ЛАБОРАТОРНА РОБОТА № 6

СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

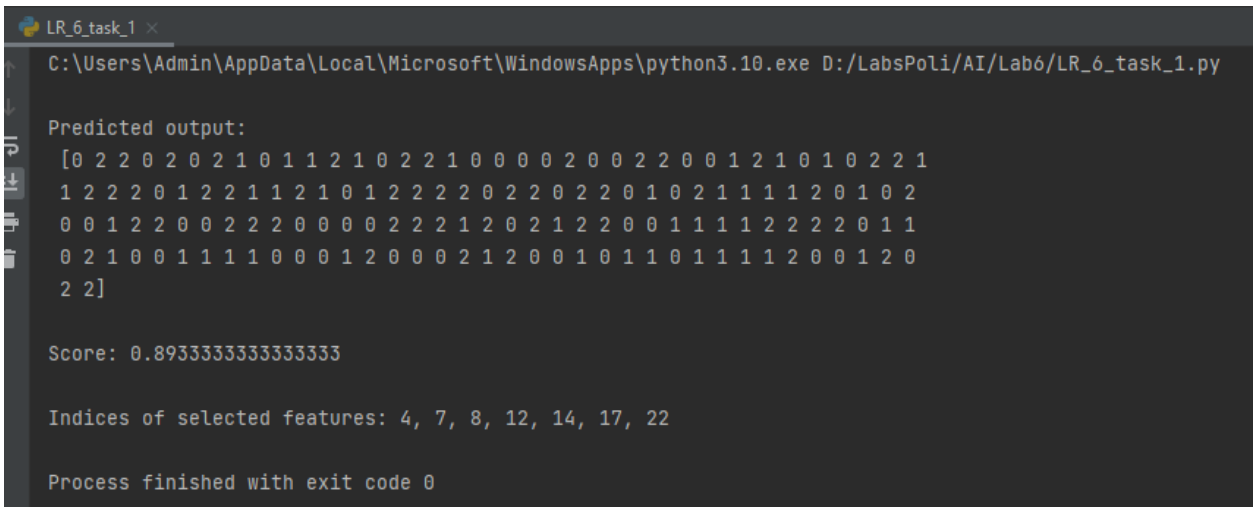
Мета роботи: використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися створювати рекомендаційні системи.

