ЛАБОРАТОРНА РОБОТА № 6

СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

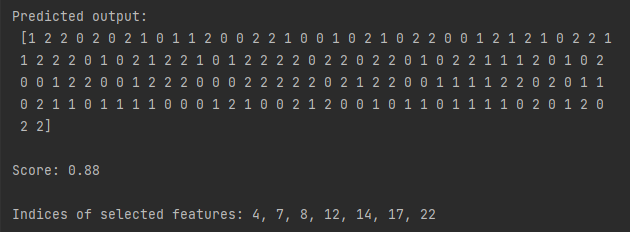
**Мета роботи:** використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати рекомендаційні системи.

Хід роботи

Завдання 2.1. Створення навчального конвеєра (конвеєра машинного навчання)

from sklearn.datasets import make\_classification  
from sklearn.feature\_selection import SelectKBest, f\_regression  
from sklearn.pipeline import Pipeline  
from sklearn.ensemble import ExtraTreesClassifier  
  
# Generate data  
X, y = make\_classification(n\_samples=150, n\_features=25, n\_classes=3, n\_informative=6, n\_redundant=0, random\_state=7)  
  
  
k\_best\_selector = SelectKBest(f\_regression, k=9)  
  
  
classifier = ExtraTreesClassifier(n\_estimators=60, max\_depth=4)  
  
  
processor\_pipeline = Pipeline([('selector', k\_best\_selector), ('erf', classifier)])  
  
  
processor\_pipeline.set\_params(selector\_\_k=7, erf\_\_n\_estimators=30)  
  
  
processor\_pipeline.fit(X, y)  
  
  
output = processor\_pipeline.predict(X)  
print("\nPredicted output:\n", output)  
  
  
print("\nScore:", processor\_pipeline.score(X, y))  
  
  
status = processor\_pipeline.named\_steps['selector'].get\_support()  
  
  
selected = [i for i, x in enumerate(status) if x]  
print("\nIndices of selected features:", ', '.join([str(x) for x in selected]))

**Рис. 1.1 Код програми**



**Рис. 1.2 Результат виконання програми**

**Висновок:** є 3 різних API для оцінки якості прогнозів моделі:

Параметр оцінки: інструменти оцінки моделі, що використовують перехресну перевірку (наприклад, model\_selection.cross\_val\_score та model\_selection.GridSearchCV) покладаються на внутрішню стратегію оцінки.

Метричні функції: модуль sklearn.metrics реалізує функції, що оцінюють помилку прогнозування для певних цілей.

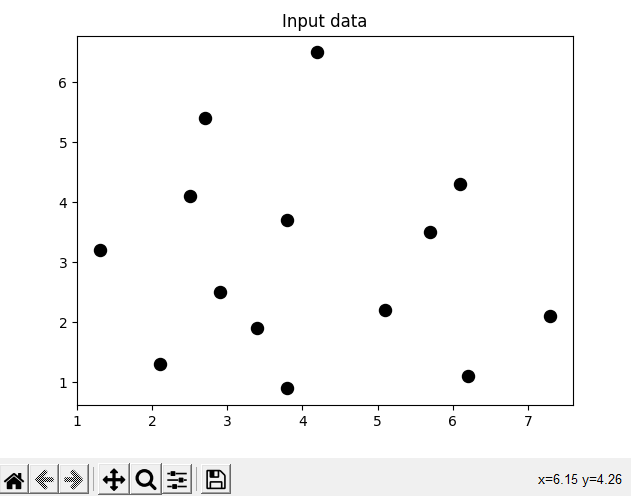
Оцінювач метод оцінки : оцінювачі є **score метод** , який забезпечує критерій оцінки за умовчанням, для завдання вони призначені для його вирішення.

