Recommender system with collaborative filtering

Студент: Кристиян Кръчмаров
Факултетен Номер: 791324005

Какво е Recommender system

Recommender system или Система за препоръки е система която помага на потребител да намери най-подходящите опции, когато търси нещо, било то в сайт за електронна търговия или платформа за развлечение.

Видове Система за препоръки

Има два основни вида системи: персонализирани и неперсонализирани. Неперсонализираните са прости, но персонализираните работят по добре, защото отговаря на нуждите на всеки потребител.

Collaborative filtering

Collaborative filtering е метод за извличане на информация, който се основава на анализа на предпочитания или поведението на потребителите. Този метод разчита на информацията от множество потребители, за да направи препоръка.
Има два основни вида филтриране

- Базирано на потребители
- Базирано на елементи

Филтрирането базирано на потребители работи на предположението че потребители, които са уцени един и същ предмет с подобни оценки, то те вероятно ще имат едно и също предпочитание за други елементи. Този метод разчита на намирането на прилики между потребителите.

Филтрирането базирано на елементи работи като сравнява елементи, които са оценени сходно от различни потребители. Като пример: Ако много хора, които са гледали Филм А, също са гледали Филм Б, можем да препоръчаме Филм Б на потребители, които са гледали Филм А.

Ще реализираме филтриране базирано на потребители

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
```

Данни

Данните които са използвани са оценки на филми на Амазон, които се намират в Kaggle.

In [2]:
```
data = pd.read_csv("Amazon.csv")

print(data.shape)
data.head()
```

```
(4848, 207)
```

Out[2]:

	user_id	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movi
0	A3R5OBKS7OM2IR	5.0	5.0	NaN	NaN	NaN	NaN	NaN	Na
1	AH3QC2PC1VTGP	NaN	NaN	2.0	NaN	NaN	NaN	NaN	Na
2	A3LKP6WPMP9UKX	NaN	NaN	NaN	5.0	NaN	NaN	NaN	Na
3	AVIY68KEPQ5ZD	NaN	NaN	NaN	5.0	NaN	NaN	NaN	Na
4	A1CV1WROP5KTTW	NaN	NaN	NaN	NaN	5.0	NaN	NaN	Na

5 rows × 207 columns

Първо ще дефинираме функция за базова оценка която е глобалната средна оценка с добавени отклонения за потребителя и елемента

In [3]:
```
def baseline_prediction(data, userid, movieid):
    global_mean = data.stack().dropna().mean()

    user_mean = data.loc[userid, :].mean()

    item_mean = data.loc[:, movieid].mean()

    user_bias = global_mean - user_mean

    item_bias = global_mean - item_mean

    baseline = global_mean + user_bias + item_bias

    return baseline
```

След това ще дефинираме функция която намира съседите на база оценката на сходство.

Първо нормализираме оценките като извадим средните оценки, за да се отчетат различните скали на оценяване между потребителите.

За да намерим сходимостта между оценките на потребителя за който се създава препоръка и оценките на всички потребители ще използваме косинусово сходство или cosine similarity. Косинусовото сходство е мярката определяне колко са сходни два вектора.

$$\text{Cosine Similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

$$\mathbf{A} \cdot \mathbf{B} = \sum_{i=1}^{n} A_i \cdot B_i$$

$$\|\mathbf{A}\| = \sqrt{\sum_{i=1}^{n} A_i^2}, \qquad \|\mathbf{B}\| = \sqrt{\sum_{i=1}^{n} B_i^2}$$

Накрая връщаме "съседите" на дадения потребител и техните оценки на сходство.

Този метод представлява имплементация на "User-KNN". В конкретния случай K, което представлява броят съседи които искаме е `neighbours_count`. За разстоянието между съседите е използвано косинусовото подобие.

Основните разлики със стандартния KNN са, че в случая връщаме съседите и техните оценки на сходство, а типичния KNN ще върне само съседите. Друга разлика е, че се работи с матрица за оценки, а не с вектор на характеристиките.

In [4]:
```
def find_neighbour(data, userid, neighbours_count=5):
    user_mean = data.mean(axis=0)
    user_removed_mean_rating = (data - user_mean).fillna(0)

    n_users = len(user_removed_mean_rating.index)
    similarity_score = np.zeros(n_users)

    user_target = user_removed_mean_rating.loc[userid].values.reshape(1, -1)

    for i, neighbour in enumerate(user_removed_mean_rating.index):
        user_neighbour = user_removed_mean_rating.loc[neighbour].values.reshape(1,

        sim_i = cosine_similarity(user_target, user_neighbour)

        similarity_score[i] = sim_i[0, 0]

    sorted_idx = np.argsort(similarity_score)[::-1]

    similarity_score = np.sort(similarity_score)[::-1]

    closest_neighbour = user_removed_mean_rating.index[sorted_idx[1:neighbours_coun
```