GitHub: https://github.com/ingaliptn/AI

Хід роботи

Завдання 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

# Генерування даних
X, y = 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]))
```

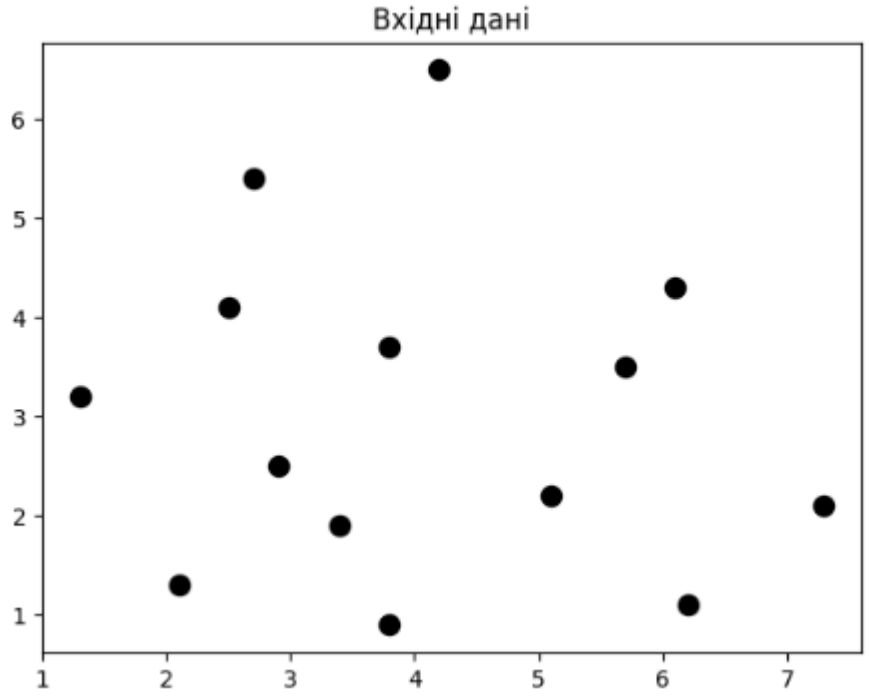


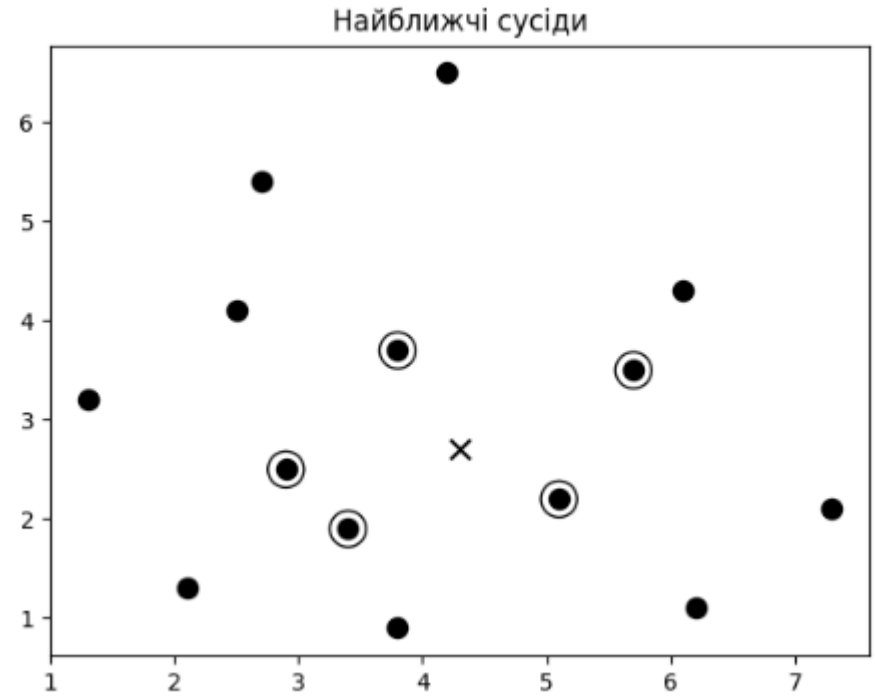
В першому рядку виведені спрогнозовані результати для всіх вхідних значень. Значення Score показує оцінку точності обрахування. В останньому рядку виведені індекси обраних ознак.

Завдання 2. 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, котрі хочемо витягти
k = 5

# Тестова точка даних
test_datapoint = [4.3, 2.7]

# Відображення вхідних даних на графіку
plt.figure()
plt.title('Вхідні дані')
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('Найближчі сусіди')
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()
```





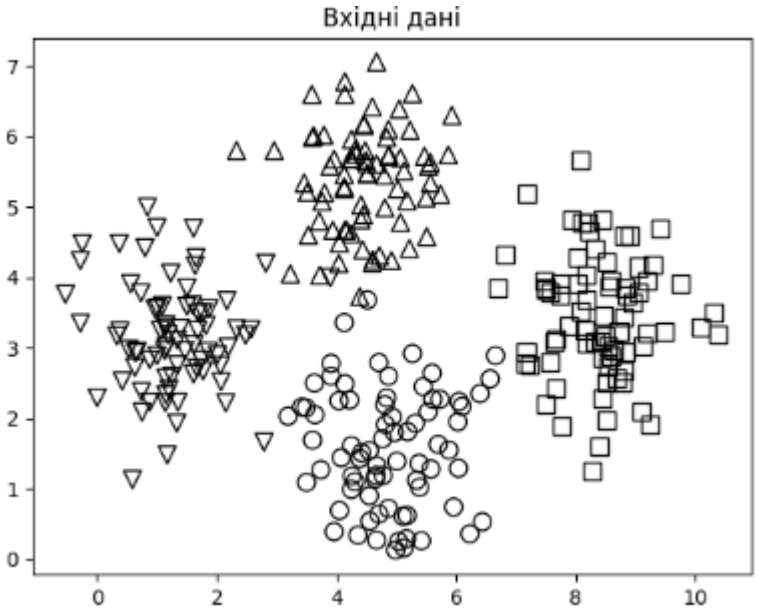
```
LR_6_task_2
C:\Users\Admin\AppData\Local\Microsoft\WindowsApps\python3.10.exe D:/LabsPoli/AI/Lab6/LR_6_task_2.py

K Nearest Neighbors:
1 ==> [5.1 2.2]
2 ==> [3.8 3.7]
3 ==> [3.4 1.9]
4 ==> [2.9 2.5]
5 ==> [5.7 3.5]

Process finished with exit code 0
```

