

ЛАБОРАТОРНА РОБОТА № 6

СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

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

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Теоретичні відомості подані на лекціях. Також доцільно вивчити матеріал поданий в літературі:

Джоши Пратик. Искусственный интеллект с примерами на Python. : Пер. с англ. - СПб. : ООО "Диалектика", 2019. - 448 с. - Парал. тит. англ. ISBN 978-5-907114-41-8 (рус.)

Можна використовувати Google Colab або Jupiter Notebook.

Системи рекомендацій

Персоналізація користувацького досвіду була пріоритетом і стала новою мантрою в галузях, орієнтованих на споживача. Можливо, ви помітили, як компанії електронної комерції розміщують персоналізовану рекламу для вас, пропонуючи, що купувати, які новини читати, яке відео дивитися, де/що їсти та з ким вам може бути цікаво спілкуватися (друзі/професіонали) у соціальних мережах. медіа сайти. Системи рекомендацій — це основна система фільтрації інформації, розроблена для прогнозування переваг користувачів і допомоги рекомендувати правильні елементи для створення специфічного для користувача досвіду персоналізації.

Існує два типи рекомендаційних систем: 1) фільтрація на основі вмісту та 2) спільна фільтрація (рис 1).

Фільтрація на основі вмісту (Content-Based Filtering)

Цей тип системи зосереджується на атрибуті подібності елементів, щоб дати вам рекомендації. Найкраще це можна зрозуміти на прикладі: якщо користувач придбав товари певної категорії, інші подібні товари з тієї ж категорії рекомендовані користувачеві (див. рис 1).

Алгоритм рекомендацій щодо схожості на основі елементів можна представити у вигляді:

$$\hat{x}_{k,m} = \frac{\sum_{i_b} sim_i(i_m, i_b)(x_{k,b})}{\sum_{i_b} |sim_i(i_m, i_b)|}$$