Завдання 2.2. Пошук найближчих сусідів

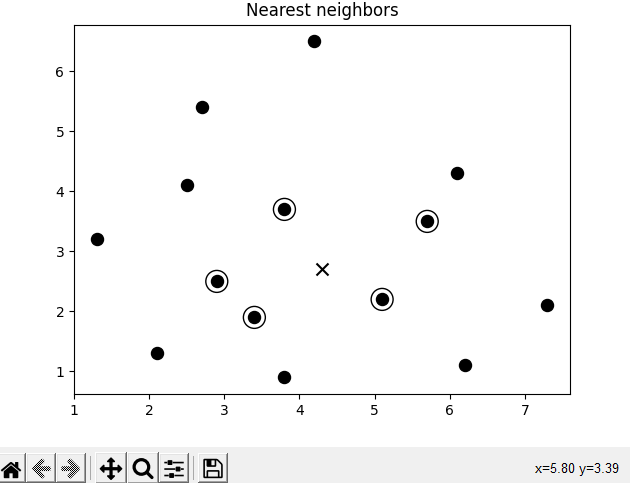
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.neighbors import NearestNeighbors  
  
X = np.array([[2.1, 1.3], [1.3, 3.2], [2.9, 2.5], [2.7, 5.4], [3.8, 0.9],  
 [7.3, 2.1], [4.2, 6.5], [3.8, 3.7], [2.5, 4.1], [3.4, 1.9],  
 [5.7, 3.5], [6.1, 4.3], [5.1, 2.2], [6.2, 1.1]])  
  
k = 5  
  
test\_datapoint = [4.3, 2.7]  
  
plt.figure()  
plt.title('Input data')  
plt.scatter(X[:,0], X[:,1], marker='o', s=75, color='black')  
  
knn\_model = NearestNeighbors(n\_neighbors=k, algorithm='ball\_tree').fit(X)  
distances, indices = knn\_model.kneighbors([test\_datapoint])  
  
print("\nK Nearest Neighbors:")  
for rank, index in enumerate(indices[0][:k], start=1):  
 print(str(rank) + " ==>", X[index])

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()

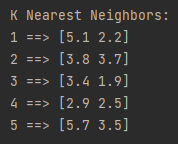
**Рис. 1.3 Код програми**



**Рис. 1.4 Результат виконання програми**



**Рис. 1.5 Результат виконання програми**



**Рис. 1.6 Результат виконання програми**

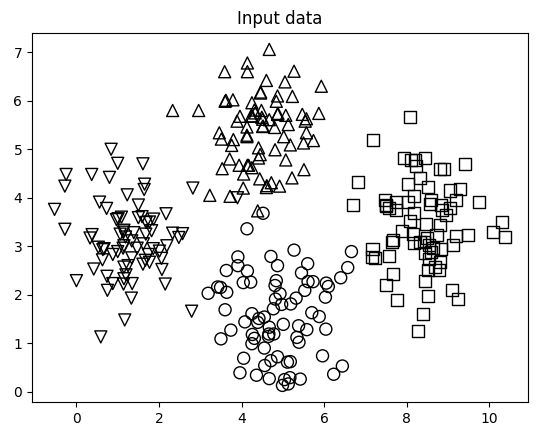
**Висновок:** здійснюється пошук найближчих сусідів заданої точки даних.  
Nearest neighbours суть якого полягає у знаходженні тих точок заданого набору, які розташовані на найближчих відстанях від зазначеної. Такий підхід часто застосовується для створення систем, що класифікують точку даних на підставі її близькості до різних класів.

Завдання 2.3. Створити класифікатор методом k найближчих сусідів

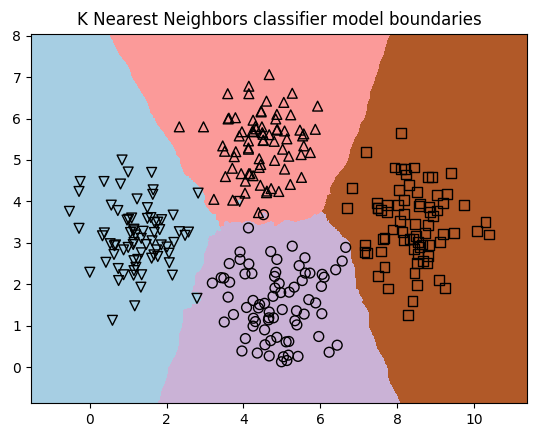
import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.cm as cm  
from sklearn import neighbors  
  
