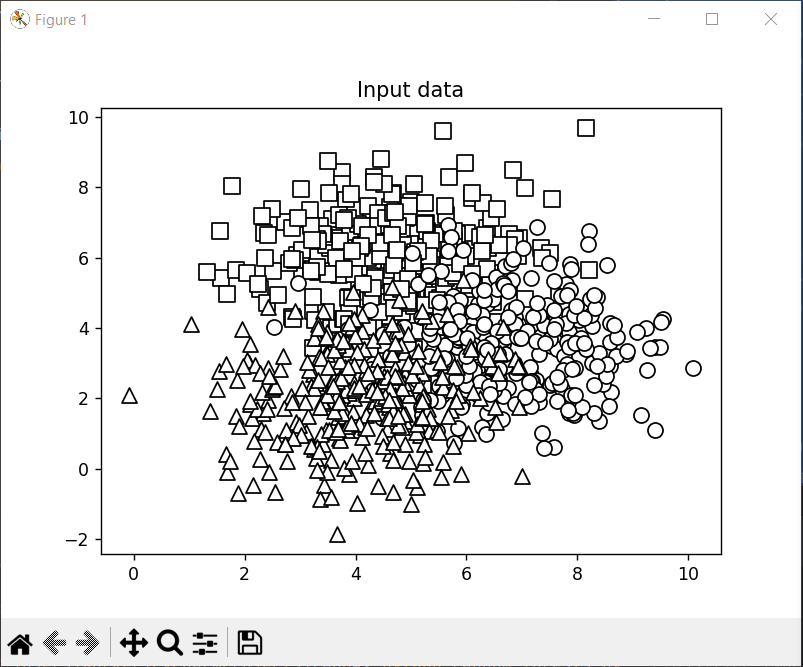
plt.scatter(class\_1[:, 0], class\_1[:, 1], s=75, facecolors='white',  
 edgecolors='black', linewidth=1, marker='o')  
 plt.scatter(class\_2[:, 0], class\_2[:, 1], s=75, facecolors='white',  
 edgecolors='black', linewidth=1, marker='^')  
 plt.title('Вхідні дані')  
  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=5)  
 params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0}  
  
 if classifier\_type == 'rf':  
 classifier = RandomForestClassifier(\*\*params)  
 else:  
 classifier = ExtraTreesClassifier(\*\*params)  
  
 classifier.fit(X\_train, y\_train)  
 visualize\_classifier(classifier, X\_train, y\_train, 'Тренувальний набір даних')  
  
 y\_test\_pred = classifier.predict(X\_test)  
 visualize\_classifier(classifier, X\_test, y\_test, 'Тестовий набір даних')  
  
 class\_names = ['Class-0', 'Class-1', 'Class-2']  
 print("\n" + "#" \* 40)  
 print("\nClassifier performance on training dataset\n")  
 print(classification\_report(y\_train, classifier.predict(X\_train), target\_names=class\_names))  
 print("#" \* 40 + "\n")  
  
 print("#" \* 40)  
 print("\nClassifier performance on test dataset\n")  
 print(classification\_report(y\_test, y\_test\_pred, target\_names=class\_names))  
 print("#" \* 40 + "\n")  
  
 test\_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2], [4, 4], [5, 2]])  
 print('\nConfidense measure:')  
 for datapoint in test\_datapoints:  
 probabilities = classifier.predict\_proba([datapoint])[0]  
 predicted\_class = 'Class-' + str(np.argmax(probabilities))  
 print('\nDatapoint:', datapoint)  
 print('Predicted class:', predicted\_class)  
  
 visualize\_classifier(classifier, test\_datapoints, [0] \* len(test\_datapoints), 'Тестові точки даних')  
 plt.show()

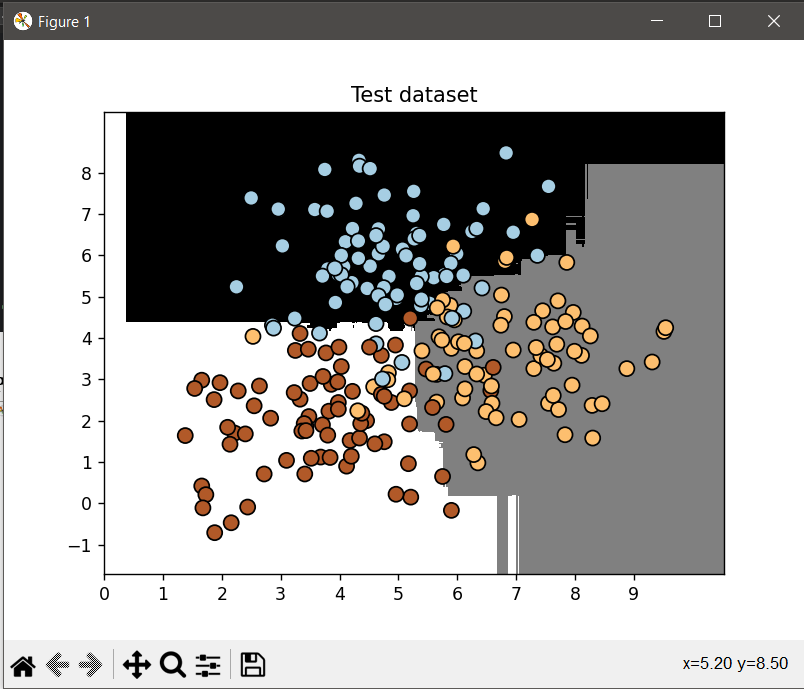
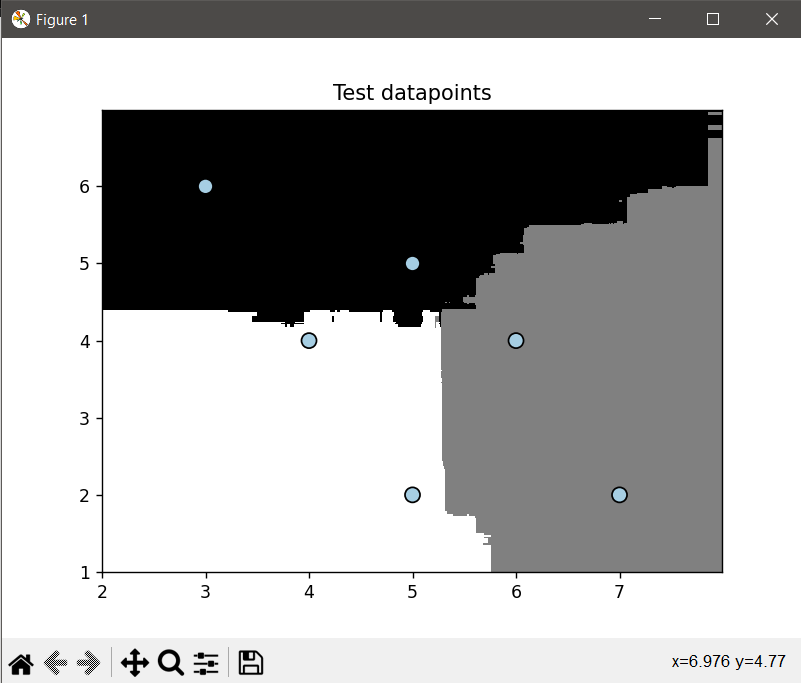
**Результат виконання:**