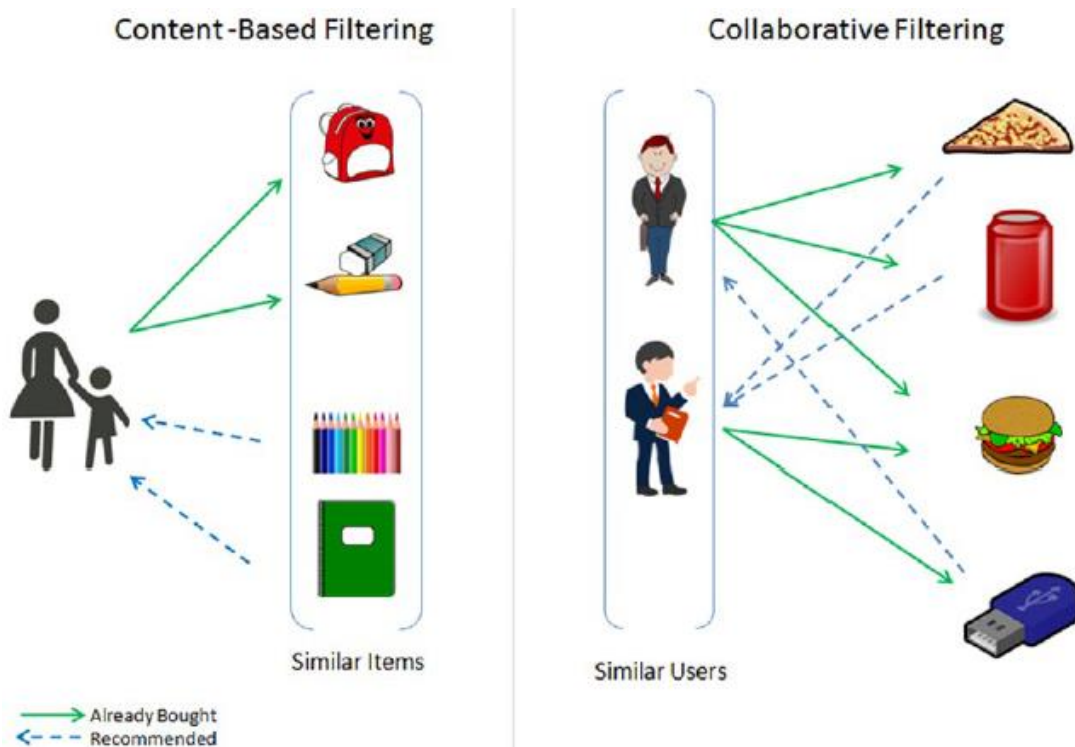


Рис.1

Спільна фільтрація (Collaborative Filtering (CF))

CF зосереджується на атрибуті подібності користувачів, тобто він знаходить людей зі схожими смаками на основі міри схожості з великої групи користувачів. На практиці існує два типи реалізації CF: на основі пам'яті та на основі моделі.

Тип на основі пам'яті в основному базується на алгоритмі подібності; Алгоритм розглядає елементи, які подобаються подібним людям, щоб створити ранжований список рекомендацій. Потім ви можете відсортувати рейтинговий список, щоб рекомендувати користувачеві перші n елементів.

Алгоритм рекомендацій щодо схожості на основі користувачів можна представити як:

$$pr_{x,k} = m_x + \frac{\sum_{u_y \in N_x} (r_{y,k} - m_y) \text{sim}(u_x, u_y)}{\sum_{u_y \in N_x} |\text{sim}(u_x, u_y)|}$$

Давайте розглянемо та дослідимо приклади рекомендаційних систем.

2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ

Завдання 2.1. Створення навчального конвеєра (конвеєра машинного навчання)

Зазвичай, системи машинного навчання будуються на модульній основі. Конкретна кінцева мета досягається з допомогою формування відповідних комбінацій окремих модулів. У бібліотеці `scikit-learn` містяться функції, що дозволяють об'єднувати різні моди в єдині конвеєр

Конвеєр може формуватися з модулів, що виконують різні функції, такі як відбір ознак, попередня обробка даних, побудова випадкових лісів, кластеризація і т.п.

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

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
from sklearn.datasets import samples_generator
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
from sklearn.ensemble import ExtraTreesClassifier
```

Згенеруємо марковані вибірккові дані для процесів навчання та тестування. Пакет `scikit-learn` включає вбудовану функцію, яка справляється із цим завданням. У наведеному нижче рядку коду створюються 150 точок даних, кожна з яких є 25-мірним вектором. Числові значення у кожному векторі ознак генеруватимуться з використанням генератора випадкових вибірок. Кожна точка даних включає шість інформативних ознак і не містить жодної надлишкової.

Використовуємо наступний код.

Генерування 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)
```

Першим блоком цього конвеєра є селектор ознак. Цей блок відбирає `k` "найкращих" ознак. Встановимо для `k` значення 9.

Вибір `k` найважливіших ознак

```
k_best_selector = SelectKBest(f_regression, k=9)
```

Наступний блок конвеєра - класифікатор на основі гранично випадкового лісу з 60 деревами та максимальною глибиною, що дорівнює чотирьом. Використовуємо наступний код.

```
# Ініціалізація класифікатора на основі гранично випадкового лісу
classifier = ExtraTreesClassifier(n_estimators=60, max_depth=4)
```

Створимо конвеєр за допомогою об'єднання описаних блоків. Ми можемо присвоїти ім'я кожному блоку, щоб їх легше було відслідковувати

```
# Створення конвеєра
processor_pipeline = Pipeline([('selector', k_best_selector),
                              ('erf', classifier)])
```

Можна змінювати параметри окремих блоків. Давайте надамо значення 7 параметру k у першому блоці і значення 30 кількість дерев (n_estimators) у другому блоці. Для визначення областей видимості змінних ми використовуємо імена, раніше надані блокам.

```
# Встановлення параметрів
processor_pipeline.set_params(selector__k=7, erf__n_estimators=30)
```