input\_file = 'data.txt'  
data = np.loadtxt(input\_file, delimiter=',')  
X, y = data[:, :-1], data[:, -1].astype(np.int)  
  
plt.figure()  
plt.title('Input data')  
marker\_shapes = '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')  
  
num\_neighbors = 12  
  
step\_size = 0.01  
  
classifier = neighbors.KNeighborsClassifier(num\_neighbors, weights='distance')  
  
classifier.fit(X, y)  
  
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))  
  
output = classifier.predict(np.c\_[x\_values.ravel(), y\_values.ravel()])  
  
output = output.reshape(x\_values.shape)  
plt.figure()  
plt.pcolormesh(x\_values, y\_values, output, cmap=cm.Paired)  
  
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')  
  
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')

\_, indices = classifier.kneighbors([test\_datapoint])  
indices = indices.astype(np.int)[0]  
  
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()

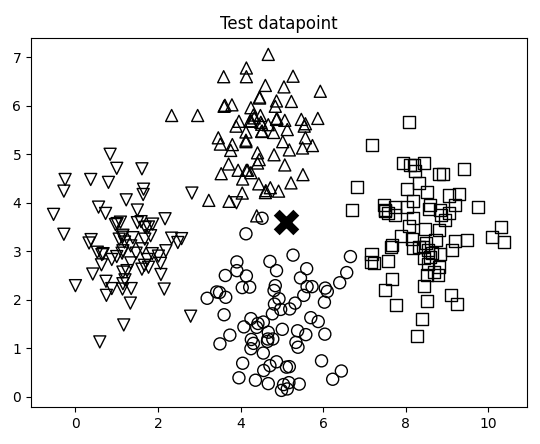
**Рис. 1.7 Код програми**



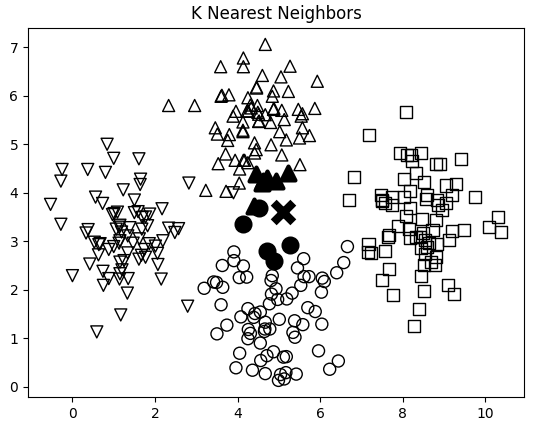
**Рис. 1.8 Результат виконання програми**



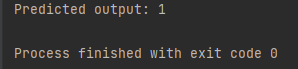
**Рис. 1.9 Результат виконання програми**



**Рис. 2.1 Результат виконання програми**



**Рис. 2.2 Результат виконання програми**



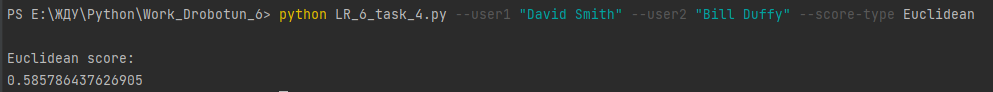
**Рис. 2.3 Результат виконання програми**

Завдання 2.4. Обчислення оцінок подібності

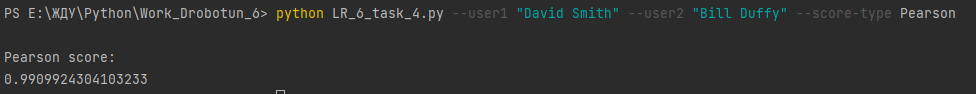
import argparse  
import json  
import numpy as np  
  
