Innovative Movie Recommendation System for Diverse Groups

Vesa Haaparanta

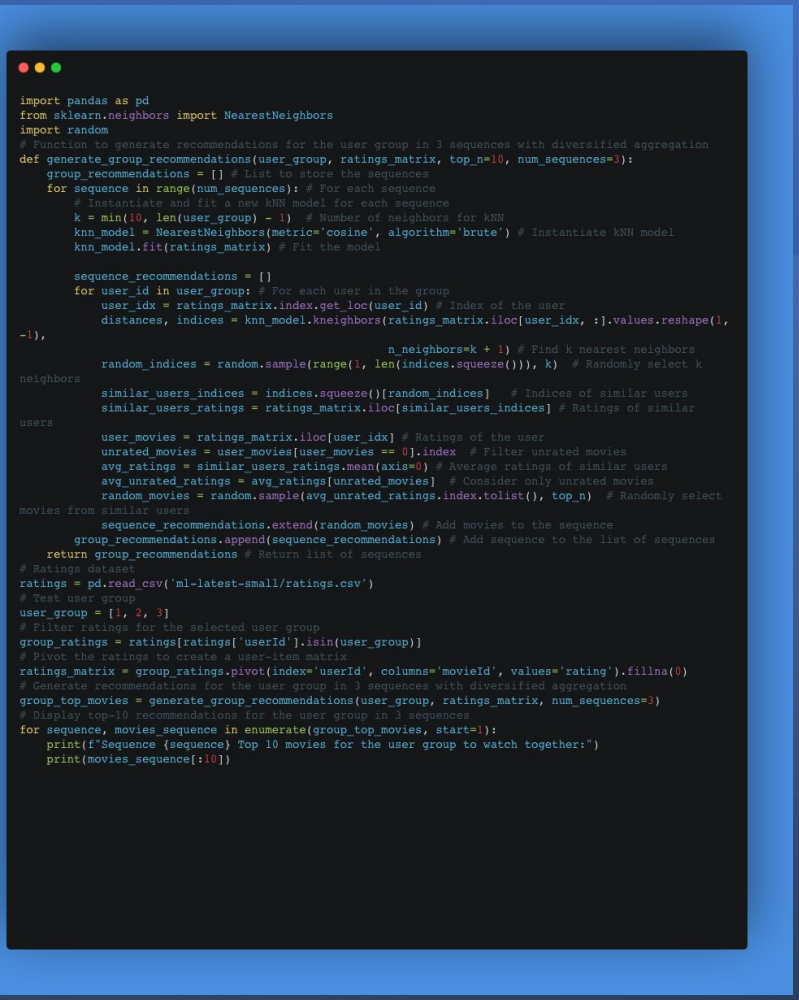
Martti Grönholm

The Intricacy of Group Movie Selection

- Imagine this: Mikko is passionate about sci-fi thrillers, Aino adores heartfelt dramas, and Leevi is fascinated by historical epics. However, Mikko finds dramas dull, and Aino isn't interested in historical stories.
- Choosing a movie for Friday's movie night is quite a challenge. An action-packed thriller might captivate Mikko and Leevi, but it could leave Aino craving more emotional depth.
- How can we pick a film that pleases everyone? With such varied preferences, it's evident that a standard solution won't work. For our film-loving group and wider gatherings, we need a strategy that respects their unique tastes and discovers a shared cinematic interest.

Steps of this recommendation system

- **Personalized Data Matrix**
  - Compiling a bespoke matrix of movie ratings, unique to each group member.
- **Algorithmic Matchmaking**
  - Employing a k-Nearest Neighbors algorithm to identify films resonating with the group's collective taste.
- **Embracing Diversity**
  - Randomly selecting potential hits from a vast sea of unrated titles, ensuring a fresh roster every time.
- **Iterative Sequences**
  - Generating not one but multiple recommendation sequences, each offering a new bouquet of cinematic experiences.



```python
import pandas as pd
from sklearn.neighbors import NearestNeighbors
import random
# Function to generate recommendations for the user group in 3 sequences with diversified aggregation
def generate_group_recommendations(user_group, ratings_matrix, top_n=10, num_sequences=3):
    group_recommendations = [] # List to store the sequences
    for sequence in range(num_sequences): # For each sequence
        # Instantiate and fit a new kNN model for each sequence
        k = min(10, len(user_group) - 1)  # Number of neighbors for kNN
        knn_model = NearestNeighbors(metric='cosine', algorithm='brute') # Instantiate kNN model
        knn_model.fit(ratings_matrix) # Fit the model

        sequence_recommendations = []
        for user_id in user_group: # For each user in the group
            user_idx = ratings_matrix.index.get_loc(user_id) # Index of the user
            distances, indices = knn_model.kneighbors(ratings_matrix.iloc[user_idx, :].values.reshape(1,
-1),
                                                      n_neighbors=k + 1) # Find k nearest neighbors
            random_indices = random.sample(range(1, len(indices.squeeze())), k)  # Randomly select k
neighbors
            similar_users_indices = indices.squeeze()[random_indices]   # Indices of similar users
            similar_users_ratings = ratings_matrix.iloc[similar_users_indices] # Ratings of similar
users
            user_movies = ratings_matrix.iloc[user_idx] # Ratings of the user
            unrated_movies = user_movies[user_movies == 0].index  # Filter unrated movies
            avg_ratings = similar_users_ratings.mean(axis=0) # Average ratings of similar users
            avg_unrated_ratings = avg_ratings[unrated_movies]  # Consider only unrated movies
            random_movies = random.sample(avg_unrated_ratings.index.tolist(), top_n)  # Randomly select
movies from similar users
            sequence_recommendations.extend(random_movies) # Add movies to the sequence
        group_recommendations.append(sequence_recommendations) # Add sequence to the list of sequences
    return group_recommendations # Return list of sequences
# Ratings dataset
ratings = pd.read_csv('ml-latest-small/ratings.csv')
# Test user group
user_group = [1, 2, 3]
# Filter ratings for the selected user group
group_ratings = ratings[ratings['userId'].isin(user_group)]
# Pivot the ratings to create a user-item matrix
ratings_matrix = group_ratings.pivot(index='userId', columns='movieId', values='rating').fillna(0)
# Generate recommendations for the user group in 3 sequences with diversified aggregation
group_top_movies = generate_group_recommendations(user_group, ratings_matrix, num_sequences=3)
# Display top-10 recommendations for the user group in 3 sequences
for sequence, movies_sequence in enumerate(group_top_movies, start=1):
    print(f"Sequence {sequence} Top 10 movies for the user group to watch together:")
    print(movies_sequence[:10])
```

Conclusion

- We can see that there is some movies which are in multiple sequences.
- We can see that every sequence is unique to different sequences