**ЛАБОРАТОРНА РОБОТА № 4**

***Тема:*** ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА

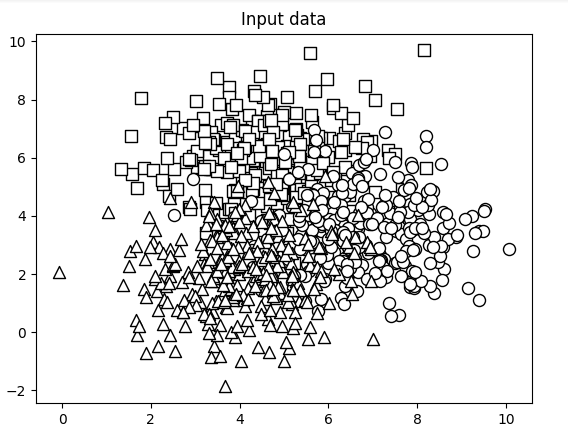
СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

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

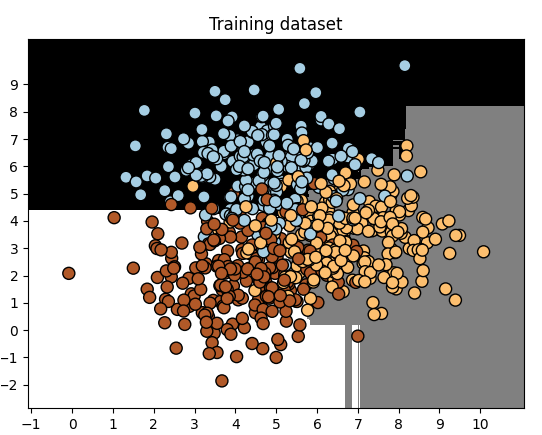
**Завдання №1:** Створення класифікаторів на основі випадкових та гранично випадкових лісів.

import argparse  
  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import classification\_report  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier  
from sklearn.metrics import classification\_report  
  
from utilities import visualize\_classifier  
  
  
# Argument parser  
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\_\_':  
 # Parse the input arguments  
 args = build\_arg\_parser().parse\_args()  
 classifier\_type = args.classifier\_type  
  
 # Load input data  
 input\_file = 'data\_random\_forests.txt'  
 data = np.loadtxt(input\_file, delimiter=',')  
 X, y = data[:, :-1], data[:, -1]  
 # Separate input data into three classes based on labels  
 class\_0 = np.array(X[y == 0])  
 class\_1 = np.array(X[y == 1])  
 class\_2 = np.array(X[y == 2])

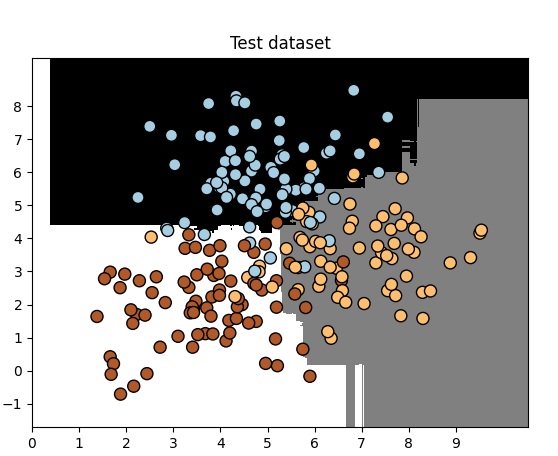
# Visualize input data  
 plt.figure()  
 plt.scatter(class\_0[:, 0], class\_0[:, 1], s=75, facecolors='white',  
 edgecolors='black', linewidth=1, marker='s')  
 plt.scatter(class\_1[:, 0], class\_1[:, 1], s=75, facecolors='white',  
 edgecolors='black', linewidth=1, marker='o')  
 plt.scatter(class\_2[:, 0], class\_2[:, 1], s=75, facecolors='white',  
 edgecolors='black', linewidth=1, marker='^')  
 plt.title('Input data')  
  
 # Split data into training and testing datasets  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.25, random\_state=5)  
  
 # Ensemble Learning classifier  
 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, 'Training dataset')  
  
 y\_test\_pred = classifier.predict(X\_test)  
 visualize\_classifier(classifier, X\_test, y\_test, 'Test dataset')  
  
 # Evaluate classifier performance  
 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), tar-get\_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")  
  
 # Compute confidence  
 test\_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2], [4, 4], [5, 2]])  
  
 print("\nConfidence 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 the datapoints  
 visualize\_classifier(classifier, test\_datapoints, [0] \* len(test\_datapoints), 'Test datapoints')  
  
 plt.show()



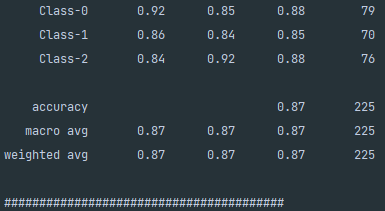
**Рис.1** Результат виконання програми



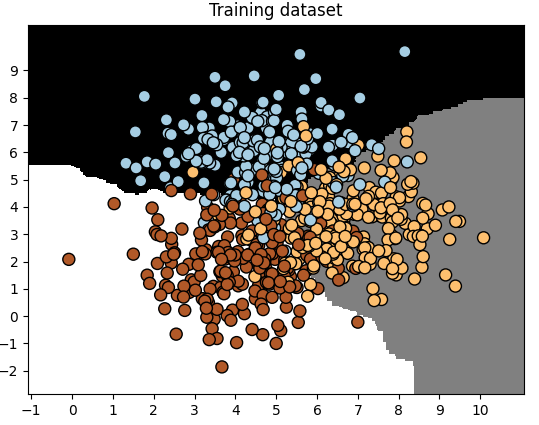
**Рис.2** Результат виконання програми



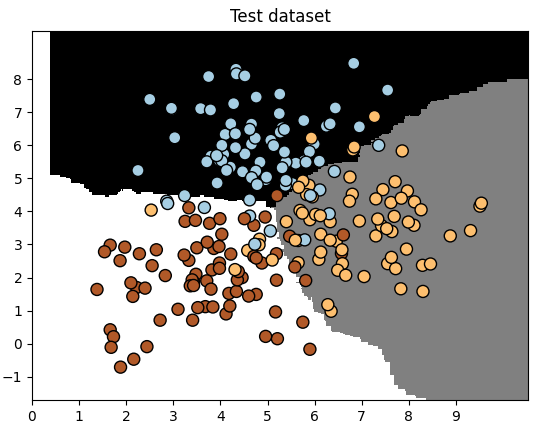
**Рис.3** Результат виконання програми



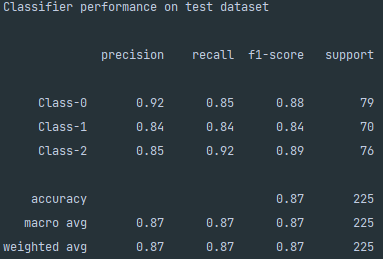
**Рис.4** Результат виконання програми



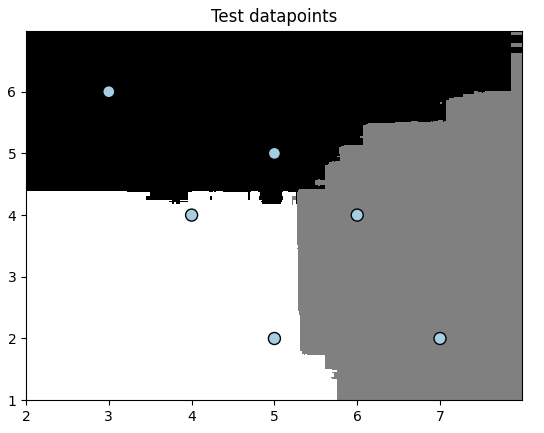
**Рис.5** Результат виконання програми



**Рис.6** Результат виконання програми



**Рис.7** Результат виконання програми

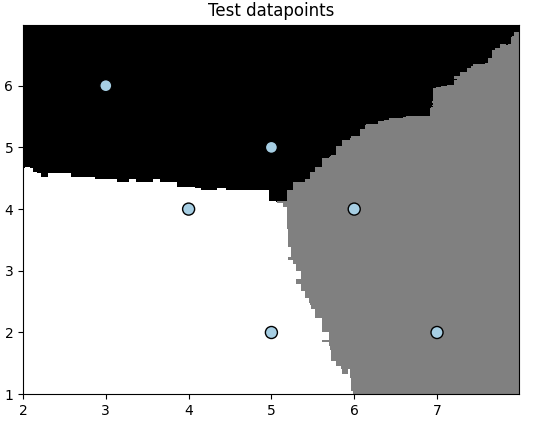


**Рис.8** Результат виконання програми

Зображення, що містить текст

Автоматично згенерований опис

