ЛАБОРАТОРНА РОБОТА № 6

СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

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

Хід роботи

GitHub репозиторій: https://github.com/evhenSuhostavets/SAI_ipzk_20_1

Завдання 1

Код 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)
proces-
sor_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:', 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])
)
```

```
Predicted output: [0 2 2 0 2 0 2 1 0 1 1 2 2 0 2 2 1 0 0 1 0 2 1 1 2 2 0 0 1 2 1 2 1 0 2 2 1
 1 2 2 2 0 1 2 2 1 2 2 1 0 1 2 2 1 2 0 2 2 0 2 2 0 2 0 2 2 1 1 1 2 0 1 0 2
 0 0 1 2 2 0 0 2 2 2 0 0 0 0 2 2 2 1 2 0 2 2 2 2 0 0 1 1 1 1 2 2 2 2 0 1 1
 0 2 1 0 0 1 1 1 1 0 0 0 1 2 1 0 0 2 1 2 0 0 1 0 1 1 0 1 1 1 1 0 2 0 1 2 0
 2 2]

Score: 0.88

Indices of selected features: 4, 7, 8, 12, 14, 17, 22
```

Рис. 1.1. Результат виконання

В результаті виконання даного завдання були згенеровані дані та створено конвеєр для вибору характеристик й навчання класифікатора на цих даних.

Перший рядок містить передачені мітки для згенерованих даних X.

Score – результат середньої точності останнього класифікатора пайплайну на даних, які є параметрами відповідної функції.

Останній рядок містить індекси обраних ознак селектором ознак.

Завдання 2

Код програми:

```python
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('Вхідні дані')
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('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 = 'x', s = 75, color = 'k')

plt.show()
```

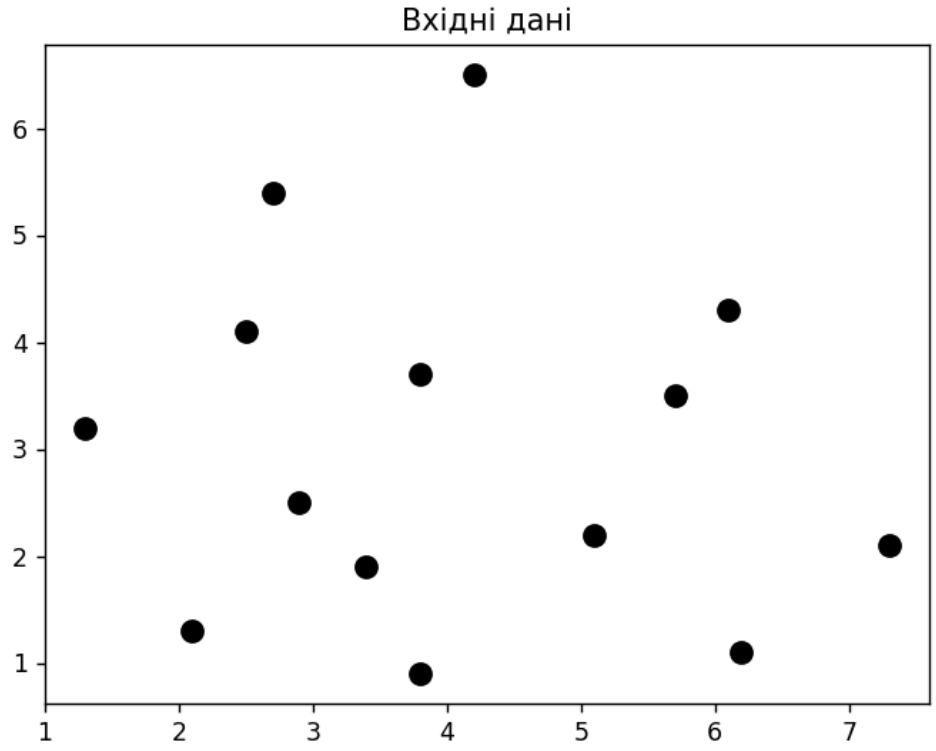

Рис. 1.2. Графік розподілу вхідних даних

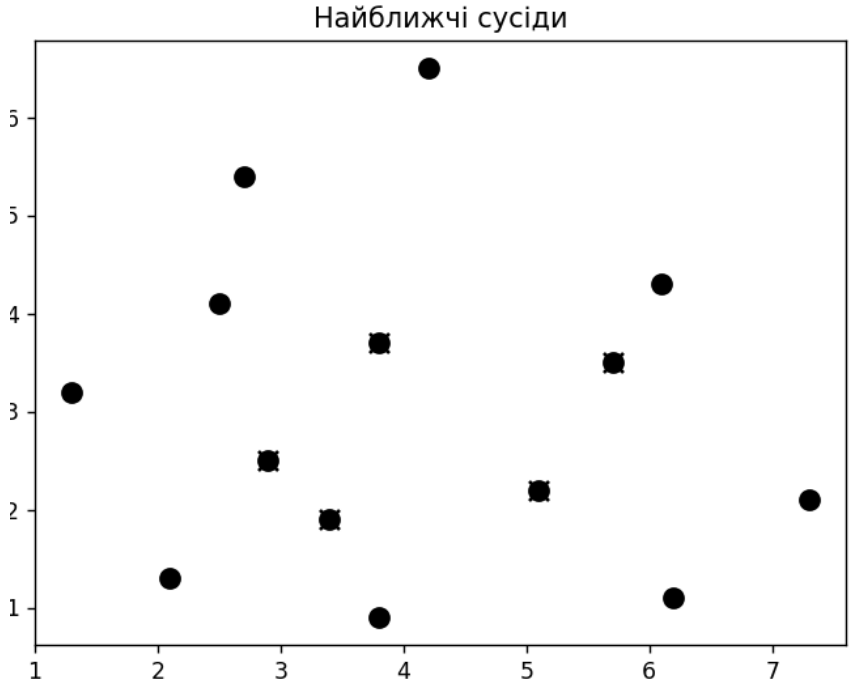

Рис. 1.3. Графік k найближчих точок до тестової точки