Навчимо конвеєр, використовуючи згенеровані перед цим вибірккові дані.

```
# Навчання конвеєра
processor_pipeline.fit(X, y)
```

Спрогнозуємо результати для всіх вхідних значень та виведемо їх.

```
# Прогнозування результатів для вхідних даних
output = processor_pipeline.predict(X)
print("\nPredicted output:\n", output)
```

Обчислимо оцінку, використовуючи марковані тренувальні дані.

```
# Виведення оцінки
print("\nScore:", processor_pipeline.score(X, y))
```

Витягнемо ознаки, відібрані блоком селектора. Ми вказали, що хочемо вибрати 7 таких ознак із загальної кількості 25. Використовуємо наступний код.

```
# 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]))
```

Після виконання цього коду у вікні терміналу відобразиться інформація
Зробіть скрін вікна терміналу та вставте його у звіт.

Напишіть висновок у звіт. У висновках поясніть:

Що міститься у першому списку?

Що означає значення *Score*?

Що міститься в останньому рядку ?

Збережіть код робочої програми під назвою *LR_6_task_1.py*

Завдання 2.2. Пошук найближчих сусідів

Для формування ефективних рекомендацій у рекомендаційних системах використовується поняття найближчих сусідів (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.

Здійсніть пошук найближчих сусідів заданої точки даних.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

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

Визначте вибірку двовимірних точок даних.

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]])
```

Визначте кількість найближчих сусідів, котрі хочемо витягти.

```
k = 5
```

Визначимо тестову точку даних, на яку будемо вилучати k найближчих сусідів.

```
# Тестова точка даних
test_datapoint = [4.3, 2.7]
```

Відобразимо на графіку вхідні дані, використовуючи як маркери чорні кружки.

```
# Відображення вхідних даних на графіку
plt.figure()
plt.title('Входные данные')
plt.scatter(X[:,0], X[:,1], marker='o', s=75, color='black')
```

Створимо та навчимо модель на основі методу k найближчих сусідів, використовуючи вхідні дані. Застосуємо цю модель для отримання найближчих сусідів нашої тестової точки даних.

```
# 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 витягнуті з моделі точки даних, що є найближчими сусідами.

```
# Виведемо 'k' найближчих сусідів
print("\nK Nearest Neighbors:")
for rank, index in enumerate(indices[0][:k], start=1):
    print(str(rank) + " ==>", X[index])
```

Візуалізуємо найближчих сусідів.

Візуалізація найближчих сусідів разом із тестовою точкою даних

```
plt.figure()
plt.title('Ближайшие соседи')
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='k')
plt.scatter(X[indices[0][:k]][:, 0], X[indices[0][:k]][:, 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()
```

У процесі виконання цього коду на екрані відобразяться два рафіка.

Обидва графіка занесіть у звіт.

У вікні терміналу з'явиться інформація. **Інформацію занесіть у звіт**

Зробіть висновки в яких укажіть:

Що відображено на першому графіку.

Що відображено на другому графіку.

Що відображено у вікні терміналу.

Збережіть код робочої програми під назвою LR_6_task_2.py

Завдання 2.3. Створити класифікатор методом k найближчих сусідів

Класифікатор на основі k найближчих сусідів – це модель класифікації, в якій задана точка класифікується з використанням алгоритму найближчих сусідів. Для визначення категорії вхідної точки, даний алгоритм знаходить у навчальному наборі k точок, що є найближчими по відношенню до заданої. Після цього призначений точці даних клас визначається "голосуванням". Ми переглядаємо класи k елементів отриманим списком і вибираємо з них той клас, якому відповідає найбільша кількість "голосів". Значення k залежить від конкретного завдання.

Використовуючи для аналізу дані, які містяться у файлі `data.txt`. Створіть класифікатор методом k найближчих сусідів.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

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

Завантажте вхідні дані з файлу `data.txt`. Кожен рядок цього файлу містить значення, розділені комою, причому дані представляють чотири класи.

Завантаження вхідних даних

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

Візуалізуйте вхідні дані, використовуючи чотири маркери різної форми. Нам потрібно перетворити мітки у відповідні маркери, і саме для цього призначена змінна `marker`.

Відображення вхідних даних на графіку

```
plt.figure()
plt.title('Входные данные')
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')
```

Визначимо кількість найближчих сусідів, які ми хочемо використати.

```
# Кількість найближчих сусідів
num_neighbors = 12
```

Визначимо крок сітки, яку будемо використовувати для візуалізації

```
step_size = 0.01
```

Створимо модель класифікатора методом *k* найближчих сусідів.

```
# Створення класифікатора на основі методу k найближчих сусідів
classifier = neighbors.KNeighborsClassifier(num_neighbors,
weights='distance')
```

Навчимо модель, використовуючи тренувальні дані.

```
# Навчання моделі на основі методу k найближчих сусідів
classifier.fit(X, y)
```

Створимо поділки, які використовуватимемо для візуалізації сітки.

```
# 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))
```

Виконаємо класифікатор на всіх точках сітки.

```
# Виконання класифікатора на всіх точках сітки
output = classifier.predict(np.c_[x_values.ravel(),
y_values.ravel()])
```

Створимо сітку з виділенням кольорів областей для візуалізації результату.

```
# Візуалізація передбачуваного результату
```



```

output = output.reshape(x_values.shape)
plt.figure()
plt.pcolormesh(x_values, y_values, output, cmap=cm.Paired)

```

Відобразимо навчальні дані поверх колірної карти для візуалізації розташування точок щодо меж.

```

# 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')

```

Задамо граничні значення для осей X та Y та вкажемо заголовок.

```

plt.xlim(x_values.min(), x_values.max())
plt.ylim(y_values.min(), y_values.max())
plt.title('Границы модели классификатора на основе K
          ближайших соседей')

```

Щоб оцінити ефективність класифікатора, визначимо тестову точку даних. Відобразимо на графіку навчальні точки даних разом із тестовою точкою для візуальної оцінки їхнього взаємного розташування.

```

# Тестування вхідної точки даних
test_datapoint = [5.1, 3.6]
plt.figure()
plt.title('Тестовая точка данных')
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')

```

Витягнемо K найближчих сусідів тестової точки даних, використовуючи модель класифікатора.

```

# Вилучення  $K$  найближчих сусідів
_, indices = classifier.kneighbors([test_datapoint])
indices = indices.astype(np.int)[0]

```

Відобразимо на графіку K найближчих сусідів, отриманих на попередньому кроці.

Відображення K найближчих сусідів на графіку

```
plt.figure()
plt.title('K ближайших соседей')
for i in indices:
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[y[i]],
                linewidth=3, s=100, facecolors='black')
```

Відобразимо на тому ж графіку тестову точку.

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

Відобразимо на тому ж графіку вхідні дані.

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

Виведемо прогнозований результат.

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

У процесі виконання цього коду на екрані відобразяться чотири графіки.

Занесіть їх у звіт. Підпишіть що міститься на кожному графіку.