**Рис.9** Результат виконання програми



**Рис.10** Результат виконання програми

Зображення, що містить текст

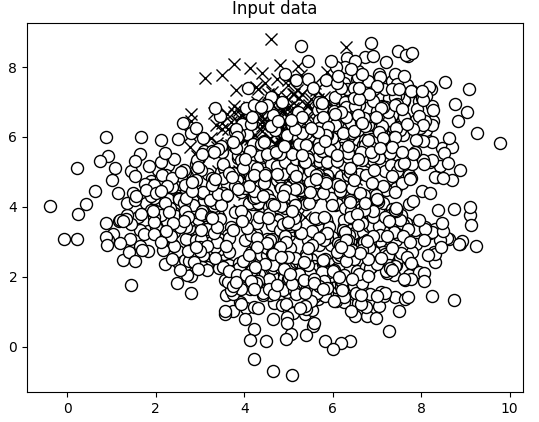
Автоматично згенерований опис

**Рис.11** Результат виконання програми

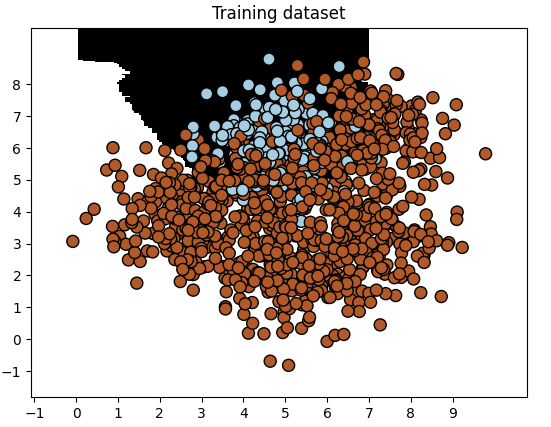
При юзs **-erf** отримав більш валідні піки. Це обумовлено тим, що в процесі навчання гранично випадкові ліси мають більше можливостей для вибору оптимальних дерев рішень, тому, як правило, вони забезпечують отримання кращих границь. Але кінцеві результати виявилися майже однаковими при використанні обох прапорців.

**Завдання №2:** Обробка дисбалансу класів.

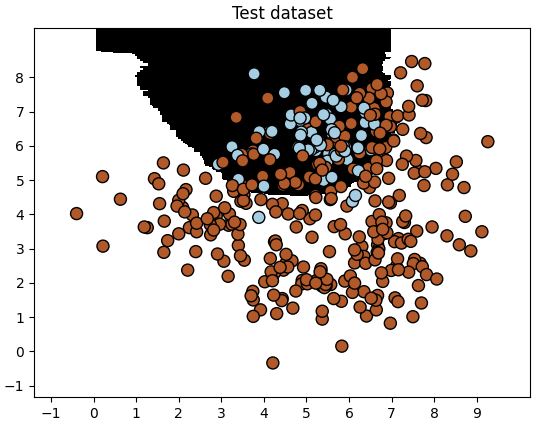
import sys  
  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report  
  
from utilities import visualize\_classifier  
  
# Завантаження вхідних даних  
input\_file = 'data\_imbalance.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1]  
  
# Поділ вхідних даних на два класи на підставі міток  
class\_0 = np.array(X[y == 0])  
class\_1 = np.array(X[y == 1])  
  
# Візуалізація вхідних даних  
plt.figure()  
plt.scatter(class\_0[:, 0], class\_0[:, 1], s=75, facecolors='black',  
 edgecolors='black', linewidth=1, marker='x')  
plt.scatter(class\_1[:, 0], class\_1[:, 1], s=75, facecolors='white',  
 edgecolors='black', linewidth=1, marker='o')  
plt.title('Input data')  
  
# 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)  
  
# Класифікатор на основі гранично випадкових лісів  
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, 'Training dataset')  
  
y\_test\_pred = classifier.predict(X\_test)  
visualize\_classifier(classifier, X\_test, y\_test, 'Test dataset')  
  
# Обчислення показників ефективності класифікатора  
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()



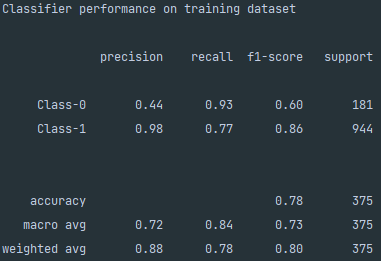
**Рис.12** Результат виконання програми



**Рис.13** Результат виконання програми



**Рис.14** Результат виконання програми



**Рис.15** Результат виконання програми

**Завдання №3:** Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку.

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  
from sklearn.metrics import classification\_report  
import pandas as pd  
  
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)  
  
 df = pd.DataFrame(classifier.cv\_results\_)  
 df\_columns\_to\_print = [column for column in df.columns if 'param' in column or 'score' in column]  
 print(df[df\_columns\_to\_print])  
  
 print("\nBest parameters:", classifier.best\_params\_)  
  
 y\_pred = classifier.predict(X\_test)  
 print("\nPerformance report:\n")  
 print(classification\_report(y\_test, y\_pred))

Зображення, що містить текст

Автоматично згенерований опис

**Рис.16** Результат виконання програми

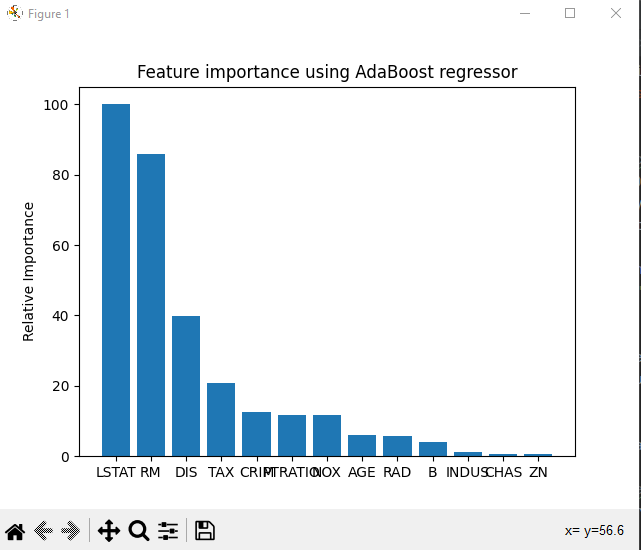
Зображення, що містить текст

Автоматично згенерований опис

**Рис.17** Результат виконання програми

**Завдання №4:** Обчислення відносної важливості ознак.

import numpy as np  
import matplotlib.pyplot as plt  
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.model\_selection import train\_test\_split  
from sklearn.utils import shuffle  
  
# Завантаження даних із цінами на нерухомість  
housing\_data = datasets.load\_boston()  
  
# Перемішування даних  
X, y = shuffle(housing\_data.data, 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 = AdaBoostRegressor(DecisionTreeRegressor(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 squared error =", round(mse, 2))  
print("Explained variance score =", 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('Feature importance using AdaBoost regressor')  
plt.show()



**Рис.18** Результат виконання програми

Зображення, що містить текст

Автоматично згенерований опис

**Рис.19** Результат виконання програми

**Завдання №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.model\_selection import train\_test\_split  
from sklearn.ensemble import ExtraTreesRegressor  
from sklearn.metrics import classification\_report  
  