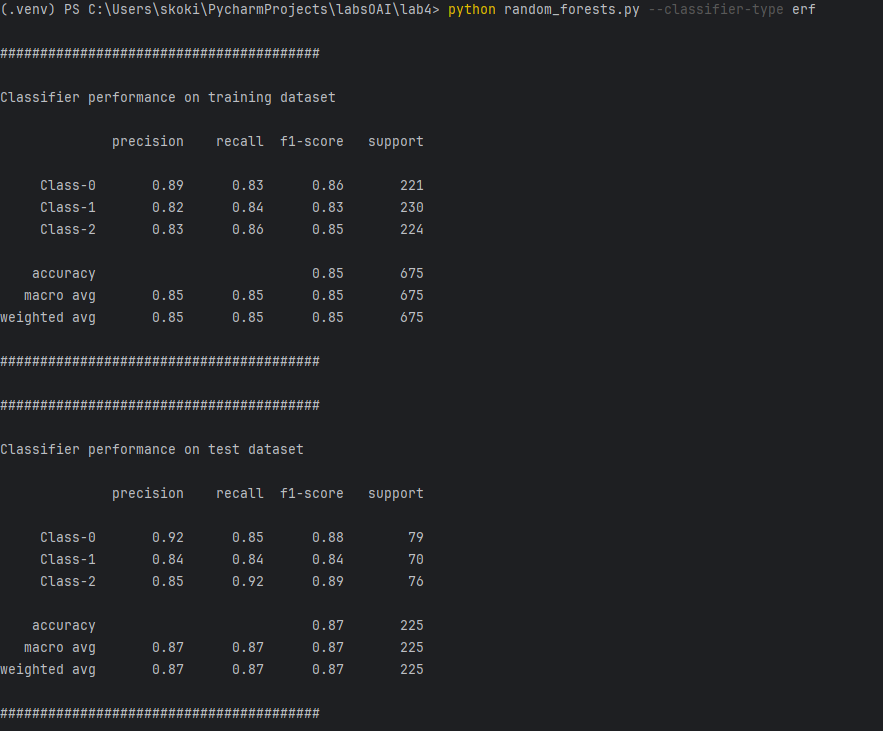


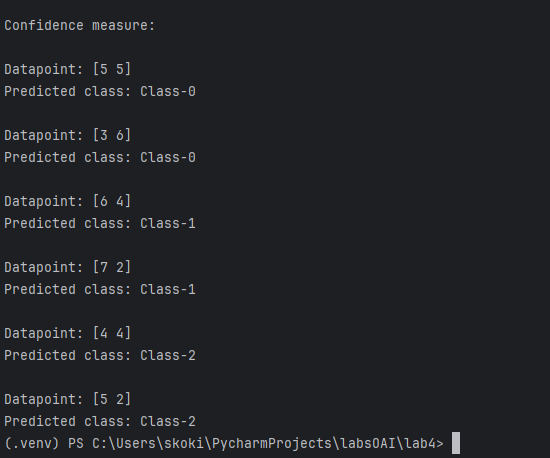
Створення класифікатора на основі випадкового лісу

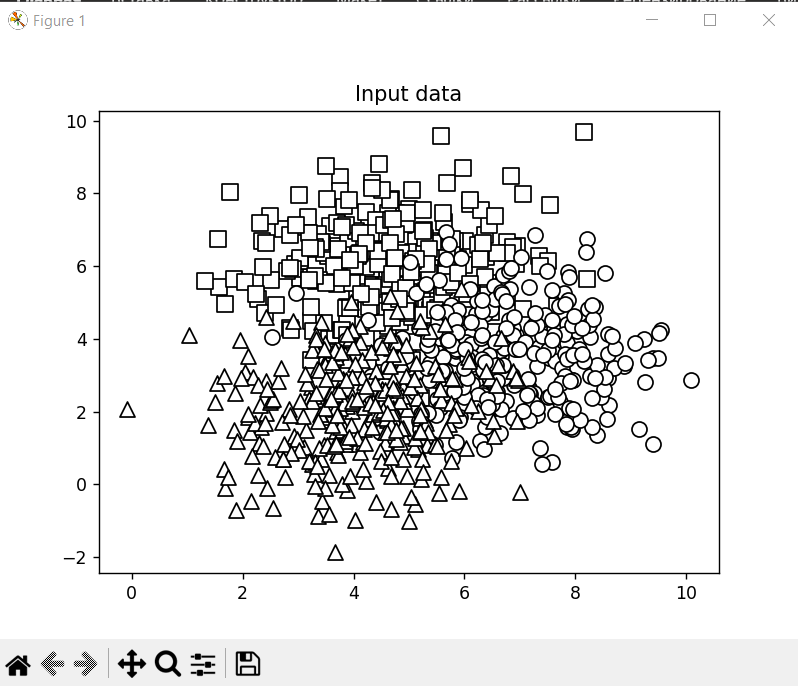
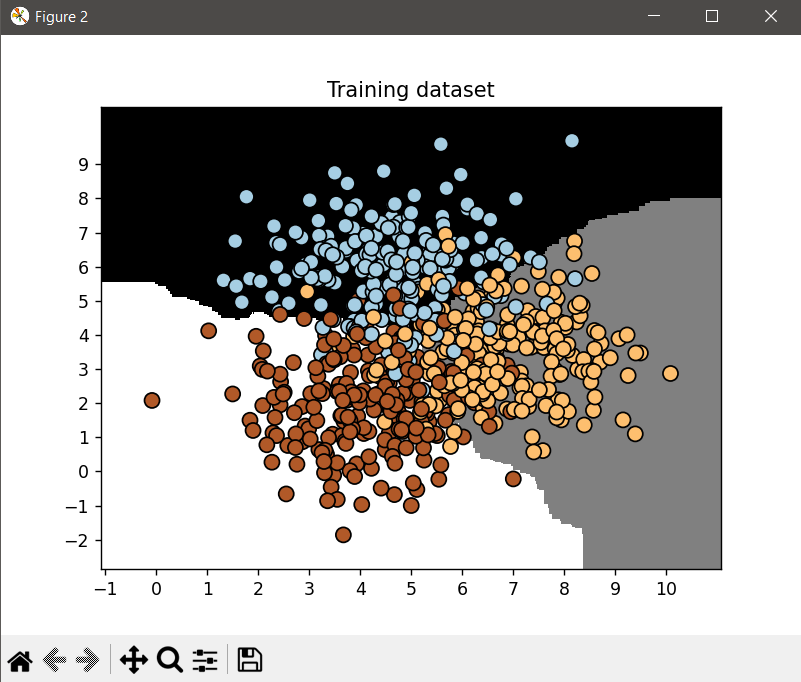
****