```
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]
```

Рис. 1.4. Результат виконання

В результаті виконання даного завдання було досліджено процес пошуку найближчих сусідів.

На першому графіку графічно зображено вхідні дані.

На другому графіку крапками зоборжено вхідні дані та хрестиками позначено точки, які є найближчими сусідами до тестової точки.

У вікні терміналу було виведено наближчі точки у порядку зростання відстані від тестової точки.

Завдання 3

Код програми:

```
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('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 = 'bl
ack', 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 = 'bl
ack', facecolors = 'none')

plt.xlim(x_values.min(), x_values.max())
plt.ylim(y_values.min(), y_values.max())
plt.title('Межі моделі класифікатора на основі K найближчих сусідів')

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 = 50, edgecolors = 'bl
ack', 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('К найближчих сусіди')
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()
```

На рисунку 1.5 зображено розподіл вхідних даних на координатній площині відповідно до їх класів.

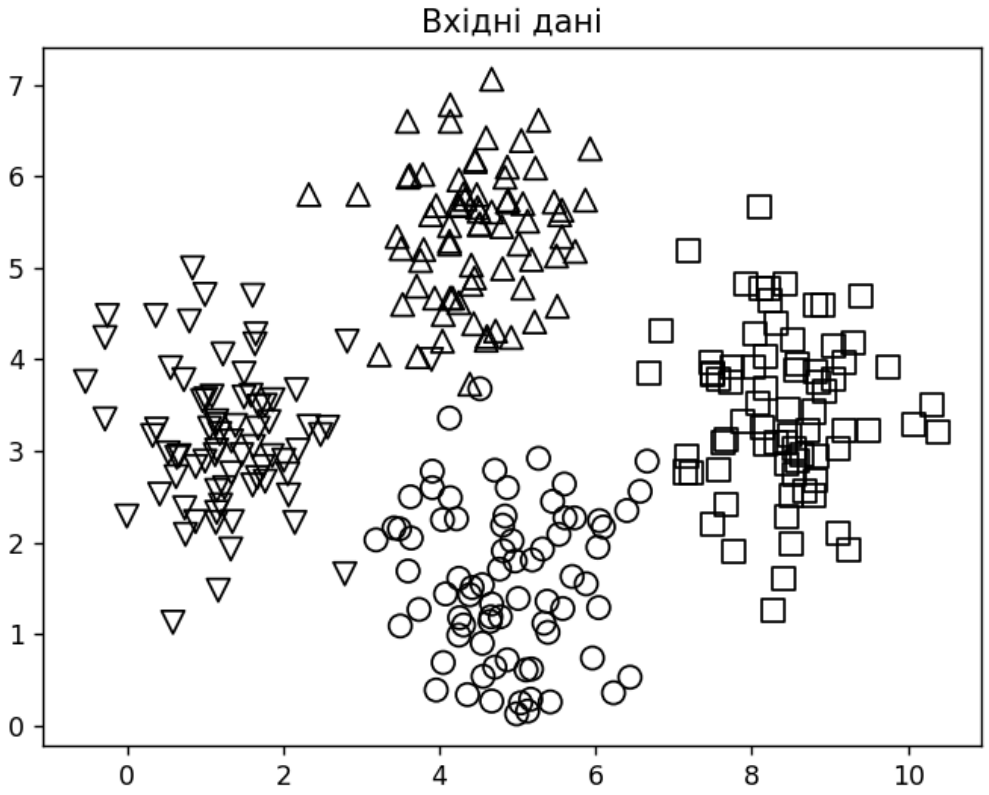

Рис. 1.5. Графічне зображення вхідних даних

Нарисунку 1.6 зображено межі кожного класу визначені за допомогою класифікатора.

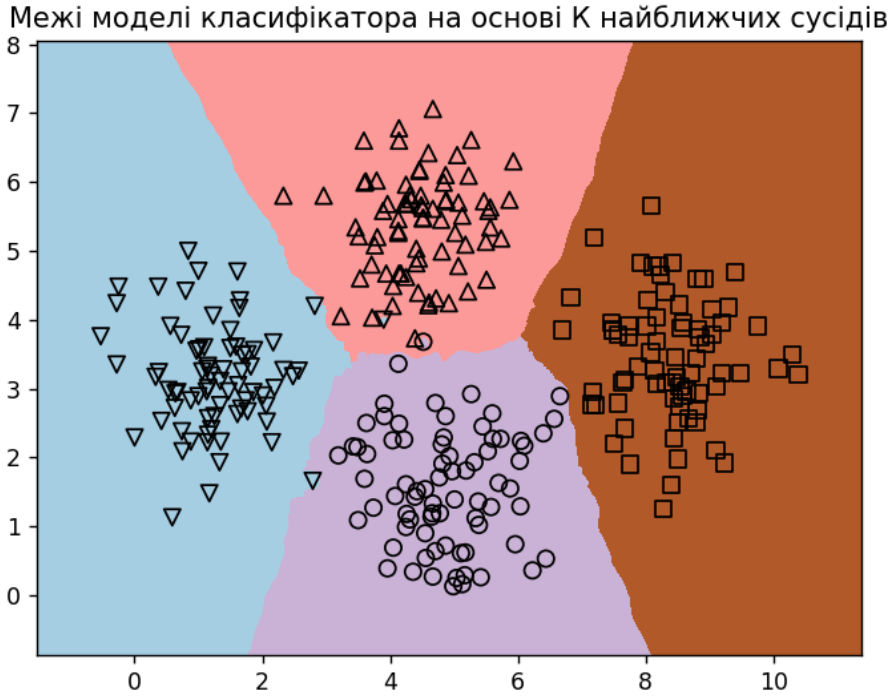

Рис. 1.6. Графічне зображення меж моделей класифікатора