# Завантаження вхідних даних  
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': 100, 'max\_depth': 4, 'random\_state': 0}  
regressor = ExtraTreesRegressor(\*\*params)  
regressor.fit(X\_train, y\_train)  
  
# Обчислення характеристик ефективності регресора на тестових даних  
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:  
 test\_datapoint\_encoded[i] = int(label\_encoder[count].transform([test\_datapoint[i]]))  
 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]))

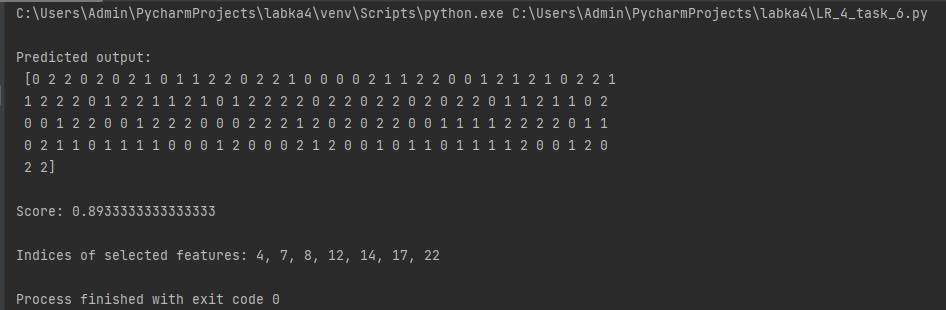
Зображення, що містить текст

Автоматично згенерований опис

**Рис.20** Результат виконання програми

**Завдання №6:** Створення навчального конвеєра (конвеєра машинного навчання).

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]))



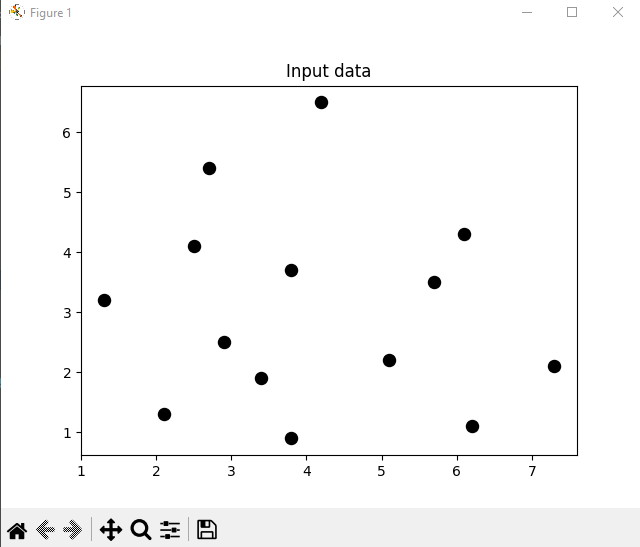
**Рис.21** Результат виконання програми

Перший абзац містить прогнозовані вихідні мітки за допомогою конвеєра. Значення Score відображає ефективність конвеєра.

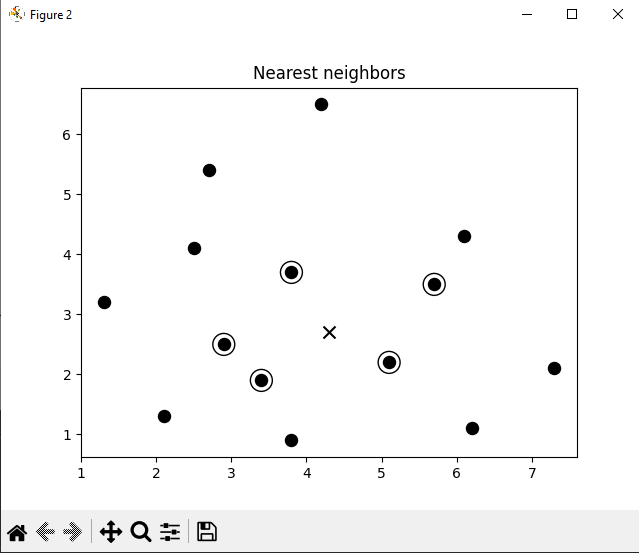
Останній абзац містить індекси вибраних ознак.