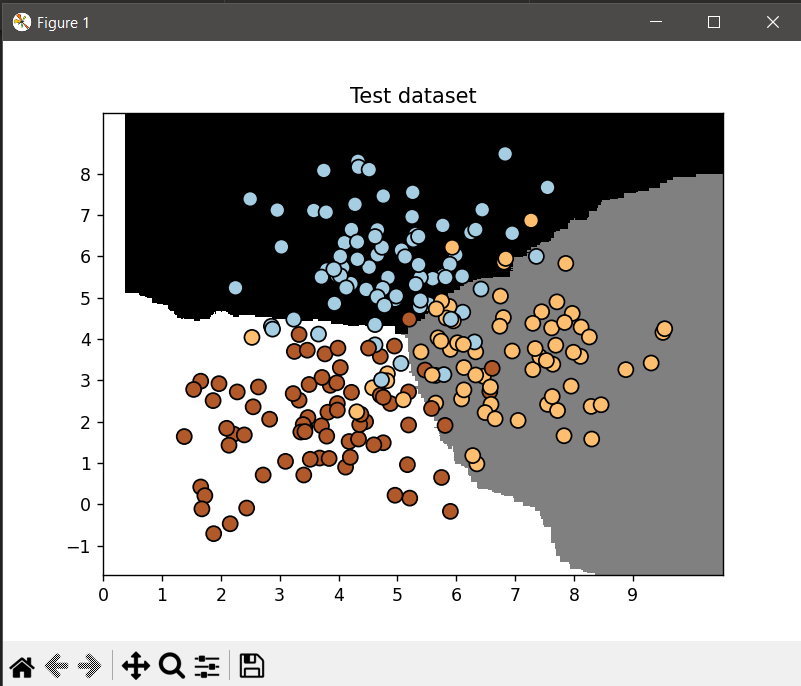
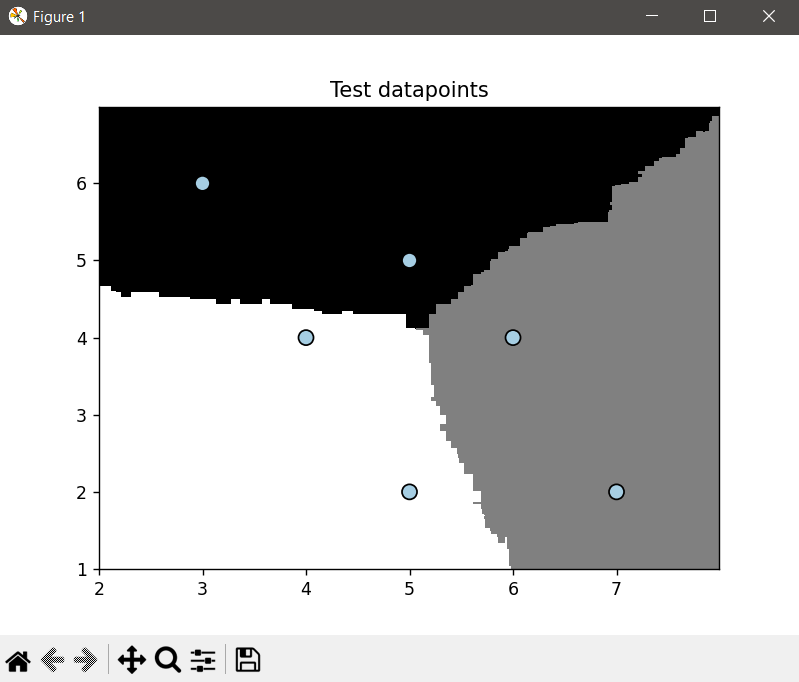
**** 

Створення класифікатора на основі гранично випадкового лісу:



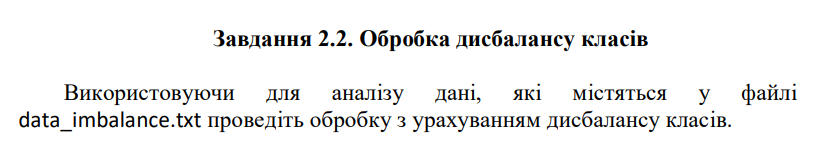


**Висновок:** У даному завданні було використано методи Random Forest та Extra Trees для класифікації даних і отримано такі значення:

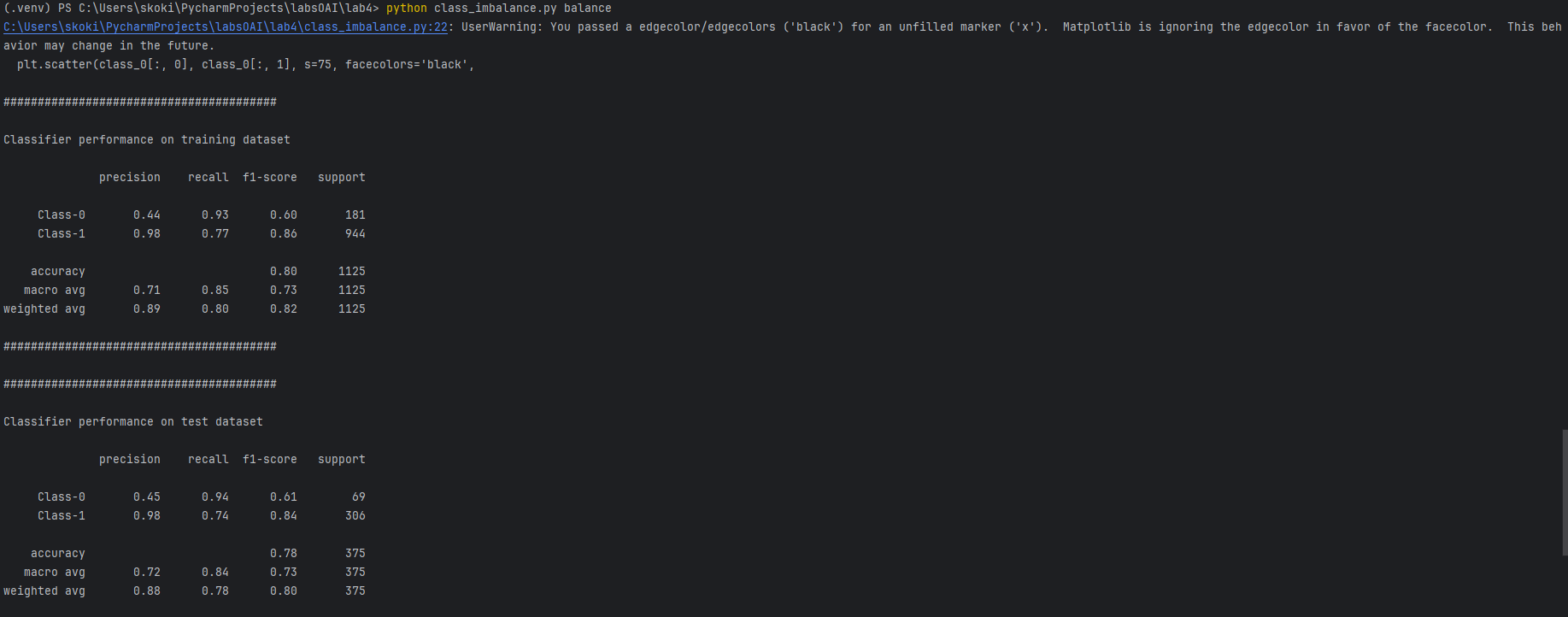
При використанні RF:  
Класифікатор досягає точності 0.87,що дорівнює приблизно 87%(тестовий набор 87%)  
При використанні ERF:  
Класифікатор досягає точності 0.85,що дорівнює приблизно 85%(тестовий набор 85%)

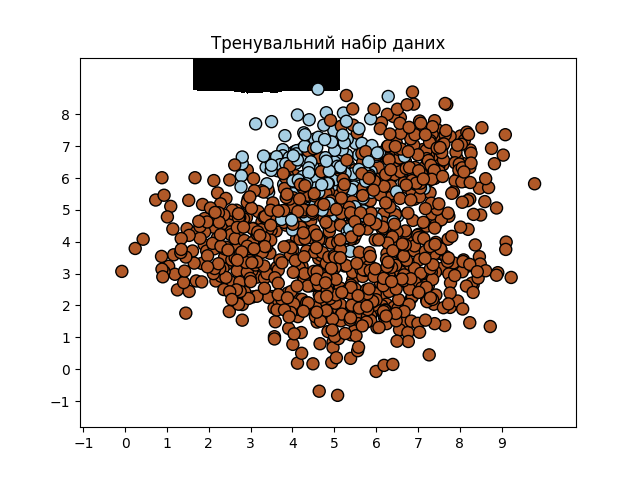
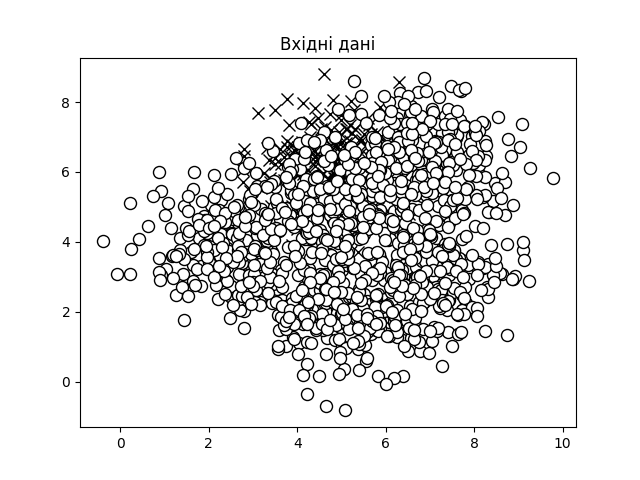