На рисунку 1.7 зображено розподіл вхідних даних відповідно до їх класів та ткстової точки на координатній площині.

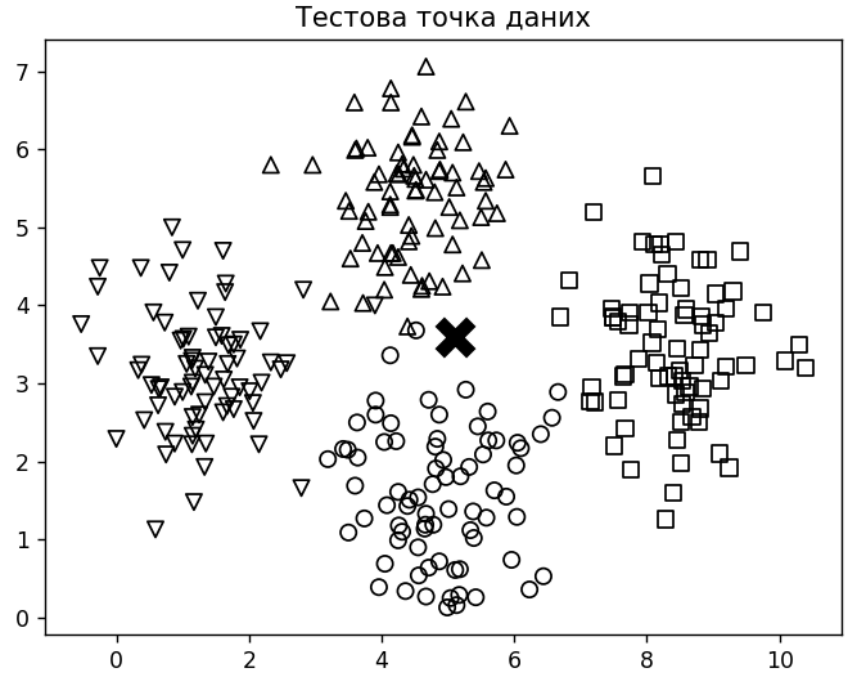

Рис. 1.7. Графічне зображення тестової точки та вхідних даних

На рисунку 1.8 виділено k найбличних сусідів тестової точки поміж усіх вхідних даних.

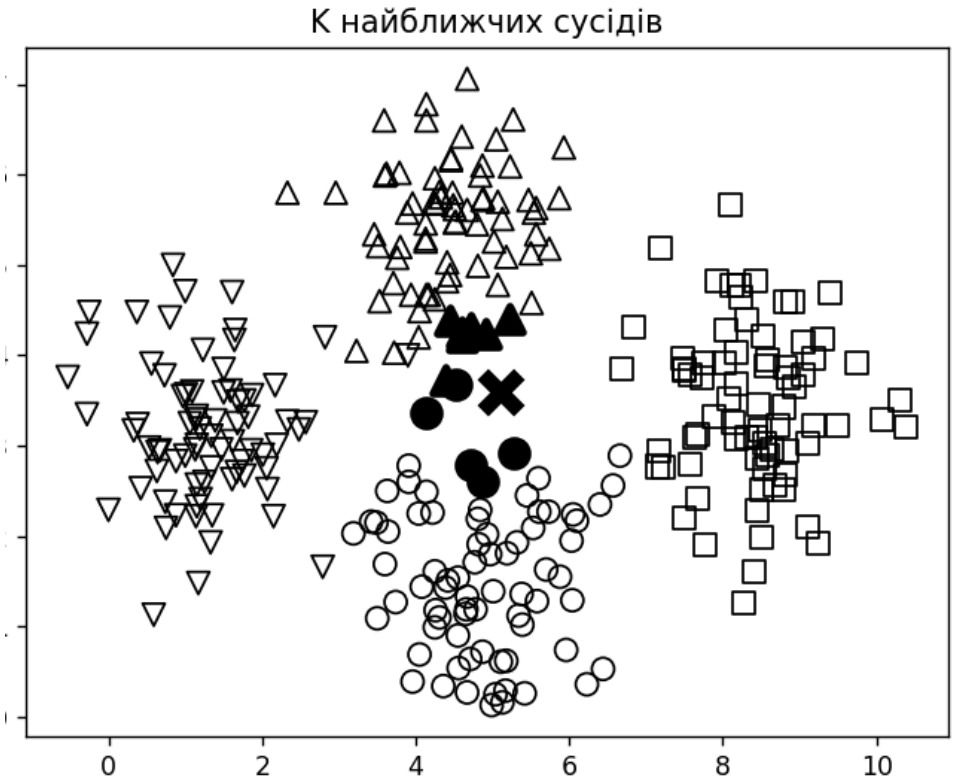

Рис. 1.8. Графічне зображення K-найближчих сусідів тестової точки поміж вхідних даних

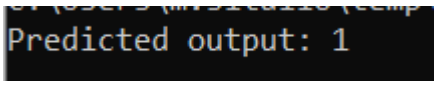

Рис. 1.9. Результат виконання

Відповідно до вікна терміналу, можемо побачити, що тестова точна належить до класу ‘1’.

В даному завданні ми навчилися використовувати класифікатор методу K найближчих сусідів.

Завдання 4

Код програми:

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

```
# обробка 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

# метод обрахування евклідової оцінки
def euclidean_score(dataset, user1, user2):
    # перевірка чи користувачі існують у вхідному наборі даних
    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')

    # визначення набору фільмів, які були оцінені обома користувачами
    common_movies = {}
    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1

    # обробка випадку, коли подібних фільмів немає
    if len(common_movies) == 0:
        return 0

    # пошук евклідової відстані між парами оцінок користувачів
    squared_diff = []
    for item in common_movies:
        squared_diff.append(np.square(dataset[user1][item] -
 dataset[user2][item]))

    # обрахування евклідової відстані
    return 1 / (1 + np.sqrt(np.sum(squared_diff)))

def pearson_score(dataset, user1, user2):
    # перевірка чи користувачі існують у вхідному наборі даних
    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')

    # визначення набору фільмів, які були оцінені обома користувачами
    common_movies = {}
    for item in dataset[user1]:
```

```
            if item in dataset[user2]:
                common_movies[item] = 1

        # обробка випадку, коли подібних фільмів немає
        num_rating = len(common_movies)
        if num_rating == 0:
            return 0

        # вираховування суми оцінок для всіх спільних фільмів
        user1_sum = np.sum([dataset[user1][item] for item in common_movies])
        user2_sum = np.sum([dataset[user2][item] for item in common_movies])

        # вираховування квадратичної суми оцінок для всіх спільних фільмів
        us-
er1_squared_sum = np.sum([np.square(dataset[user1][item]) for item in common_m
ovies])
        us-
er2_squared_sum = np.sum([np.square(dataset[user2][item]) for item in common_m
ovies])

        # вираховування суми добутків оцінок для всіх спільних фільмів
        sum_of_products = np.sum([dataset[user1][item] * dataset[user2][item] for
item in common_movies])

        # вираховування параметрів оцінки подібності за Пірсоном
        Sxy = sum_of_products - (user1_sum * user2_sum / num_rating)
        Sxx = user1_squared_sum - np.square(user1_sum) / num_rating
        Syy = user2_squared_sum - np.square(user2_sum) / num_rating

        # перевірка випадку нульового добутку
        if Sxx * Syy == 0:
            return 0

        # вираховування оцінки подібності за Пірсоном
        return Sxy / np.sqrt(Sxx * Syy)

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

        # зчитування файлу з рейтинговими оцінками
        rating_file = 'ratings.json'
        with open(rating_file, 'r') as f:
            data = json.loads(f.read())

        # обрахування подібності відповідно до обраної оцінки
        if score_type == 'Euclidean':
```

```
        print('\nEuclidean score:')
        print(euclidean_score(data, user1, user2))
    else:
        print('\nPearson score:')
        print(pearson_score(data, user1, user2))
```

```
C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py --user1 "David Smith" --user2 "Bil
Duffy" --score-type Euclidean

Euclidean score:
0.585786437626905
```

Рис. 1.10. Результат виконання програми з евклідовою оцінкою

```
C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py --user1 "David Smith" --user2 "Bill
Duffy" --score-type Pearson