**Завдання №7:** Пошук найближчих сусідів.

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()



**Рис.22** Результат виконання програми



**Рис.23** Результат виконання програми

Зображення, що містить текст

Автоматично згенерований опис

**Рис.24** Результат виконання програми

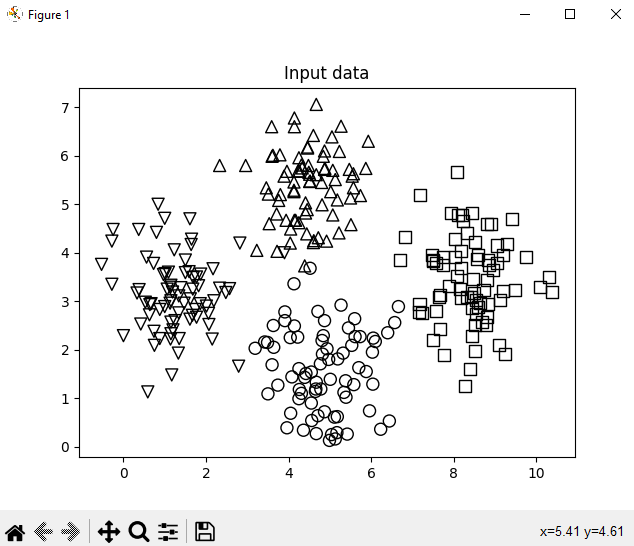
Перший скрін = вхідні дані.

Другий скрін = вхідні дані, тестову точку і її 5 найближчих сусідів. Вони обведені.

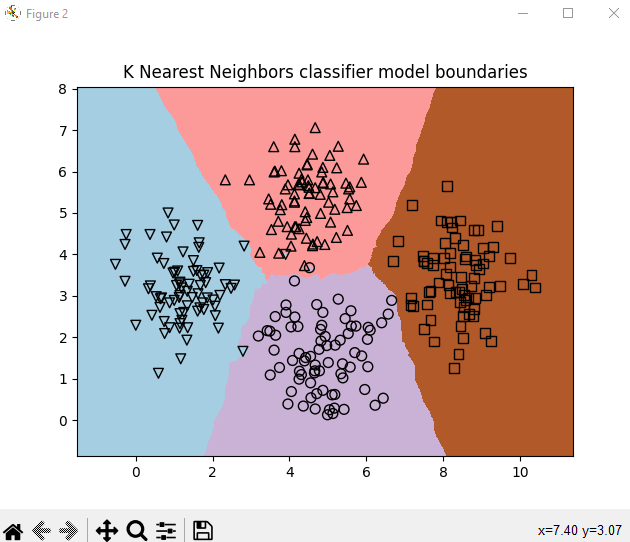
Третій скрін = 5 найближчих сусідів.

**Завдання №8:** Створити класифікатор методом k найближчих сусідів.

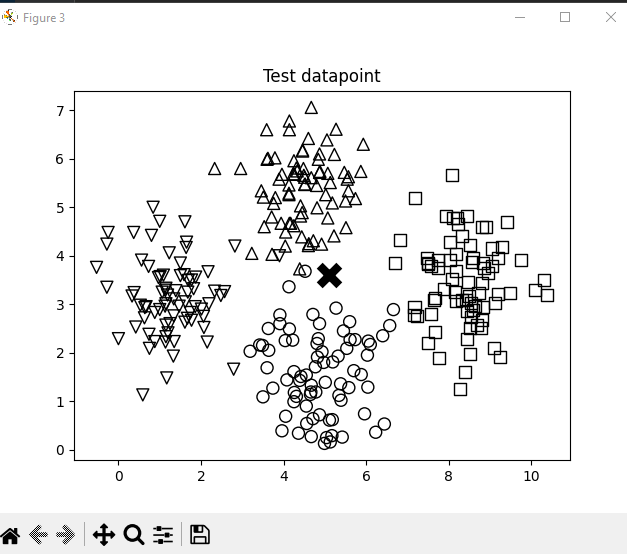
import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.cm as cm  
from sklearn import neighbors, datasets  
  