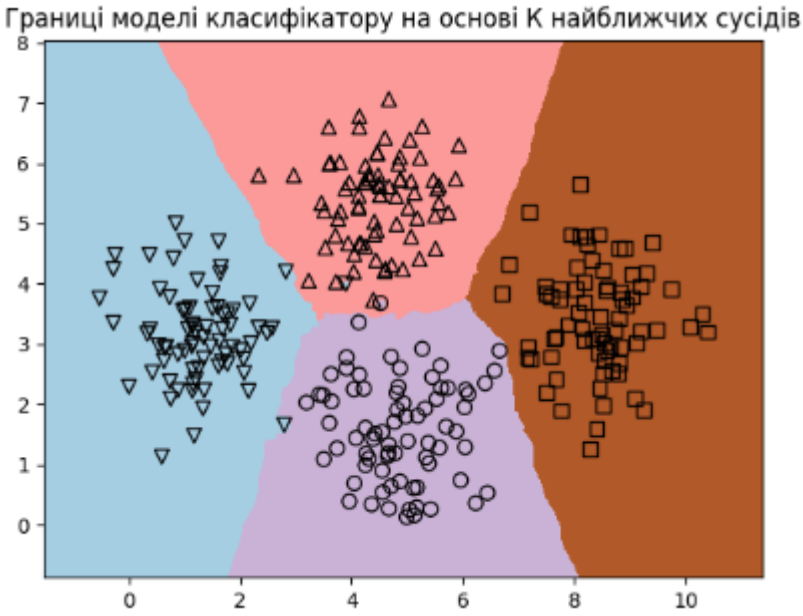
На першому графіку зображено розташування вхідних даних. На другому графіку зображені найближчі сусіди до тестової точки. У вікні терміналу вказані координати найближчих сусідів до тестової точки.

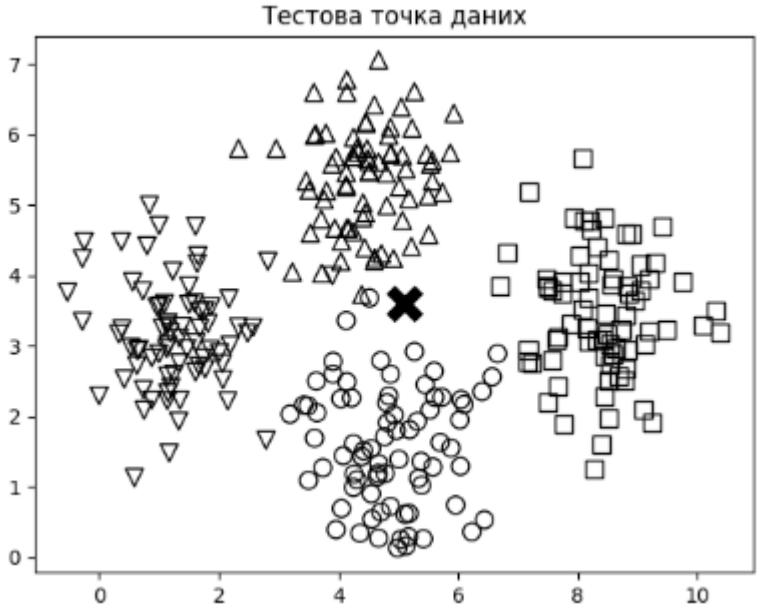
Завдання 3. Створити класифікатор методом k найближчих сусідів