  
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(description='Compute similarity score')  
 parser.add\_argument('--user1', dest='user1', required=True,  
 help='First user')  
 parser.add\_argument('--user2', dest='user2', required=True,  
 help='Second user')  
 parser.add\_argument("--score-type", dest="score\_type", required=True,  
 choices=['Euclidean', 'Pearson'], help='Similarity metric to be used')  
 return parser  
  
  
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')  
  
 common\_movies = {}  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 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')  
  
 common\_movies = {}  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 num\_ratings = len(common\_movies)  
  
 if num\_ratings == 0:  
 return 0  
  
 user1\_sum = np.sum([dataset[user1][item] for item in common\_movies])  
 user2\_sum = np.sum([dataset[user2][item] for item in common\_movies])  
  
 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])  
  
 sum\_of\_products = np.sum([dataset[user1][item] \* dataset[user2][item] for item in common\_movies])  
  
 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))

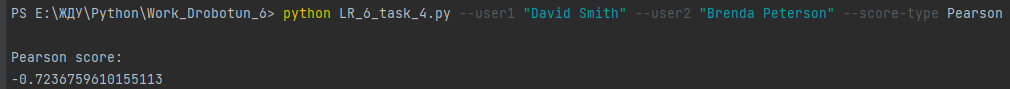
**Рис. 2.4 Код програми**



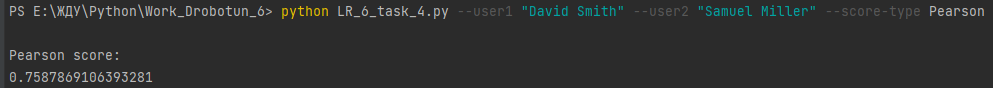
**Рис. 2.5 Результат виконання програми**



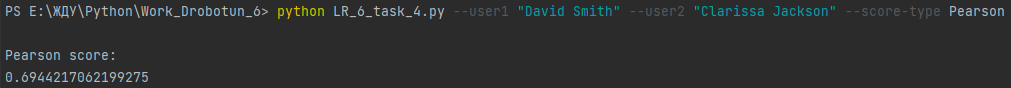
**Рис. 2.6 Результат виконання програми**



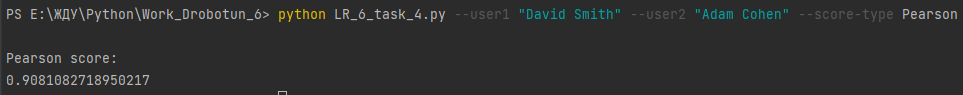
**Рис. 2.7 Результат виконання програми**



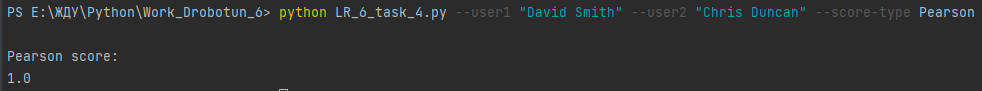
**Рис. 2.8 Результат виконання програми**



**Рис. 2.9 Результат виконання програми**



**Рис. 3.1 Результат виконання програми**



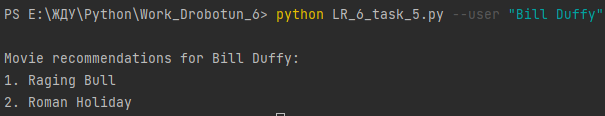
**Рис. 3.2 Результат виконання програми**