# Завантаження вхідних даних  
input\_file = 'data.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1].astype(np.int)  
  
# Відображення 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()



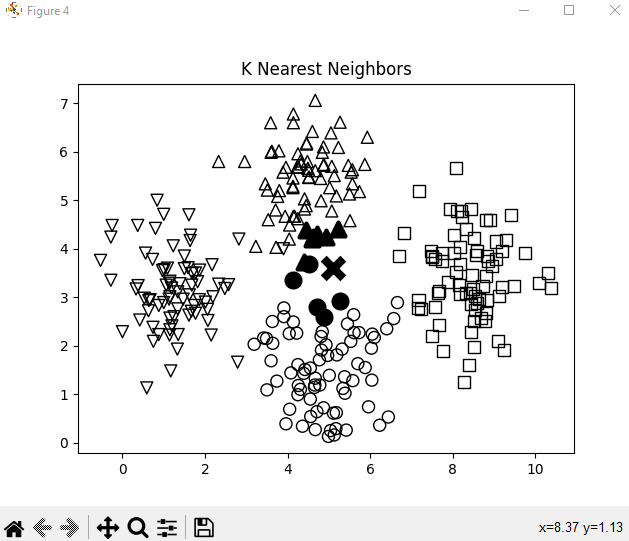
**Рис.25** Результат виконання програми



**Рис.26** Результат виконання програми



**Рис.27** Результат виконання програми



**Рис.28** Результат виконання програми



**Рис.29** Результат виконання програми

Перший скрін = вхідні дані.

Другий = межі класифікатора.

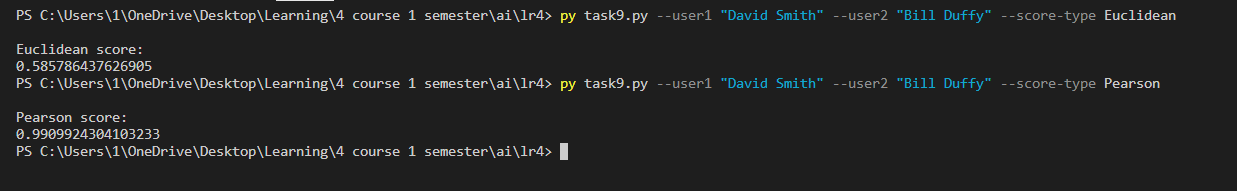
Третій = тестова точка до вхідного набору даних.

Четвертий = 12 найближчих сусідів.

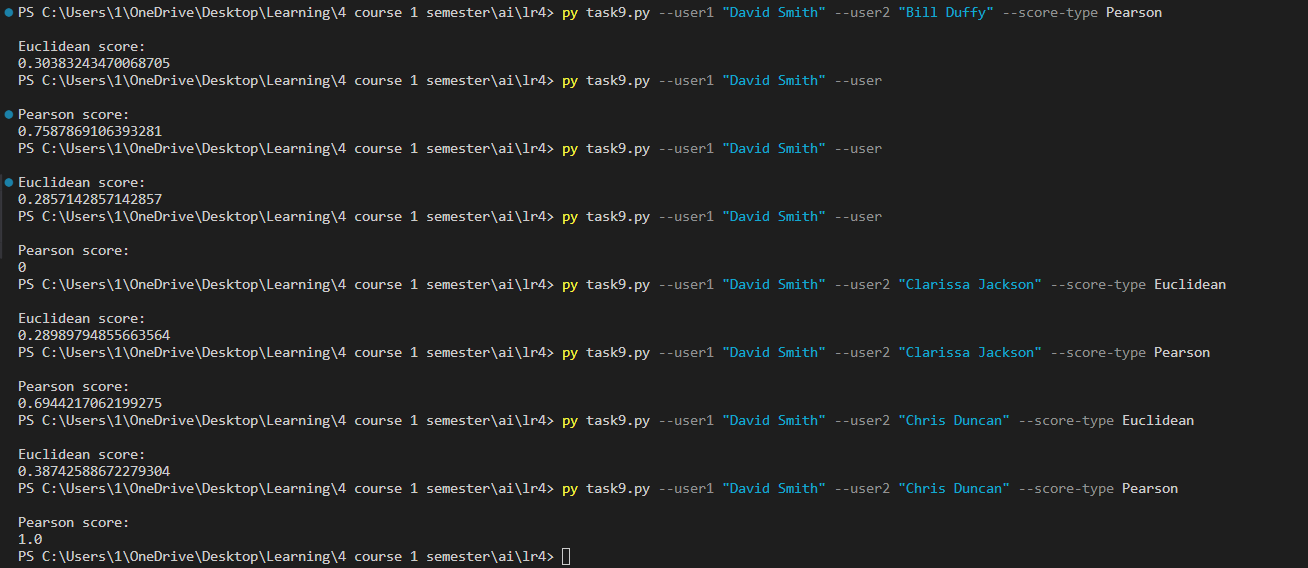
Тестова точка = 1 клас.