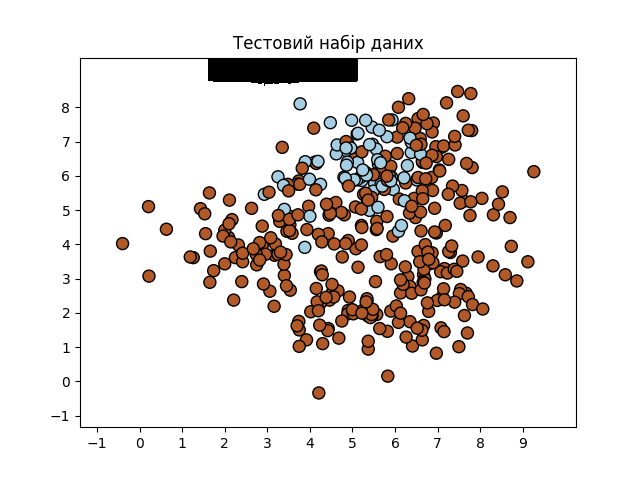
**Лістинг програми:**

import sys  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report  
from utilities import visualize\_classifier  
  
input\_file = 'data\_imbalance.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
class\_0 = np.array(X[y == 0])  
class\_1 = np.array(X[y == 1])  
  
plt.figure()  
plt.scatter(class\_0[:, 0], class\_0[:, 1], s=75, facecolor='black',  
 edgecolors='black', linewidths=1, marker='x')  
  
plt.scatter(class\_1[:, 0], class\_1[:, 1], s=75, facecolor='white',  
 edgecolors='black', linewidths=1, marker='o')  
  
plt.title('Вхідні дані')  
  
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}  
if len(sys.argv) > 1:  
 if sys.argv[1] == 'balance':  
 params = {'n\_estimators': 100, 'max\_depth': 4, 'random\_state': 0, 'class\_weight': 'balanced'}  
 else:  
 raise TypeError('Invalid input argument; should be \'balance\'')  
  
classifier = ExtraTreesClassifier(\*\*params)  
classifier.fit(X\_train, y\_train)  
visualize\_classifier(classifier, X\_train, y\_train, 'Тренувальний набір даних')  
  
y\_test\_pred = classifier.predict(X\_test)  
visualize\_classifier(classifier, X\_test, y\_test, 'Тестовий набір даних')  
  
class\_names = ['Class-0', 'Class-1']  
print("\n" + "#" \* 40)  
print("\nClassifier performance on training dataset\n")  
print(classification\_report(y\_train, classifier.predict(X\_train), target\_names=class\_names))  
print("#" \* 40 + "\n")  
  
print("#" \* 40)  
print("\nClassifier performance on test dataset\n")  
print(classification\_report(y\_test, y\_test\_pred, target\_names=class\_names))  
print("#" \* 40 + "\n")  
  
plt.show()

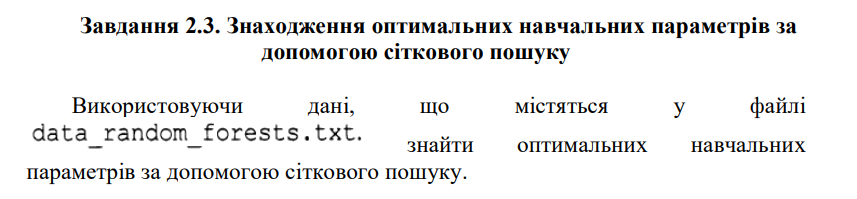
**Результат виконання:**



****

****

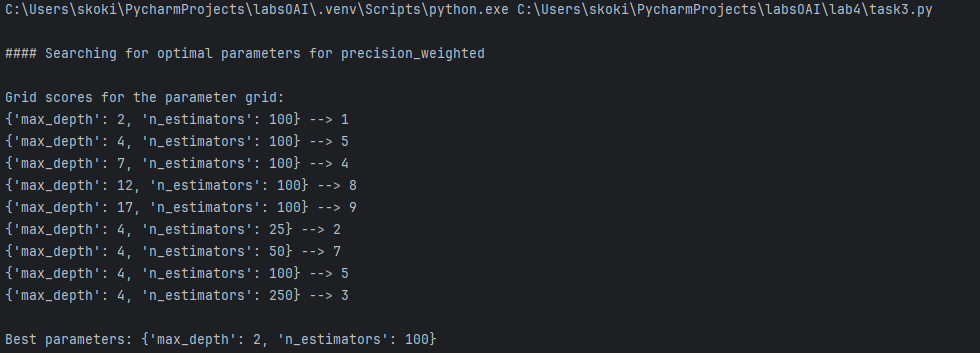
**Висновок:** У даному завданні було використано класифікатор для даних з урахуванням дисбалансу класів,на навчальному та тестовому наборах було отримано точність 0.78(близько 78%).

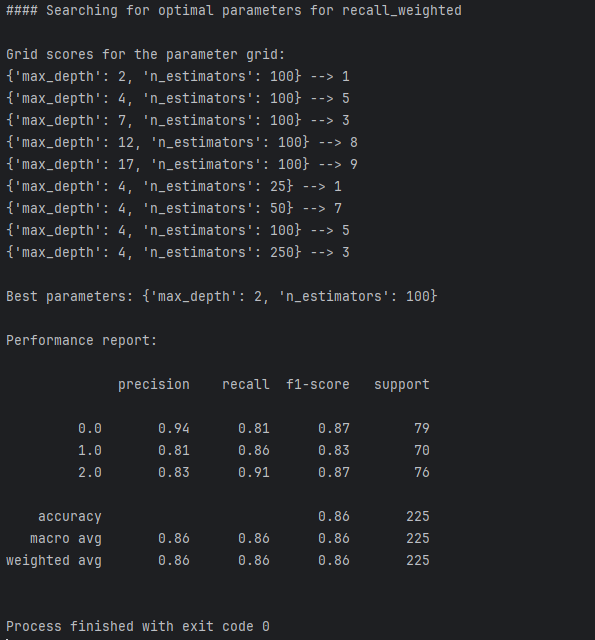


**Лістинг програми:**

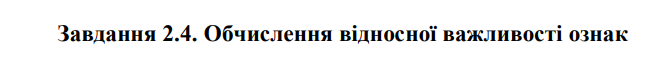
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import classification\_report  
from sklearn.model\_selection import GridSearchCV  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.model\_selection import train\_test\_split  
  
#Використовуємо для нашого аналізу дані, що містяться у файлі  
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, 17]},  
 {'max\_depth': [4], 'n\_estimators': [25, 50, 100, 250]}]  
  
metrics = ['precision\_weighted', 'recall\_weighted']  
  
for metric in metrics:  
 print('\n#### Searching for optimal parameters for', metric)  
  
 classifier = GridSearchCV(ExtraTreesClassifier(random\_state=0), parameter\_grid, cv=5, scoring=metric)  
 classifier.fit(X\_train, y\_train)  
   
 #Виведемо оцінку для кожної комбінації параметрів.  
 print('\nGrid scores for the parameter grid:')  
 for i in range(0, len(classifier.cv\_results\_['params'])):  
 print(classifier.cv\_results\_['params'][i], '-->', classifier.cv\_results\_['rank\_test\_score'][i])  
 print('\nBest parameters:', classifier.best\_params\_)  
#Виведемо звіт із результатами роботи класифікатора  
y\_pred = classifier.predict(X\_test)  
print('\nPerformance report:\n')  
print(classification\_report(y\_test, y\_pred))

**Результат виконання:**





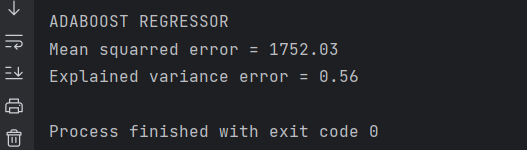
**Висновок:** У даному завданні було використано GridSearchCV для пошуку оптимальних параметрів для класифікатора ExtraTreesClassifier на основі двох метрик - **precision\_weighted** і **recall\_weighted**. Класифікатор показав високу продуктивність(точність близко 86%).