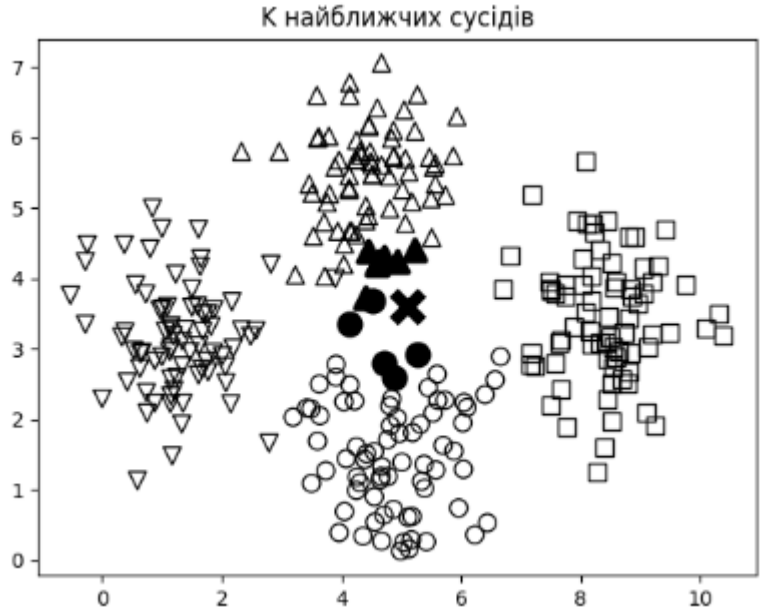
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import neighbors

# Завантаження вхідних даних
input_file = 'data.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1].astype(int)

# Відображення вхідних даних на графіку
plt.figure()
plt.title('Вхідні дані')
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

# Крок сітки
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')

# Задання граничних значень для осей
plt.xlim(x_values.min(), x_values.max())
plt.ylim(y_values.min(), y_values.max())
plt.title('Границі моделі класифікатору на основі К найближчих сусідів')

# Тестування вхідної точки даних
test_datapoint = [5.1, 3.6]
plt.figure()
plt.title('Тестова точка даних')
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 К найближчих сусідів
_, indices = classifier.kneighbors([test_datapoint])
indices = indices.astype(int)[0]

# Відображення К найближчих сусідів на графіку
plt.figure()
plt.title('К найближчих сусідів')
for i in indices:
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[y[i]],
                linewidth=3, s=100, facecolors='black')

# Відображення тестової точки
plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',
            linewidth=6, s=200, facecolors='black')

# Відображення вхідних даних
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
                s=75, edgecolors='black', facecolors='none')

# Виведення прогнозованого результату
print("Predicted output:", classifier.predict([test_datapoint])[0])
plt.show()
```


Вхідні дані


Границі моделі класифікатору на основі К найближчих сусідів


Тестова точка даних




```
C:\Users\Admin\AppData\Local\Microsoft\WindowsApps\python3.10.exe D:/LabsPoli/AI/Lab6/LR_6_task_3.py
Predicted output: 1

Process finished with exit code 0
```

Точка належить до класу з індексом 1, трикутник.

Завдання 4. Обчислення оцінок подібності

```
import argparse
import json
import numpy as np

# Створення парсеру для обробки вхідних аргументів
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

    # Обчислення квадрату різниці між рейтинговими оцінками
    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)))

# Вичислення коефіцієнту корреляції Пірсона для user1 та user2
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 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

    # У відсутності відхилення оцінки дорівнює 0
    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))
```

Евклідова оцінка подібності та оцінка подібності Пірсона користувачів David Smith та Bill Duffy