Завдання 2.5. Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації

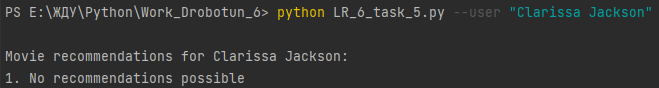
import argparse  
import json  
import numpy as np  
  
  
def pearson\_score(dataset, user1, user2):  
 if user1 not in dataset:  
 raise TypeError('Cannot find ' + user1 + ' in the dataset')  
  
 if user2 not in dataset:  
 raise TypeError('Cannot find ' + user2 + ' in the dataset')  
  
 common\_movies = {}  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 num\_ratings = len(common\_movies)  
  
 if num\_ratings == 0:  
 return 0  
  
  
 user1\_sum = np.sum([dataset[user1][item] for item in common\_movies])  
 user2\_sum = np.sum([dataset[user2][item] for item in common\_movies])  
  
  
 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])  
  
 sum\_of\_products = np.sum([dataset[user1][item] \* dataset[user2][item] for item in common\_movies])  
  
 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)  
 def find\_similar\_users(dataset, user, num\_users):  
   
 if user not in dataset:  
 raise TypeError('Cannot find ' + user + ' in the dataset')  
  
 scores = np.array([[x, pearson\_score(dataset, user,  
 x)] for x in dataset if x != user])  
  
 scores\_sorted = np.argsort(scores[:, 1])[::-1]  
  
 top\_users = scores\_sorted[:num\_users]  
  
 return scores[top\_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  
  
  
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']  
  
 movie\_scores = np.array([[score / similarity\_scores[item], item]  
 for item, score in overall\_scores.items()])  
  
 movie\_scores = movie\_scores[np.argsort(movie\_scores[:, 0])[::-1]]  
  
 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. 3.3 Код програми**



**Рис. 3.4 Результат виконання програми**



**Рис. 3.6 Результат виконання програми**

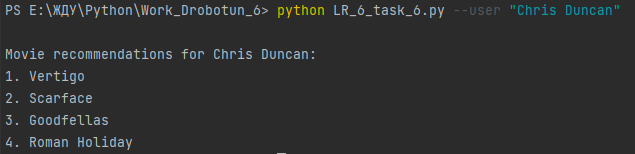
Завдання 2.6.Створення рекомендаційної системи фільмів

import argparse  
import json  
import numpy as np  
  
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')  
  
 common\_movies = {}  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 num\_ratings = len(common\_movies)  
  
 if num\_ratings == 0:  
 return 0  
  
  
 user1\_sum = np.sum([dataset[user1][item] for item in common\_movies])  
 user2\_sum = np.sum([dataset[user2][item] for item in common\_movies])  
  
  
 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])  
  
 sum\_of\_products = np.sum([dataset[user1][item] \* dataset[user2][item] for item in common\_movies])  
  
 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

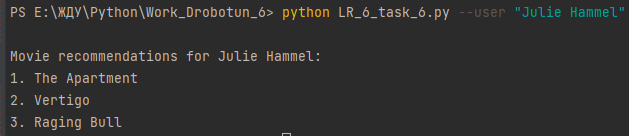
if Sxx \* Syy == 0:  
 return 0  
  
 return Sxy / np.sqrt(Sxx \* Syy)  
def find\_similar\_users(dataset, user, num\_users):  
 if user not in dataset:  
 raise TypeError('Cannot find ' + user + ' in the dataset')  
  
 scores = np.array([[x, pearson\_score(dataset, user,  
 x)] for x in dataset if x != user])  
  
 scores\_sorted = np.argsort(scores[:, 1])[::-1]  
  
 top\_users = scores\_sorted[:num\_users]  
  
 return scores[top\_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  
  
  
  
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']  
  
 movie\_scores = np.array([[score / similarity\_scores[item], item]  
 for item, score in overall\_scores.items()])  
  
 movie\_scores = movie\_scores[np.argsort(movie\_scores[:, 0])[::-1]]  
  
 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. 3.7 Код програми**



**Рис. 3.8 Результат виконання програми**



**Рис. 3.9 Результат виконання програми**

**Висновок:** я використовав спеціалізовані бібліотеки та мову програмування Python дослідивши методи регресії даних у машинному навчанні.

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.