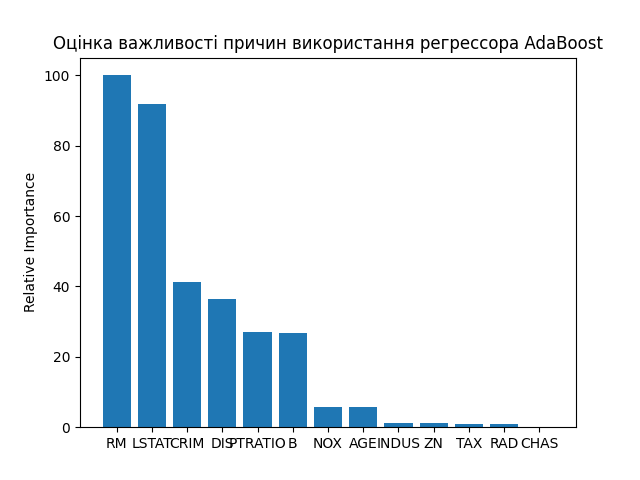


Лістинг програми:

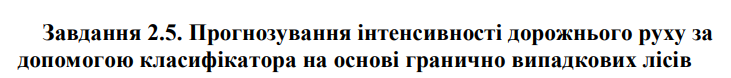
from ctypes import util  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import AdaBoostClassifier  
from sklearn import datasets, preprocessing, utils  
from sklearn.metrics import mean\_squared\_error, explained\_variance\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.utils import shuffle  
  
  
#Завантаження даних із цінами на нерухомість  
housing\_data = datasets.load\_boston()  
print(housing\_data['DESCR'])  
#Перемішування даних  
label\_encoder = preprocessing.LabelEncoder()  
X, y = shuffle(housing\_data.data, label\_encoder.fit\_transform(housing\_data.target), random\_state=7)  
  
#Розбиття 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=7)  
  
# Модель на основі регресора AdaBoost  
regressor = AdaBoostClassifier(DecisionTreeClassifier(max\_depth=4), n\_estimators=400, random\_state=7)  
regressor.fit(X\_train, y\_train)  
  
# Обчислення показників ефективності регресора AdaBoost  
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 squarred error =', round(mse, 2))  
print('Explained variance error =', round(evs, 2))  
  
# Вилучення важливості ознак  
feature\_importances = regressor.feature\_importances\_  
feature\_names = housing\_data.feature\_names  
  
# Нормалізація значень важливості ознак  
feature\_importances = 100.0 \* (feature\_importances / max(feature\_importances))  
  
# Сортування та перестановка значень  
index\_sorted = np.flipud(np.argsort(feature\_importances))  
# Розміщення міток уздовж осі Х  
pos = np.arange(index\_sorted.shape[0]) + 0.5  
  
# Побудова стовпчастої діаграми  
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 причин використання регрессора AdaBoost')  
plt.show()

**Результат виконання:**





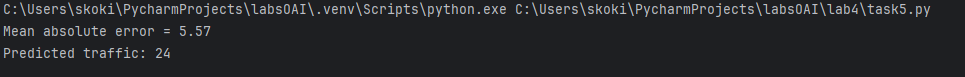
Висновок: Було використано регресор AdaBoost для цін нерухомості,велика величина помилки становить 1752.03,що означає те,що модель не є точною,значення EVS дорівнює 0.56,це означає що моделю пояснює 56% змінності.Ознаки такі як: RM,LSTATCRIM мають найбільші показники,найменші показники мають ознаки:ZN,NAX,RAD,ознака CHAS взагалі немає ніякого значення.



Лістинг програми:

import numpy as np  
from sklearn.metrics import mean\_absolute\_error  
from sklearn import preprocessing  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.model\_selection import train\_test\_split  
  
#Завантажимо дані із файлу traffic\_data.txt.  
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)  
  
# Перетворення рядкових даних на числові  
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)  
  
# 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)  
  
# Регресор на основі гранично випадкових лісів  
params = {'n\_estimators': 200, 'max\_depth': 15, 'random\_state': 0}  
regressor = ExtraTreesClassifier(\*\*params)  
regressor.fit(X\_train, y\_train)  
  
# Обчислення характеристик ефективності регресора на тестових дани  
y\_pred = regressor.predict(X\_test)  
print('Mean absolute error =', round(mean\_absolute\_error(y\_test, y\_pred), 2))  
  
# Тестування кодування на одиночному прикладі  
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:  
 encoder = label\_encoder[count]  
 test\_datapoint\_encoded[i] = int(encoder.transform([test\_datapoint[i]])[0])  
 count = count + 1  
  
test\_datapoint\_encoded = np.array(test\_datapoint\_encoded)  
  
# Прогнозування результату для тестової точки даних  
print('Predicted traffic:', int(regressor.predict([test\_datapoint\_encoded])[0]))

**Результат виконання:**

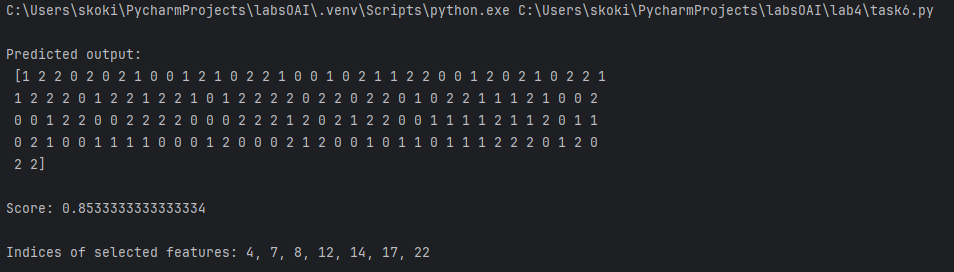


****

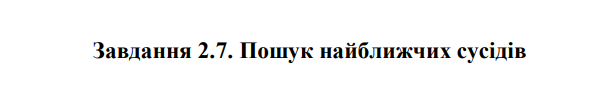
**Лістинг програми:**

from sklearn.datasets import \_samples\_generator  
from sklearn.feature\_selection import SelectKBest, f\_regression  
from sklearn.pipeline import Pipeline  
from sklearn.ensemble import ExtraTreesClassifier  
  
# Генерація даних  
X, y = \_samples\_generator.make\_classification(n\_samples=150,  
 n\_features=25, n\_classes=3, n\_informative=6,  
 n\_redundant=0, random\_state=7)  
  
# Вибір k найважливіших ознак  
k\_best\_selector = SelectKBest(f\_regression, k=9)  
  
# Ініціалізація класифікатора на основі гранично випадкового лісу  
classifier = ExtraTreesClassifier(n\_estimators=60, max\_depth=4)  
  