```
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Euclidean
Euclidean score:
0.585786437626905
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Pearson
Pearson score:
0.9909924304103233
```

Евклідова оцінка подібності та оцінка подібності Пірсона користувачів David Smith та Brenda Peterson

```
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Brenda Peterson" --score-type Euclidean
Euclidean score:
0.1424339656566283
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Brenda Peterson" --score-type Pearson
Pearson score:
-0.7236759610155113
```

Евклідова оцінка подібності та оцінка подібності Пірсона користувачів David Smith та Samuel Miller

```
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Euclidean
Euclidean score:
0.30383243470068705
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Pearson

Pearson score:
0.7587869106393281
```

Евклідова оцінка подібності та оцінка подібності Пірсона користувачів David Smith та Julie Hammel

```
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Julie Hammel" --score-type Euclidean

Euclidean score:
0.2857142857142857
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Julie Hammel" --score-type Pearson

Pearson score:
0
```

Евклідова оцінка подібності та оцінка подібності Пірсона користувачів David Smith та Clarissa Jackson

```
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Euclidean

Euclidean score:
0.28989794855663564
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Pearson

Pearson score:
0.6944217062199275
```

Евклідова оцінка подібності та оцінка подібності Пірсона користувачів David Smith та Adam Cohen

```
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Adam Cohen" --score-type Euclidean

Euclidean score:
0.38742588672279304
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Adam Cohen" --score-type Pearson

Pearson score:
0.9081082718950217
```

Евклідова оцінка подібності та оцінка подібності Пірсона користувачів David Smith та Clarissa Jackson

```
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Euclidean

Euclidean score:
0.28989794855663564
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_4.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Pearson

Pearson score:
0.6944217062199275
PS D:\LabsPoli\AI\Lab6>
```

Завдання 5. Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації

```python
import argparse
import json
import numpy as np

from LR_6_task_4 import pearson_score


# Створення парсеру для обробки вхідних аргументів
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


# Знаходження аналогічних користувачів
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 + ":")
    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))
```

```
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_5.py --user "Bill Duffy"

Users similar to Bill Duffy:
User                    Similarity score
-----------------------------------------
David Smith              0.99
Samuel Miller            0.88
Adam Cohen               0.86
PS D:\LabsPoli\AI\Lab6>
```

```
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_5.py --user "Clarissa Jackson"

Users similar to Clarissa Jackson:
User                    Similarity score
----------------------------------------
Chris Duncan             1.0
Bill Duffy               0.83
Samuel Miller            0.73
PS D:\LabsPoli\AI\Lab6>
```

Завдання 6. Створення рекомендаційної системи фільмів

```
import argparse
import json
import numpy as np

from LR_6_task_4 import pearson_score


# Створення парсеру для обробки вхідних аргументів
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 = {}

    # Обчислення оцінки подібності між вказаним користувачем та всіма іншими
користувачами у наборі даних.
    for user in [x for x in dataset if x != input_user]:
        similarity_score = pearson_score(dataset, input_user, user)

        # Якщо оцінка подібності менша за 0, переходимо до наступного
користувача.
        if similarity_score <= 0:
            continue

        # Вилучення списку фільмів, що вже отримали рейтингову оцінку від
поточного користувача, але ще не оцінених
        # зазначеним користувачем
        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)
```

```
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_6.py --user "Chris Duncan"

Movie recommendations for Chris Duncan:
1. Vertigo
2. Scarface
3. Goodfellas
4. Roman Holiday
PS D:\LabsPoli\AI\Lab6>
```

```
PS D:\LabsPoli\AI\Lab6> python3 .\LR_6_task_6.py --user "Julie Hammel"

Movie recommendations for Julie Hammel:
1. The Apartment
2. Vertigo
3. Raging Bull
PS D:\LabsPoli\AI\Lab6>
```

Висновок

Я, використовуючи спеціалізовані бібліотеки та мову програмування Python навчився створювати рекомендаційні системи.