У вікні терміналу визначте, до якого класу відноситься тестова точка.
Дані з терміналу занесіть у звіт.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_6_task_3.py
Зробіть висновок

Завдання 2.4. Обчислення оцінок подібності

При побудові рекомендаційних систем дуже важливу роль відіграє вибір способу порівняння різних об'єктів, що входять до набору даних. Припустимо, що наш набір даних включає інформацію про користувачів та їх переваги. Для того, щоб щось рекомендувати, ми повинні розуміти, як

порівнювати смаки різних людей. І тут першому плані виходять оцінки *подібності* (*similarity scores*). Оцінка подібності дає уявлення про те у якій мірі два об'єкта можуть вважатися аналогічними один одному. Для цієї мети часто використовують оцінки двох типів: евклідові і по Пірсону. В основу евклідової оцінки (метрики) покладено відстань між двома точками даних. Якщо вам необхідно освіжити знання щодо того, що таке евклідова відстань, то загляньте в Вікіпедію (https://ru.wikipedia.org/wiki/Евклидова_метрика). Евклідова відстань не обмежена за величиною. Тому ми беремо відповідне значення та перетворимо його таким чином, щоб нове значення знаходилося в діапазоні від 0 до 1. Якщо евклідова відстань між двома об'єктами велика, то відповідна евклідова оцінка повинна мати невелику величину, оскільки низька оцінка вказує на малу міру подібності між об'єктами. Отже, евклідова відстань обернено пропорційно евклідовій оцінці.

Оцінка подібності за Пірсоном (Pearson score) – це математична міра кореляції двох об'єктів. Для її обчислення використовують коваріацію (covariance) двох об'єктів та їх індивідуальні стандартні відхилення (Standard deviations). Значення цієї оцінки можуть бути змінені в межах від -1 до +1. Оцінка +1 свідчить про високий рівень подібності об'єктів, тоді як оцінка -1 - великі відмінності з-поміж них. Оцінка 0 свідчить про відсутність кореляції між двома об'єктами.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

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

Створимо парсер для обробки вхідних аргументів. Ними будуть служити імена двох користувачів та тип оцінки, яка використовуватиметься для обчислення ступеня подібності.

```
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
```

Визначимо функцію, що обчислює евклідову оцінку для двох заданих наборів

```
# 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')
```

Визначимо змінну, яка використовуватиметься для відстеження фільмів, які отримали рейтингову оцінку від обох користувачів.

```
# Фільми, оцінені обома користувачами, user1 та user2
common_movies = {}
```

Витягнемо фільми, які отримали рейтингову оцінку від обох користувачів.

```
for item in dataset[user1]:
    if item in dataset[user2]:
        common_movies[item] = 1
```

У разі відсутності фільмів, оцінених обома користувачами, обчислення оцінки подібності стає неможливим.

```
# За відсутності фільмів, оцінених обома користувачами,
# оцінка приймається рівною 0
if len(common_movies) == 0:
    return 0
```

Обчислимо квадрат різниці між рейтинговими оцінками та використовуємо його для отримання евклідової оцінки.

```
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)))
```

Визначимо функцію, яка обчислює оцінку подібності за Пірсоном для двох заданих користувачів з числа включених до набору даних. У разі відсутності інформації про цих користувачів генерується виняток.

```
# Вычислим коэффициент корреляции Пирсона для user1 и 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')
```

Визначимо змінну, яка використовуватиметься для відстеження фільмів, які отримали рейтингову оцінку від обох користувачів.

```
# Фільми, оцінені обома користувачами, user1 та user2
common_movies = {}
```

Витягнемо фільми, які отримали рейтингову оцінку від обох користувачів.

```
for item in dataset[user1]:
    if item in dataset[user2]:
        common_movies[item] = 1
```

У разі відсутності фільмів, оцінених обома користувачами, обчислення оцінки подібності стає неможливим.

```
num_ratings = len(common_movies)
```

```
# За відсутності фільмів, оцінених обома користувачами,
# оцінка приймається рівною 0
```

```

if num_ratings == 0:
    return 0

```

Обчислимо суму рейтингових оцінок усіх фільмів, оцінених обома користувачами.

```

# 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])

```

Обчислимо суму квадратів рейтингових оцінок усіх фільмів, оцінених обома користувачами.

```

# 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])

```

Обчислимо суму творів рейтингових оцінок усіх фільмів, оцінених обома користувачами.

```

# 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])

```

Обчислимо різні параметри, необхідні обчислення оцінки подібності по Пірсону з допомогою результатів попередніх обчислень.

```

# 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
```

Повертаємо значення оцінки за Пірсоном.

```
return Sxy / np.sqrt(Sxx * Syy)
```

Визначимо основну функцію та розберемо вхідні аргументи.