# Створення конвеєра  
processor\_pipeline = Pipeline([('selector', k\_best\_selector), ('erf', classifier)])  
  
# Встановлення параметрів  
processor\_pipeline.set\_params(selector\_\_k=7, erf\_\_n\_estimators=30)  
  
# Навчання конвеєра  
processor\_pipeline.fit(X, y)  
  
# Прогнозування результатів для вхідних даних  
output = processor\_pipeline.predict(X)  
print("\nPredicted output:\n", output)  
  
# Виведення оцінки  
print("\nScore:", processor\_pipeline.score(X, y))  
  
# Виведення ознак, відібраних селектором конвеєра  
status = processor\_pipeline.named\_steps['selector'].get\_support()  
  
# Вилучення та виведення індексів обраних ознак  
selected = [i for i, x in enumerate(status) if x]  
print("\nIndices of selected features:", ', '.join([str(x) for x in selected]))

**Результат виконання:**



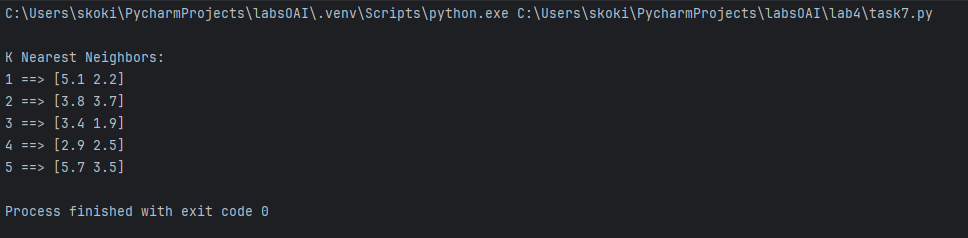
**Висновок:** Перший список - Це передбачені класи для кожного вхідного прикладу після застосування класифікатора на основі гранично випадкового лісу,кожне число представляє клас.Значення Score – оцінка точності. В останньому рядку представлені індекси ознак які були вибрані як найважливіші для класифікатора.

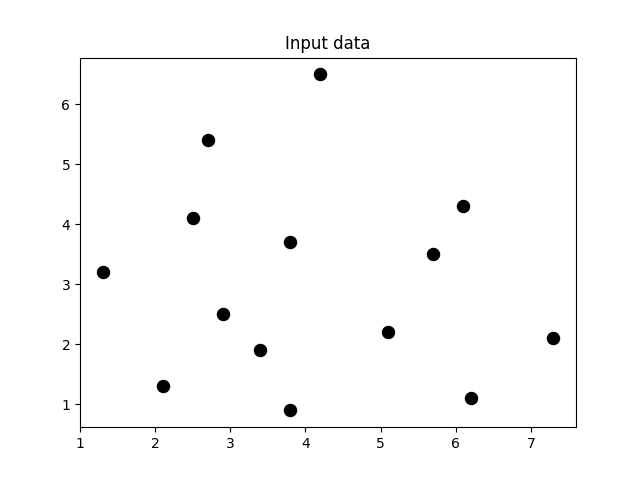


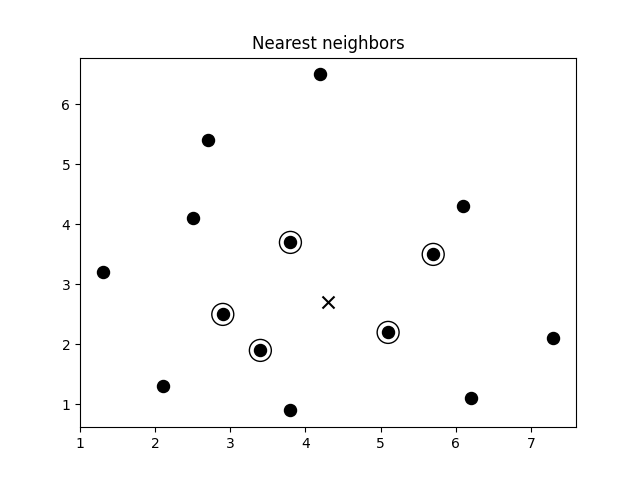
**Лістинг програми:**

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.neighbors import NearestNeighbors  
  
# Вхідні дані  
X = np.array([[2.1, 1.3], [1.3, 3.2], [2.9, 2.5], [2.7, 5.4], [3.8, 0.9],  
 [7.3, 2.1], [4.2, 6.5], [3.8, 3.7], [2.5, 4.1], [3.4, 1.9],  
 [5.7, 3.5], [6.1, 4.3], [5.1, 2.2], [6.2, 1.1]])  
  
# Кількість найближчих сусідів  
k = 5  
# Тестова точка даних  
test\_datapoint = [4.3, 2.7]  
  
# Відображення вхідних даних на графіку  
plt.figure()  
plt.title('Input data')  
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='black')  
  
# Побудова моделі на основі методу k найближчих сусідів  
knn\_model = NearestNeighbors(n\_neighbors=k, algorithm='ball\_tree').fit(X)  
distances, indices = knn\_model.kneighbors([test\_datapoint])  
  
# Виведемо 'k' найближчих сусідів  
print("\nK Nearest Neighbors:")  
for rank, index in enumerate(indices[0][:k], start=1):  
 print(str(rank) + " ==>", X[index])  
  
# Візуалізація найближчих сусідів разом із тестовою точкою даних  
plt.figure()  
plt.title('Nearest neighbors')  
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='k')  
plt.scatter(X[indices][0][:][:, 0], X[indices][0][:][:, 1],  
 marker='o', s=250, color='k', facecolors='none')  
plt.scatter(test\_datapoint[0], test\_datapoint[1],  
 marker='x', s=75, color='k')  
plt.show()

**Результат виконання:**



****

****

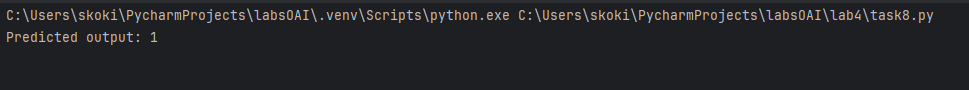
**Висновок:** На першому графіку відображені вхідні дані(набір точок у двовимірному просторі). На другому графіку показані найближчі сусіди до тестової точки([4.3, 2.7]).В терміналі виводиться список найближчик сусідів для тестової точки разом з їх координатами



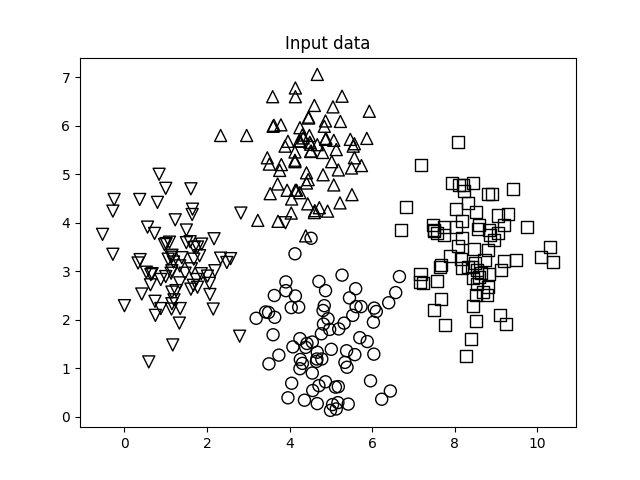
**Лістинг програми:**

import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.cm as cm  
from sklearn import neighbors  
  
# Завантаження вхідних даних  
input\_file = 'data.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1].astype(np.int\_)  
  
# Відображення вхідних даних на графіку  
plt.figure()  
plt.title('Input data')  
marker\_shapes = 'v^os'  
mapper = [marker\_shapes[i] for i in y]  
for i in range(X.shape[0]):  
 plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],  
 s=75, edgecolors='black', facecolors='none')  
  