Pearson score:
0.9909924304103233
```

Рис. 1.11. Результат виконання програми з оцінкою подібності Пірсона

```
C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py
--user1 "David Smith" --user2 "Brenda Peterson" --score-type Euclidean

Euclidean score:
0.1424339656566283

C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py
--user1 "David Smith" --user2 "Brenda Peterson" --score-type Pearson

Pearson score:
-0.7236759610155113

C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py
--user1 "David Smith" --user2 "Samuel Miller" --score-type Euclidean

Euclidean score:
0.30383243470068705

C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py
--user1 "David Smith" --user2 "Samuel Miller" --score-type Pearson

Pearson score:
0.7587869106393281

C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py
--user1 "David Smith" --user2 "Julie Hammel" --score-type Euclidean

Euclidean score:
0.2857142857142857

C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py
--user1 "David Smith" --user2 "Julie Hammel" --score-type Pearson

Pearson score:
0

C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py
--user1 "David Smith" --user2 "Clarissa Jackson" --score-type Euclidean

Euclidean score:
0.28989794855663564

C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py
--user1 "David Smith" --user2 "Clarissa Jackson" --score-type Pearson
```

Рис. 1.12. Результат виконання програми для інших пар користувачів

```
C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py
--user1 "David Smith" --user2 "Adam Cohen" --score-type Euclidean

Euclidean score:
0.38742588672279304

C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py
--user1 "David Smith" --user2 "Adam Cohen" --score-type Pearson

Pearson score:
0.9081082718950217

C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py
--user1 "David Smith" --user2 "Chris Duncan" --score-type Euclidean

Euclidean score:
0.38742588672279304

C:\Users\m.sitailo\temp\University\Штучний інтелект\SAI\Lab06>python LR_6_task_4.py
--user1 "David Smith" --user2 "Chris Duncan" --score-type Pearson

Pearson score:
1.0
```

Рис. 1.13. Результат виконання програми для інших пар користувачів

В результаті виконання даного завдання було досліджено обчислення оцінок подібності за допомогою евклідової відстані та методу подібності за Пірсоном.

Завдання 5

Код програми:

```
import argparse
import json
import numpy as np

# обробка вхідних аргументів
def build_arg_parser():
    par-
ser = 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

# вирахування оцінки подібності за Пірсоном
def pearson_score(dataset, user1, user2):
    # перевірка чи користувачі існують у вхідному наборі даних
    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')
```

```
    # визначення набору фільмів, які були оцінені обома користувачами
    common_movies = {}
    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1

    # обробка випадку, коли подібних фільмів немає
    num_rating = len(common_movies)
    if num_rating == 0:
        return 0

    # вираховування суми оцінок для всіх спільних фільмів
    user1_sum = np.sum([dataset[user1][item] for item in common_movies])
    user2_sum = np.sum([dataset[user2][item] for item in common_movies])

    # вираховування квадратичної суми оцінок для всіх спільних фільмів
    us-
er1_squared_sum = np.sum([np.square(dataset[user1][item]) for item in common_m
ovies])
    us-
er2_squared_sum = np.sum([np.square(dataset[user2][item]) for item in common_m
ovies])

    # вираховування суми добутків оцінок для всіх спільних фільмів
    sum_of_products = np.sum([dataset[user1][item] * dataset[user2][item] for
item in common_movies])

    # вираховування параметрів оцінки подібності за Пірсоном
    Sxy = sum_of_products - (user1_sum * user2_sum / num_rating)
    Sxx = user1_squared_sum - np.square(user1_sum) / num_rating
    Syy = user2_squared_sum - np.square(user2_sum) / num_rating

    # перевірка випадку нульового добутку
    if Sxx * Syy == 0:
        return 0

    # вираховування оцінки подібності за Пірсоном
    return Sxy / np.sqrt(Sxx * Syy)

# метод пошуку подібних користувачів
def find_similar_users(dataset, user, num_users):
    # перевірка чи користувач існує у наборі даних
    if user not in dataset:
        raise TypeError('Cannot find ' + user + ' in the dataset')