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

Завантажимо рейтингові оцінки з файлу ratings.json у словнику.

```
ratings_file = 'ratings.json'

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

Обчислимо оцінку подібності виходячи з вхідних аргументів.

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

Виконаємо цей код для деяких комбінацій вхідних даних. Припустимо, ми хочемо обчислити евклідову оцінку подібності користувачів David Smith та Bill Duffy.

```
$ python3 compute_scores.py --user1 "David Smith" --user2  
"Bill Duffy" --score-type Euclidean
```

Після виконання наведеної команди у вікні терміналу відобразиться інформація.

Інформацію занесіть у звіт.

Щоб обчислити оцінку подібності Пірсона для тієї ж пари користувачів, виконайте наступну команду.

```
$ python3 compute_scores.py --user1 "David Smith" --user2  
"Bill Duffy" --score-type Pearson
```

У вікні терміналу відобразиться інформація.

Інформацію занесіть у звіт.

Виконайте аналогічні команди для інших поєднань параметрів

Обчисліть оцінки для

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

Інформацію занесіть у звіт.

Збережіть код робочої програми з обов'язковими коментарями під назвою LR_6_task_4.py

Код програми та рисунок занесіть у звіт.

Зробіть висновок

Завдання 2.5. Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації

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.

Складаючи прогнози для переваг індивідуальних користувачів, ми використовуємо спільну інформацію про уподобання інших користувачів. Саме тому цей метод фільтрації називається колаборативним.

В даному випадку основне припущення полягає в тому, що якщо дві людини дають однакові рейтингові оцінки деякому набору фільмів, то їх оцінки фільмів з невідомого набору також будуть приблизно однаковими. Знаходячи загальні оціночні судження щодо одних фільмів, ми можемо прогнозувати оцінки щодо інших фільмів. З попереднього завдання ми дізналися, як порівнювати між собою різних користувачів у межах одного набору даних. Ці методики оцінки подібності використовуються для пошуку користувачів зі схожими уподобаннями в нашому наборі даних. Як правило, колаборативне фільтрування застосовують, коли мають справу з наборами даних великого розміру. Методи такого типу можна використовувати в різних областях, включаючи фінансовий аналіз, онлайн-покупки, маркетингові дослідження, вивчення купівельних звичок і т.п.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import argparse
import json
import numpy as np
from compute_scores import pearson_score
```

Визначимо функцію для парсингу вхідних аргументів. У разі єдиним вхідним аргументом є ім'я користувача.

```
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
```

Визначимо функцію, яка знаходитиме в наборі даних користувачів, аналогічних зазначеному. Якщо інформація про вказаний користувач відсутня, генерується виняток.

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

Ми вже імпортували функцію, необхідну для обчислення оцінки подібності Пірсона. Обчислимо з її допомогою оцінку подібності за Пірсоном між вказаним користувачем та всіма іншими користувачами у наборі даних.

```
# 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])
```

Відсортуємо оцінки щодо спадання.

```
# Сортування оцінок за спаданням
scores_sorted = np.argsort(scores[:, 1])[::-1]
```

Витягнемо перші num_users користувачів і повернемо масив.

```
# Вилучення оцінок перших 'num_users' користувачів
top_users = scores_sorted[:num_users]
return scores[top_users]
```

Визначимо основну функцію і розберемо вхідні аргументи, щоб витягти ім'я користувача.

```
if __name__ == '__main__':
    args = build_arg_parser().parse_args()
    user = args.user
```

Завантажимо дані з файлу ratings.json, в якому містяться імена користувачів та рейтингові оцінки фільмів.

```
ratings_file = 'ratings.json'

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

Знайдемо перших трьох користувачів, аналогічних користувачеві, вказаному за допомогою вхідного аргументу. Якщо хочете, можете вказати як аргумент іншу кількість користувачів. Виведемо імена користувачів разом із оцінками подібності.