# Кількість найближчих сусідів  
num\_neighbors = 12  
  
# Розмір кроку сітки візуалізації  
step\_size = 0.01  
  
# Створення класифікатора на основі методу k найближчих сусідів  
classifier = neighbors.KNeighborsClassifier(num\_neighbors, weights='distance')  
  
# Навчання моделі на основі методу k найближчих сусідів  
classifier.fit(X, y)  
  
# Створення сітки для відображення меж на графіку  
x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1  
y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1  
x\_values, y\_values = np.meshgrid(np.arange(x\_min, x\_max, step\_size), np.arange(y\_min, y\_max, step\_size))  
  
# Виконання класифікатора на всіх точках сітки  
output = classifier.predict(np.c\_[x\_values.ravel(), y\_values.ravel()])  
  
# Візуалізація передбачуваного результату  
output = output.reshape(x\_values.shape)  
plt.figure()  
plt.pcolormesh(x\_values, y\_values, output, cmap=cm.Paired)  
  
# Накладання навчальних точок на карту  
for i in range(X.shape[0]):  
 plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],  
 s=50, edgecolors='black', facecolors='none')  
  
plt.xlim(x\_values.min(), x\_values.max())  
plt.ylim(y\_values.min(), y\_values.max())  
plt.title('K Nearest Neighbors classifier model boundaries')  
  
# Тестування вхідної точки даних  
test\_datapoint = [5.1, 3.6]  
plt.figure()  
plt.title('Test datapoint')  
for i in range(X.shape[0]):  
 plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],  
 s=75, edgecolors='black', facecolors='none')  
  
plt.scatter(test\_datapoint[0], test\_datapoint[1], marker='x',  
 linewidth=6, s=200, facecolors='black')  
  
# Вилучення K найближчих сусідів  
\_, indices = classifier.kneighbors([test\_datapoint])  
indices = indices.astype(np.int\_)[0]  
  
# Відображення K найближчих сусідів на графіку  
plt.figure()  
plt.title('K Nearest Neighbors')  
  
for i in indices:  
 plt.scatter(X[i, 0], X[i, 1], marker=mapper[y[i]],  
 linewidth=3, s=100, facecolors='black')  
  
plt.scatter(test\_datapoint[0], test\_datapoint[1], marker='x',  
 linewidth=6, s=200, facecolors='black')  
  
for i in range(X.shape[0]):  
 plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],  
 s=75, edgecolors='black', facecolors='none')  
  
print("Predicted output:", classifier.predict([test\_datapoint])[0])  
  
plt.show()

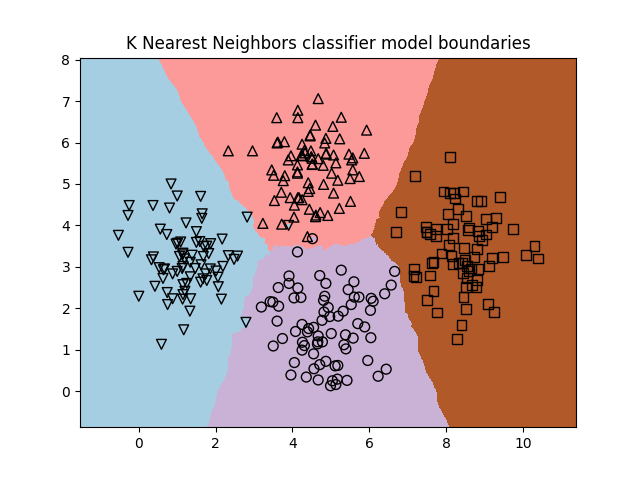
**Результат виконання:**

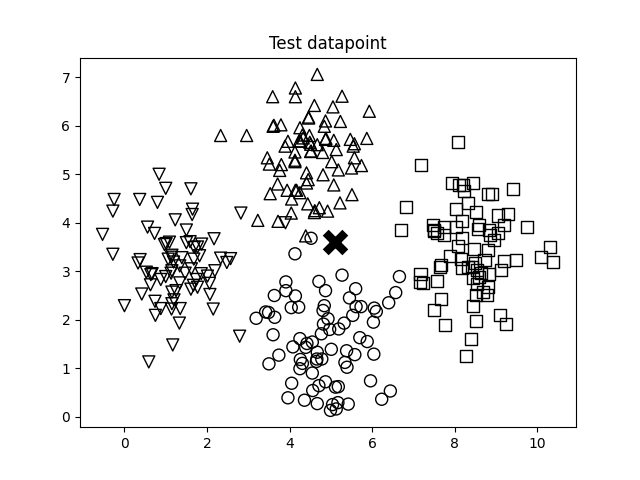


№1.Вхідні дані:

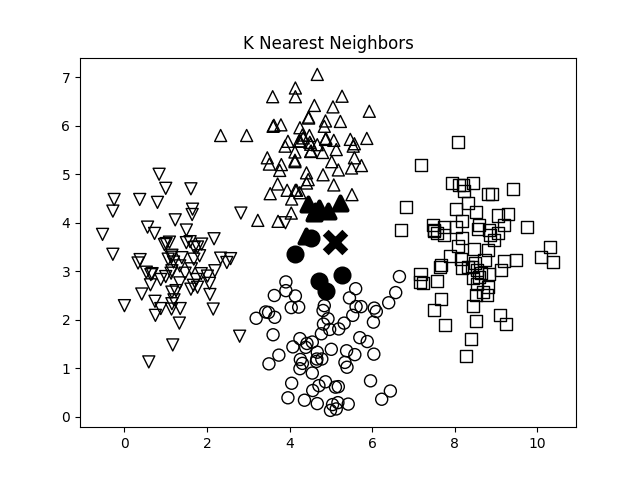


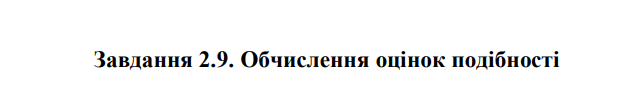
№2. Відображені межі,які були побудовані за доп. Моделі k-найближчих сусідів:



№3.Відображені вхідні дані разом з тестовою точкою:  


№4. Показані K-найближчих сусідів тестової точки:

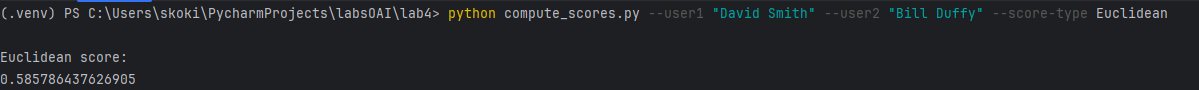




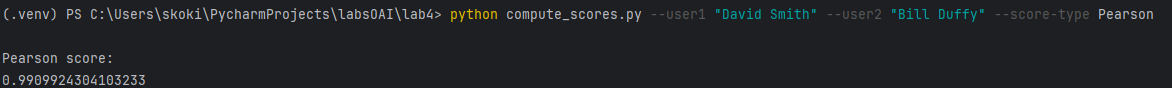
**Лістинг програми:**

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  
  
  
# Обчислення оцінки евклідова відстані між користувачами user1 та user2  
def euclidean\_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, user1 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)))  
# Обчислення кореляційної оцінки Пірсона між користувачем1 і користувачем2  
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, user1 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 кори-стувачами  
 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 користува-чами  
 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))