**Завдання №9:** Обчислення оцінок подібності.

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 userl 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, 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)))  
  
  
# 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, 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 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))

****

**Рис.30** Результат виконання програми

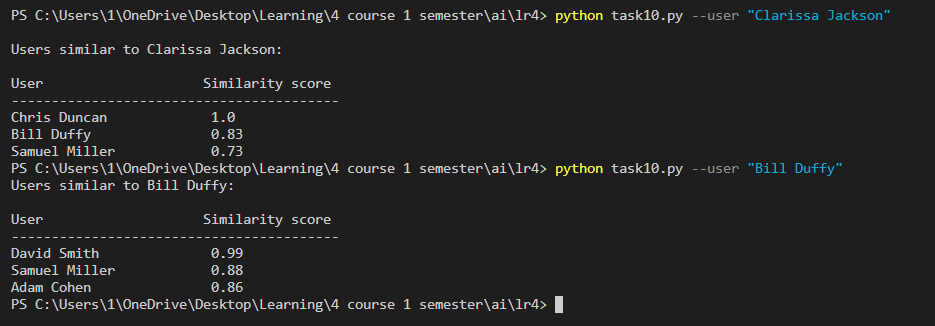
****

**Рис.31** Результат виконання програми

Оцінка подібності за Пірсоном демонструє кращі результати в порівнянні з евклідовою оцінкою подібності.

**Завдання №10:** Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації.

import argparse  
import json  
import numpy as np  
  
from task9 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))

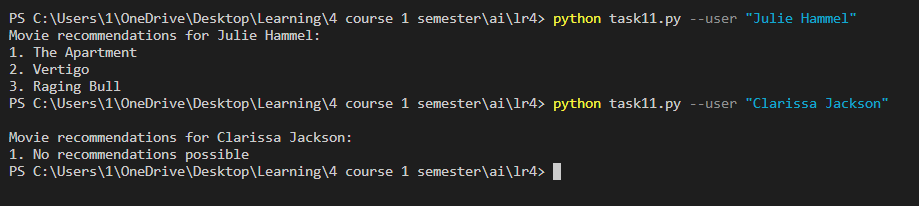


**Рис.32** Результат виконання програми

Юзер “Clarissa Jackson” має одинакові вподобання з користувачем “Chris Duncan”, а користувач “Bill Duffy” – майже однакові з “David Smith”.

**Завдання №11**: Створення рекомендаційної системи фільмів.

import argparse  
import json  
import numpy as np  
  
from task9 import pearson\_score  
from task10 import find\_similar\_users  
  
  
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)



**Рис. 33.** Результат виконання програми

Для Julie = 3 рекомендації

Для Clarisse = 0

**Висновок**: Після виконання лабораторної роботи навчився використовувати спеціалізовані бібліотеки та мову програмування Python дослідив методи ансамблів у машинному навчанні та створити рекомендаційні системи.

**Github:**