    # вирахування подібності оцінок з всіма іншими користувачами
    scores = np.array([[x, pearson_score(dataset, user, x)] for x in dataset i
f x != user])
    # сортування оцінок подібності за спаданням
```

```
    scores_sorted = np.argsort(scores[:, 1])[::-1]
    # вибір num_users найбільш подібних користувачів
    top_users = scores_sorted[:num_users]

    return scores[top_users]

if __name__ == '__main__':
    # зчитування параметрів
    args = build_arg_parser().parse_args()
    user = args.user

    # зчитування файлу з рейтинговими оцінками
    rating_file = 'ratings.json'
    with open(rating_file, 'r') as f:
        data = json.loads(f.read())

    print('\nUsers similar to ', user + ':\n')
    # пошук подібних користувачів
    similar_users = find_similar_users(data, user, 3)
    print('User\t\t\tSimilarity score')
    print('-' * 41)
    # виведення імен та оцінок подібності найбільш схожих користувачів
    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
```

Рис. 1.14. Результат виконання програми для Bill Duffy

```
Users similar to  Clarissa Jackson:

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

Рис. 1.15. Результат виконання програми для Clarissa Jackson

В результаті виконання даного завдання було досліджено метод пошуку користувачів зі схожими уподобаннями методом колаборативної фільтрації.

Завдання 6

Код програми:

```
import argparse
import json
import numpy as np

# обробка вхідних аргументів
def build_arg_parser():
    par-
ser = argparse.ArgumentParser(description = 'Find the movie recommendations fo
r the given user')
    parser.add_argument('--
user', dest = 'user', required = True, help = 'Input user')

    return parser

# вирахування оцінки подібності за Пірсоном
def pearson_score(dataset, user1, user2):
    # перевірка чи користувачі існують у вхідному наборі даних
    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')

    # визначення набору фільмів, які були оцінені обома користувачами
    common_movies = {}
    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1

    # обробка випадку, коли подібних фільмів немає
    num_rating = len(common_movies)
    if num_rating == 0:
        return 0

    # вираховування суми оцінок для всіх спільних фільмів
    user1_sum = np.sum([dataset[user1][item] for item in common_movies])
    user2_sum = np.sum([dataset[user2][item] for item in common_movies])

    # вираховування квадратичної суми оцінок для всіх спільних фільмів
    us-
er1_squared_sum = np.sum([np.square(dataset[user1][item]) for item in common_m
ovies])
```

```
    us-
er2_squared_sum = np.sum([np.square(dataset[user2][item]) for item in common_m
ovies])

    # вираховування суми добутків оцінок для всіх спільних фільмів
    sum_of_products = np.sum([dataset[user1][item] * dataset[user2][item] for
item in common_movies])

    # вираховування параметрів оцінки подібності за Пірсоном
    Sxy = sum_of_products - (user1_sum * user2_sum / num_rating)
    Sxx = user1_squared_sum - np.square(user1_sum) / num_rating
    Syy = user2_squared_sum - np.square(user2_sum) / num_rating

    # перевірка випадку нульового добутку
    if Sxx * Syy == 0:
        return 0

    # вираховування оцінки подібності за Пірсоном
    return Sxy / np.sqrt(Sxx * Syy)

# метод пошуку подібних користувачів
def find_similar_users(dataset, user, num_users):
    # перевірка чи користувач існує у наборі даних
    if user not in dataset:
        raise TypeError('Cannot find ' + user + ' in the dataset')

    # вирахування подібності оцінок з всіма іншими користувачами
    scores = np.array([[x, pearson_score(dataset, user, x)] for x in dataset i
f x != user])
    # сортування оцінок подібності за спаданням
    scores_sorted = np.argsort(scores[:, 1])[::-1]
    # вибір num_users найбільш подібних користувачів
    top_users = scores_sorted[:num_users]

    return scores[top_users]

def get_recommendations(dataset, input_user):
    # перевірка чи користувач існує у наборі даних
    if input_user not in dataset:
        raise TypeError('Cannot find ' + user + ' in the dataset')

    overall_scores = {}
    similarity_scores = {}

    # аналіз користувачів, окрім користувача для якого ми шукаємо рекомендації
    for user in [x for x in dataset if x != input_user]:
        # вирахування оцінки подібності за Пірсоном
        similarity_score = pearson_score(dataset, input_user, user)
        # якщо оцінка подібності <= 0, користувачі не є подібними та їх рекоме
ндації не будуть корисними
```

```
            if similarity_score <= 0:
                continue

            # відфільтровуємо список фільмів переглянутих поточним користувачем, я
кі ще не дивився користувач для якого ми шукаємо рекомендації
            fil-
tered_list = [x for x in dataset[user] if x not in dataset[input_user] or data
set[input_user][x] == 0]
            for item in filtered_list:
                # вираховуємо зважену оцінку фільму та зберігаємо цю оцінку та оці
нку подібності для подалюшого використання
                over-
all_scores.update({item: dataset[user][item] * similarity_score})
                similarity_scores.update({item: similarity_score})

            # якщо не було знайдено фільмів, які дивився тільки поточний користува
ч, виводимо повідомлення про те, що ми не можемо надати рекомендації
            if len(overall_scores) == 0:
                return ['No recommendations possible']

            # конвертуэмо зважену оцінку в оцінку, яка була поставлена поточним ко
ристувачем
            mov-
ie_scores = np.array([[score / similarity_scores[item], item] for item, score
in overall_scores.items()])
            # сортуємо фільми за спаданням їх оцінки
            movie_scores = movie_scores[np.argsort(movie_scores[:, 0])[::-1]]
            # формуємо список рекомендцій
            movie_recommendations = [movie for _, movie in movie_scores]

            return movie_recommendations


if __name__ == '__main__':
    # зчитування параметрів
    args = build_arg_parser().parse_args()
    user = args.user

    # зчитування файлу з рейтинговими оцінками
    rating_file = 'ratings.json'
    with open(rating_file, 'r') as f:
        data = json.loads(f.read())

    print('\nMovie recommendations for ', user + ':')
    # отримання та виведення списку рекомендації
    movies = get_recommendations(data, user)
    for i, movie in enumerate(movies):
        print(str(i + 1) + '.', movie)
```