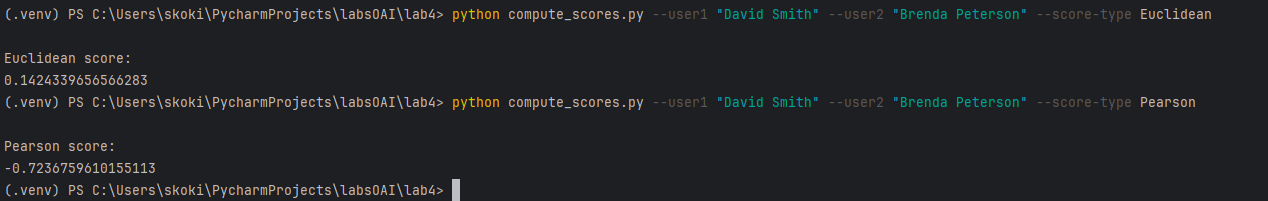
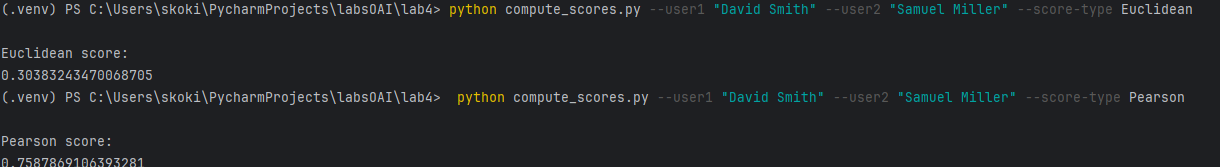
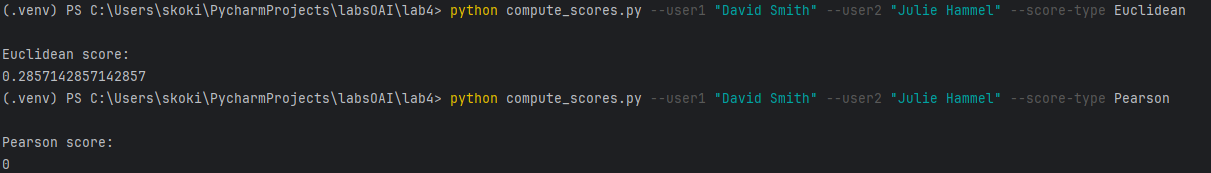
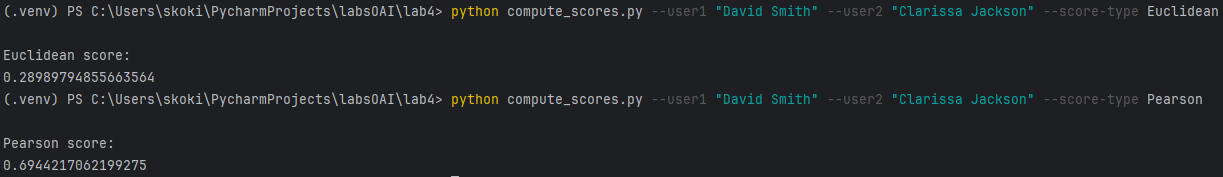
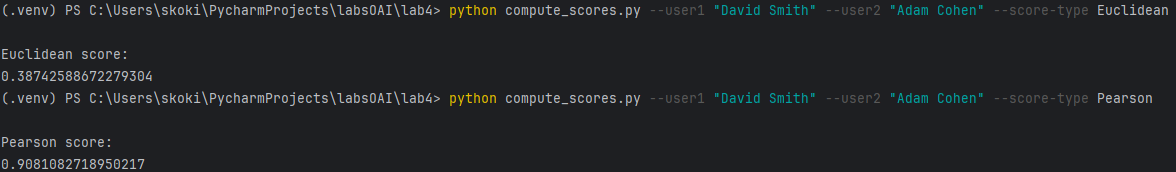
**Результат виконання:**

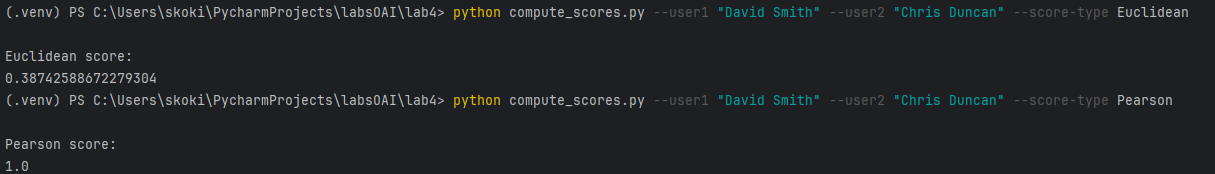
Обчислення евклідової оцінки:

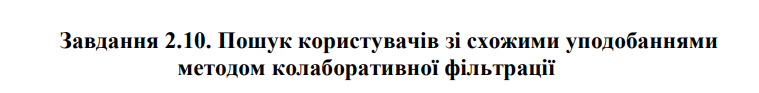
Обчислення оцінки подібності:



Аналогічні команди для:

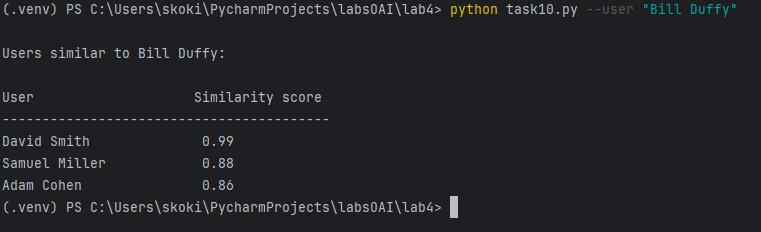
1. David Smith та Brenda Peterson
2. David Smith та Samuel Miller  
   
3. David Smith та Julie Hammel  
   
4. David Smith та Clarissa Jackson  
   
5. David Smith та Adam Cohen  
   
6. David Smith та Chris Duncan

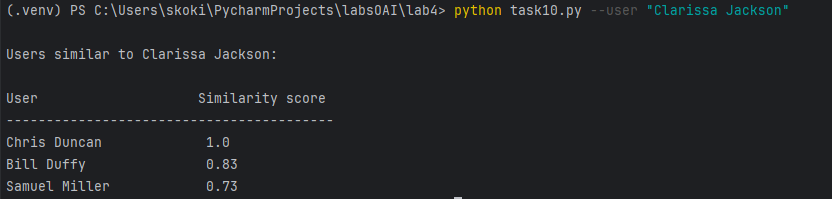




**Лістинг програми:**

import argparse  
import json  
import numpy as np  
  
from LR\_4\_task\_9 import pearson\_score  
  
  
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(description='Find users who are similar to the in-put user')  
 parser.add\_argument('--user', dest='user', required=True,  
 help='Input user')  
 return parser  
  
  
# Знаходження користувачів у наборі даних, схожих на введеного користувача  
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))

**Результат виконання:**



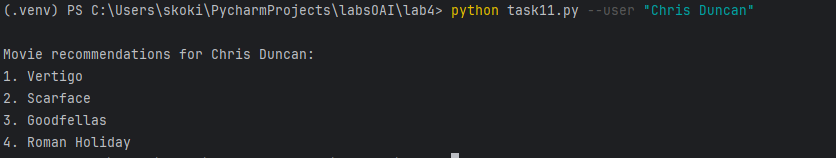
**Висновок:** У даному завданні ми знаходимо користувачів, схожих на введеного користувача, в наборі даних з рейтингами. Результати показують користувачів з найвищими оцінками подібності.



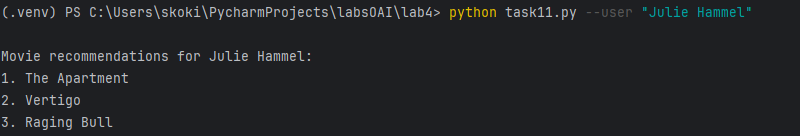
**Лістинг програми:**

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  
  
  
# Отримання 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)

**Результат виконання:**

Chris Duncan:  


Julie Hammel:



**Висновок до лабораторної роботи:** Під час виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python було досліджено методи ансамблів у машинному навчанніта створити рекомендаційні системи.