```
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))
```

Виконайте цей код і знайдіть користувачів, аналогічних користувачеві Bill Duffy.

```
$ python3 collaborative_filtering.py --user "Bill Duffy"
```

У вікні терміналу з'явиться інформація про подібних користувачів.

Інформацію занесіть у звіт.

Виконайте той самий код і знайдіть користувачів, аналогічних користувачеві Clarissa Jackson.

Інформацію занесіть у звіт.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_6_task_5.py

Код програми занесіть у звіт.

Зробіть висновок

Завдання 2.6. Створення рекомендаційної системи фільмів

Створіть рекомендаційну систему на основі даних, наданих у файлі ratings.json. У цьому файлі міститься інформація про користувачів та оцінки, дані ними різним фільмам. Щоб рекомендувати фільми конкретному користувачу, ми повинні знайти аналогічних користувачів у наборі даних та використовувати інформацію про їх переваги для формування відповідної рекомендації.

Створіть новий файл Python та імпортуйте такі пакети.

```
import argparse
import json
import numpy as np
from compute_scores import pearson_score
from collaborative_filtering import find_similar_users
```

Визначимо функцію для парсингу вхідних аргументів. У разі єдиним вхідним аргументом є ім'я користувача.

```
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
```

Визначимо функцію, яка отримуватиме рекомендації для зазначеного користувача. Якщо інформація про вказаного користувача відсутня у наборі даних, генерується виняток.

```
# 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')
```

Визначимо змінні для відстеження оцінок.

```
overall_scores = {}
similarity_scores = {}
```

Обчислимо оцінку 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
```

Вилучимо список 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]
```

Відслідкуємо зважену рейтингову оцінку для кожного елемента відфільтрованого списку, виходячи з оцінок подібності. Також відстежимо оцінки подібності.

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

У разі відсутності відповідних фільмів, ми не можемо надати жодних рекомендацій.

```
if len(overall_scores) == 0:
    return ['No recommendations possible']
```

Нормалізуємо оцінки на підставі виважених оцінок.

```
# Генерація рейтингів фільмів за допомогою їх нормалізації
movie_scores = np.array([[score/similarity_scores[item],
                           item] for item, score in overall_scores.items()])
```

Виконаємо сортування оцінок та отримаємо рекомендації фільмів.

```
# 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
```

Визначимо основну функцію і проаналізуємо вхідні аргументи, щоб витягти ім'я зазначеного користувача.

```
if __name__ == '__main__':
    args = build_arg_parser().parse_args()
    user = args.user
```

Завантажимо дані про рейтинг фільмів із файлу ratings.json.

```
ratings_file = 'ratings.json'

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

Вилучимо рекомендації фільмів та виведемо результати.

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

Знайдемо рекомендації для користувача Chris Duncan.

```
$ python3 movie_recommender.py --user "Chris Duncan"
```

У вікні терміналу з'явиться інформація.

Інформацію занесіть у звіт.

Знайдіть рекомендації фільмів для користувача Julie Hammel.

Інформацію занесіть у звіт.

Знайдіть рекомендації фільмів для користувача Chris Duncan.

Інформацію занесіть у звіт.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_6_task_6.py

Код програми занесіть у звіт.

Зробіть висновок

Коди комітити на GitHub. У кожному звіті повинно бути посилання на GitHub.

Назвіть бланк звіту СШІ-ЛР-6-NNN-XXXXX.doc

де NNN – позначення групи

XXXXX – позначення прізвища студента.

Переконвертуйте файл звіту в СШІ-ЛР-6-NNN-XXXXX.pdf