```
    neighbour_similarities = list(similarity_score[1:neighbours_count + 1])

    return {
        'closest_neighbour': closest_neighbour,
        'closest_neighbour_similarity': neighbour_similarities,
    }
```

Следващия метод който е необходим е метод който да прогнозира как потребителя би оценил конкретния елемент.
Започвайки с базовата оценка се гледа за всеки съсед на потребителя как той е оценил елемента. Ако има оценка то се изчислява разликата между реалната оценка и базовата стойност на съседа за този елемент. Тази разлика се използва за претегляне на сходството между потребителя и конкретния съсед. Прогнозната оценка се коригира с базовата оценка, ако няма сходство (`similarity_sum == 0`). Накрая оценката се ограничава зададения диапазон (`min_rating` и `max_rating`)

In [5]:
```
def predict_item_rating(userid, movieid, data, neighbour_data, neighbour_count, min
    baseline = baseline_prediction(data, userid, movieid)

    similarity_rating_total = 0
    similarity_sum = 0

    for i in range(neighbour_count):
        neighbour_rating = data.loc[neighbour_data['closest_neighbour'][i], movieid

        if np.isnan(neighbour_rating):
            continue

        neighbour_baseline = baseline_prediction(data, neighbour_data['closest_neig

        adjusted_rating = neighbour_rating - neighbour_baseline

        similarity_rating = neighbour_data['closest_neighbour_similarity'][i] * adj

        similarity_rating_total += similarity_rating

        similarity_sum += neighbour_data['closest_neighbour_similarity'][i]

    # Prevent invalid division
    if similarity_sum > 0:
        user_item_prediction_rating = baseline + (similarity_rating_total / similar
    else:
        user_item_prediction_rating = baseline

    # Clip prediction to within allowed range
    user_item_prediction_rating = max(min(user_item_prediction_rating, max_rating),

    return user_item_prediction_rating
```

В последната функция ще намерим най-близките съседи и ще прогнозираме оценките на всички непознати елементи. Ако `recommend_seen = True` то ще прогнозираме и

оценките на всички познати продукти. След това сортираме и връщаме `items_count` на брой елемента.

In [6]:
```
def recommend_items(data, userid, neighbours_count, items_count, recommend_seen=Fal
    neighbour_data = find_neighbour(data=data, userid=userid, neighbours_count=neig

    prediction_df = pd.DataFrame()

    predicted_raitings = []

    mask = np.isnan(data.loc[userid])

    items_to_predict = data.columns[mask]

    if recommend_seen:
        items_to_predict = data.columns

    for movie in items_to_predict:
        predictions = predict_item_rating(userid=userid, movieid=movie, data=data,
                                          neighbour_count=5)

        predicted_raitings.append(predictions)

    prediction_df['movieId'] = data.columns[mask]

    prediction_df['predictions'] = predicted_raitings

    prediction_df = prediction_df.sort_values('predictions', ascending=False).head(

    return prediction_df
```

Тук се прочитат данните и се настройва колоната `user_id` да е индекс, което означава че тази колона няма да се счита за колона с данни. С това се улеснява достъпа до данните

In [7]:
```
# read data
data = pd.read_csv("Amazon.csv")

# dataframe index
data = data.set_index('user_id')
```

Генериране на примерна препоръка за първия наличен потребител в данните

In [8]:
```
user_recommendation = recommend_items(data=data, userid="A3R5OBKS7OM2IR", neighbour
                                      recommend_seen=False)

print(user_recommendation)
```

```
      movieId  predictions
0      Movie3          5.0
57    Movie60          5.0
42    Movie45          5.0
156  Movie159          5.0
50    Movie53          5.0
```

In [9]:
```
users = data.index.to_series().sample(n=5).tolist()

for user in users:
    recomendation = recommend_items(data=data, userid=user, neighbours_count=5, ite
    print("Recomendations for user: {}".format(user))
    print(recomendation)
    print("\n")
```

```
Recomendations for user: A16IIQA8V9IQS5
      movieId  predictions
50    Movie52          5.0
139  Movie141          5.0
62    Movie64          5.0
136  Movie138          5.0
60    Movie62          5.0


Recomendations for user: A17DBY927PTFOB
      movieId  predictions
72    Movie73          5.0
44    Movie45          5.0
88    Movie90          5.0
152  Movie154          5.0
19    Movie20          5.0


Recomendations for user: A2EMKC5VI5MZXB
    movieId  predictions
44  Movie45          5.0
57  Movie58          5.0
63  Movie64          5.0
72  Movie73          5.0
52  Movie53          5.0


Recomendations for user: A3QIB5PMK3NY4V
      movieId  predictions
0      Movie1            5
141  Movie143            5
131  Movie133            5
132  Movie134            5
133  Movie135            5


Recomendations for user: A2ANDU9EQ4B09K
    movieId  predictions
63  Movie64          5.0
57  Movie58          5.0
89  Movie90          5.0
25  Movie26          5.0
82  Movie83          5.0
```

In [ ]:

In [ ]: