Лабораторна робота №6

Посилання на гіт-хаб: https://github.com/ViMIMercurysMight/python-ai

Тема: Створення рекомендаційних систем

Завдання 1: Створення навчального конвеєра (конвеєра машинного навчання)

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

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

X, y = _samples_generator.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)
process_pipeline = Pipeline([('selector', k_best_selector), ('erf', classifier)])


process_pipeline.set_params(selector__k=7, erf__n_estimators=30)
process_pipeline.fit(X, y)
output = process_pipeline.predict(X)
print("\nPredicted output:\n", output)

print("\nScore:", process_pipeline.score(X, y))
status = process_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]))
```

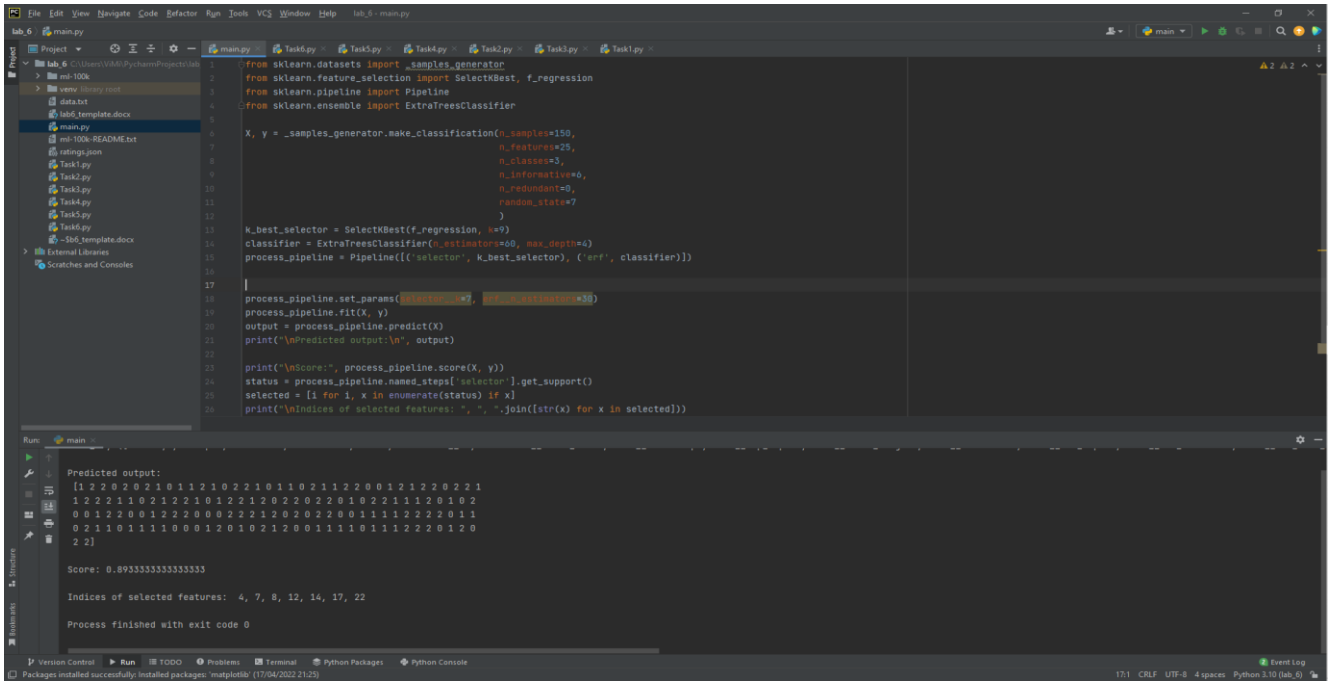

Рисунок 1. Результати виконання

Як можна побачити з виводу результату виконання (рис.1) В першому виводі данних міститься масив передбачення для усього набору даних, наступним рядком виведено оцінку навчання яка отримана шляхом трансформаціх данних, і останнім рядком вивденно вибрані шляхом аналізу вхідного набору дані для «рекомендації» їх.

Завдання 2: Пошук найближчих сусідів

Лістинг програми:

```
#Task 2
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()
```

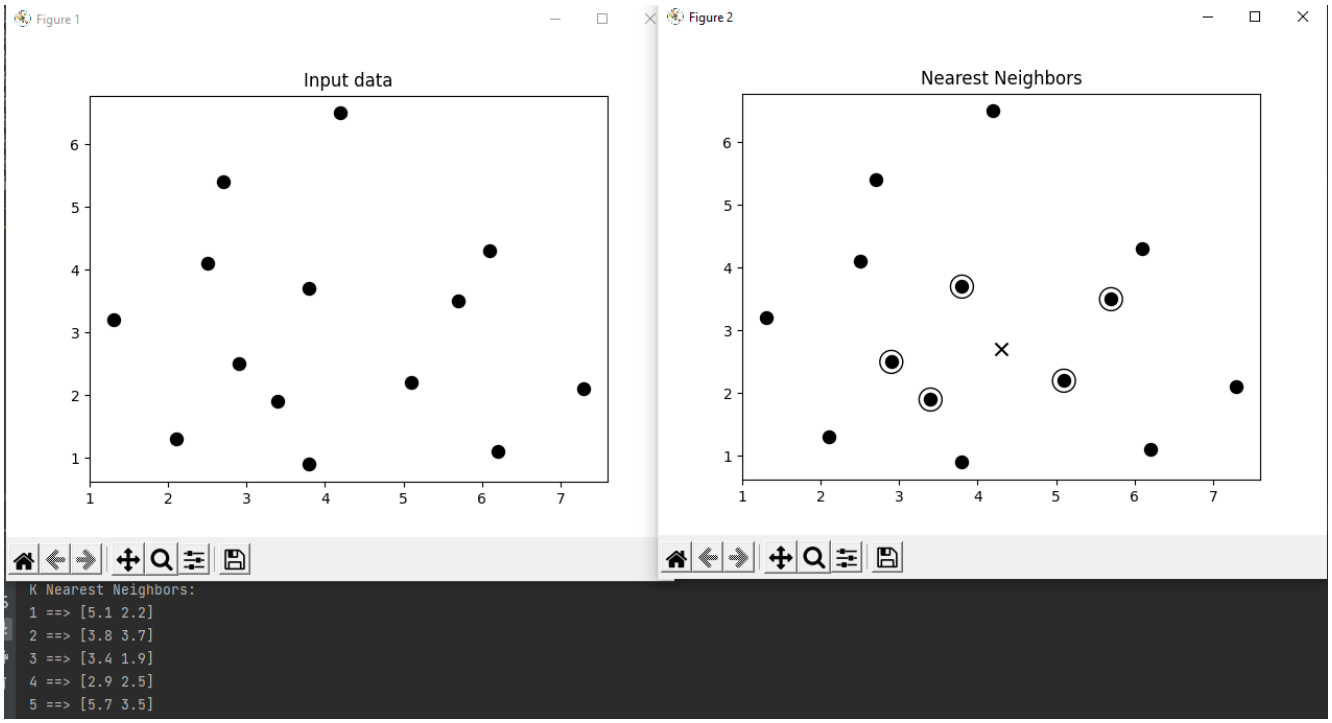

Рисунок 2. Результат виконання програми

Як можна бачити на першому графыку зображені усі сусіди, на другому визначені найближчі у середньому сусіди. В вікно терміналу було виведено дані про (координати тестової точки) цих п’яти найближчих сусідів

Завдання 3: Створити класифікатор методом k найближчих сусідів

Лістинг програми:

```
#Task 3
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import neighbors, datasets

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

plt.figure()
plt.title("InputData")
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('Edges model classifier on base K nearest neighbors')

test_datapoint = [5.1, 3.6]
plt.figure()
plt.title('Test data point')
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', linewidths=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]],
                linewidths=3,
                s=100,
                facecolors='black')
plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',
            linewidths=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()
```

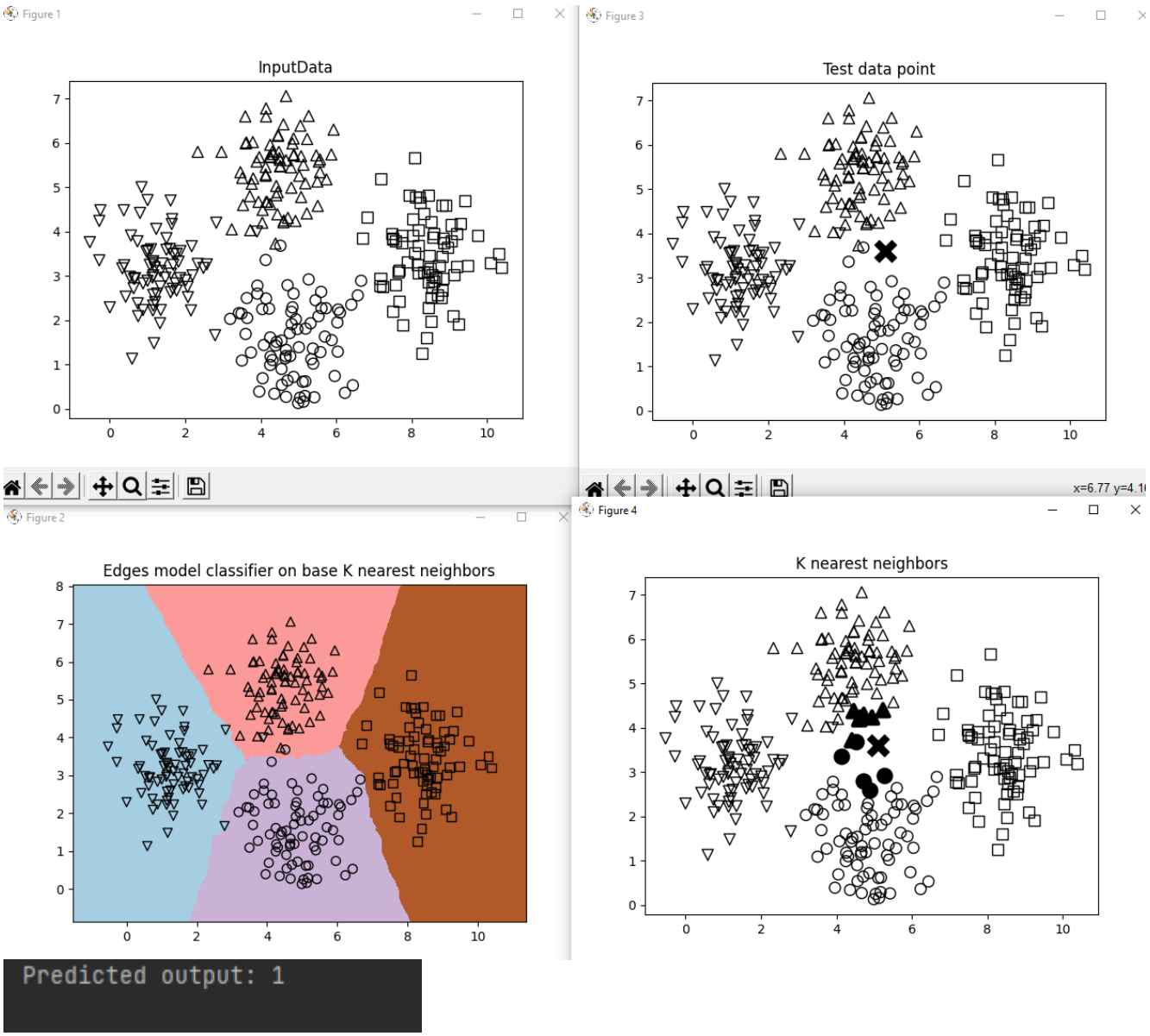

Рисунок 4. Результато виконання

Результато виконання програми стало чотири діагрми. На першій було нанесено усі точки віднесені до різних типів. На другому визначено центр, на третьому ці точки було віднесено до чотирох кластерів з позначенням їх меж, на останньому було визначено усі найближчі до центра точки й зафарбовано їх чорним.
Сама тестова точка відноситься до класу 1

Завдання 4: Обчислення оцінки подібності

Лістинг програми:

```
#Task 4

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 us-
er')
    parser.add_argument('--score-type',
                        dest='score_type',
                        required=True,
                        choices=['Euclidean', 'Pearson'],
                        help='Similarity metric to be user')
    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)))

#Коеф. кореляції Пірсона
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)

    # За відсутності фільмів, оцінених обома користувачами,
    # оцінка приймається рівною 0
    if num_ratings == 0:
        return 0

    # Обчислення суми рейтингових оцінок усіх фільмів,
    # оцінених обома користувачами
    user1_sum = np.sum([dataset[user1][item] for item in common_movies])
    user2_sum = np.sum([dataset[user2][item] for item in common_movies])

    # Обчислення Суми квадратів рейтингових оцінок всіх
    # фільмів, оцінених обома користувачами
```

```
    user1_squared_sum = np.sum([np.square(dataset[user1][item])
                                for item in common_movies])
    user2_squared_sum = np.sum([np.square(dataset[user2][item])
                                for item in common_movies])
    # Обчислення суми творів рейтингових оцінок всіх
    # фільмів, оцінених обома користувачами
    sum_of_products = np.sum([dataset[user1][item] *
                              dataset[user2][item] for item in common_movies])

    # Обчислення коефіцієнта кореляції Пірсона
    Sxy = sum_of_products - (user1_sum * user2_sum / num_ratings)
    Sxx = user1_squared_sum - np.square(user1_sum) / num_ratings
    Syy = user2_squared_sum - np.square(user2_sum) / num_ratings

    if Sxx * Syy == 0:
        return 0

    return Sxy / np.sqrt(Sxx * Syy)

if __name__=='__main__':
    args = build_arg_parser().parse_args()
    user1 = args.user1
    user2 = args.user2
    score_type = args.score_type

ratings_file = 'ratings.json'

with open(ratings_file, 'r') as f:
    data = json.loads(f.read())

if score_type == 'Euclidean':
    print("\nEuclidean score:")
    print(euclidean_score(data, user1, user2))
else:
    print("\nPearson score:")
    print(pearson_score(data, user1, user2))
```

Проведемо тестові запуски программи для користувачів David Smith та Bill Duffy (рис.3)

```
Euclidean score:
0.585786437626905
```

```
Pearson score:
0.9909924304103233
```

Рисунок 3. Результат запиту програми

Проведемо запити программи для пар: David Smith та Brenda Peterson, David Smith та Samuel Miller, David Smith та Julie Hammel, David Smith та Clarissa Jackson , David Smith та Adam Cohen David Smith та Chris Duncan (рис 4)

```
Euclidean score:
0.1424339656566283
0.30383243470068705
0.2857142857142857
0.28989794855663564
0.38742588672279304
0.38742588672279304

Pearson score:
-0.7236759610155113
0.7587869106393281
0
0.6944217062199275
0.9081082718950217
1.0
```

Рисунок 4. Результат запиту для багатьох користувачів

Завдання 5: Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації

Для полегшення тестування програми, код оцінювання з попереднього завдання було імплементовано напрюму в поточне

Лістинг програми:

```
#Task 5
import argparse
import json
import numpy as np


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)


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')
    # 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]

user = "Bill Duffy"
#user = "Clarissa Jackson"
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))
```

```
Users similar to Bill Duffy:

User            Similarity score
-------------------------------------
David Smith          0.99
Samuel Miller        0.88
Adam Cohen       0.86
```

```
Users similar to Clarissa Jackson:

User              Similarity score
-----------------------------------------
Chris Duncan           1.0
Bill Duffy        0.83
Samuel Miller          0.73
```

Рисунок 5. Результат виконання роботи

Шляхом використання методу колаборативної фільтрації було знайдено користувачів зі схожими вподобаннями.

Завдання 6: Створення рекомендаційної системи фільмів

Подібно до попереднього завдання код попереднів прикладів було імплементовано в код програми напряму без імпорутвання.

Лістинг програми

```
#Task 6
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 build_arg_parser():
    parser = argparse.ArgumentParser(description=
                                     'Find users who are similar to the input user
')
    parser.add_argument('--user', dest='user', required=True, help='Input user')
    return parser


def find_similar_users(dataset, user, num_users):
    if user not in dataset:
        raise TypeError('Cannot find ' + user + ' in the dataset')
    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]

# Отримати рекомендації щодо фільмів
# для вказаного користувача
def get_recommendations(dataset, input_user):
    if input_user not in dataset:
        raise TypeError('Cannot find ' + input_user + ' in the dataset')

    overall_scores = {}
    similarity_scores = {}

    for user in [x for x in dataset if x != input_user]:
```

```
        similarity_score = pearson_score(dataset, input_user, user)
        if similarity_score <= 0:
            continue
        filtered_list = [x for x in dataset[user] if x not in \
                    dataset[input_user] or dataset[input_user][x] == 0]

        for item in filtered_list:
            overall_scores.update({item: dataset[user][item] * similarity_score})
            similarity_scores.update({item: similarity_score})

    if len(overall_scores) == 0:
        return ["No recommendations possible"]
    # Генерація рейтингів фільмів за допомогою їх нормалізації
    movie_scores = np.array([[score/similarity_scores[item],
                              item] for item, score in overall_scores.items()])
    # Сортування за спаданням
    movie_scores = movie_scores[np.argsort(movie_scores[:, 0])[::-1]]
    # Вилучення рекомендацій фільмів
    movie_recommendations = [movie for _, movie in movie_scores]
    return movie_recommendations

user = "Chris Duncan"
#user = "Julie Hammel"

ratings_file = 'ratings.json'
with open(ratings_file, 'r') as f:
    data = json.loads(f.read())

print("\n Movie recommendations for " + user + ":")
movies = get_recommendations(data, user)
for i, movie in enumerate(movies):
    print(str(i+1) + '. ' + movie)
```

```
 Movie recommendations for Julie Hammel:
1. The Apartment
2. Vertigo
3. Raging Bull
```

```
 Movie recommendations for Chris Duncan:
1. Vertigo
2. Scarface
3. Goodfellas
4. Roman Holiday
```

Рисунок 5. Результат виконання програми

Процес пошуку рекомендацій в фільмах в деякому сенсі подібний до процесу пошуку корисувачів зі схожими вподобаннями, але в якості основи було обрано оцінки фільмів. Для розширення можливостей системи можливо розширити фільтр для використання таких критеріїв як жанр, режисер тощо, що в цілому не є складною задачею.

Висновок: Розглянули способи побудови рекомендаційних систем, та оцінювання.