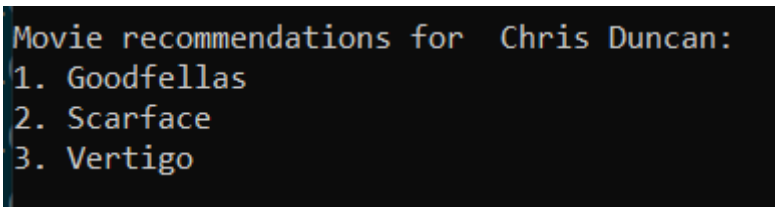

Рис. 1.16. Результат виконання програми для Chris Duncan

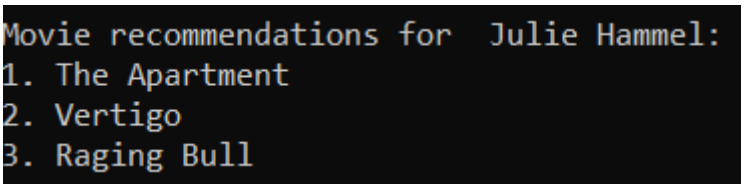

Рис. 1.17. Результат виконання програми для Julie Hammel

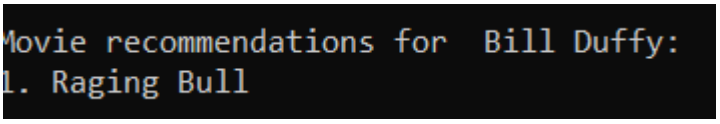

Рис. 1.18. Результат виконання програми для Bill Duffy

В результаті виконання даного завдання ми навчилися створювати систему рекомендацій фільмів на основі подібності між користувачами.

Висновки: на даній лабораторній роботі ми навчилися створювати рекомендаційні системи використовуючи спеціалізовані бібліотеки та мову програмування Python.