Group Recommendation with Disagreement Handling

Martti Grönholm

Vesa Haaparanta

When groups movie tastes collide

Selecting a movie for a group can be as complex as the varied tastes of its members. When planning a movie night together, this diversity becomes a tangible challenge. Let's explore an example: Alice adores romantic comedies, Bob is thrilled by action films, and Charlie has a penchant for documentaries. The issue arises when we discover that both Bob and Charlie have a strong aversion to romantic comedies, and Bob also finds documentaries dull.

At first glance, action movies might seem like the go-to choice since they're not actively disliked by Alice and Charlie. However, this simplistic approach may overlook a genre that could be more universally appealing. Also, while manually selecting a genre might work for a few individuals, it becomes impractical with larger groups. A more sophisticated method is needed to accommodate everyone's preferences, ensuring movie night is a hit for all.

Steps what our program should do

- Identify Individual Preferences:
  - Program initially identifies movies that each user is likely to enjoy, based on their individual ratings and viewing history.
- Calculate Disagreement Scores:
  - Next, it calculates a 'disagreement score' for each movie by evaluating the differences in ratings between users. This score reflects the level of contention a movie is likely to cause within the group.
- Adjust Recommendations:
  - Utilizing the disagreement scores, the program then adjusts the initial recommendations. Movies with high disagreement scores are deprioritized to ensure that the final selection is more likely to appeal to the entire group.
- Generate Group Recommendations:

  Finally, the program presents a curated list of movie recommendations tailored to the group, aiming for a pleasurable movie-watching experience.

Identify Individual Preferences	Calculate Disagreement Scores	Adjust Recommendations	Generate Group Recommendations
This function generates movie recommendations for a specific user. It works by first finding users who are similar to the given user based on their movie ratings. Then, it looks at the movies rated by these similar users and considers them for recommendations if the original user hasn't already rated them. Each potential recommendation is weighted by the similarity of the user who rated it, ensuring that recommendations are more influenced by users who are more similar. The function finally returns a sorted list of movies with predicted ratings, indicating how much the user is expected to enjoy each movie.	This function calculates the disagreement score between two users based on their movie ratings. It works by identifying movies that both users have rated and then computing the absolute difference in their ratings for these common movies. This score represents how differently the two users feel about the movies they've both seen. The function returns the average of these absolute differences, providing a single score that quantifies the level of disagreement between the two users.	This function modifies a set of group movie recommendations based on the level of disagreement among the users in the group. It calculates the disagreement score for each pair of users in the group and then adjusts the group's movie recommendations accordingly. The idea is to slightly decrease the weight of recommendations for movies where there's higher disagreement among group members, aiming to find a more agreeable set of recommendations for everyone in the group.	This function generates movie recommendations for a group of users, taking into account the individual preferences and disagreements among the group members. It first gathers individual recommendations for each user and then combines them into a group recommendation. This group recommendation is then modified based on the level of disagreement among users, ensuring that the final list of recommendations is more likely to satisfy the entire group. The function returns the top 10 movie recommendations for the group.

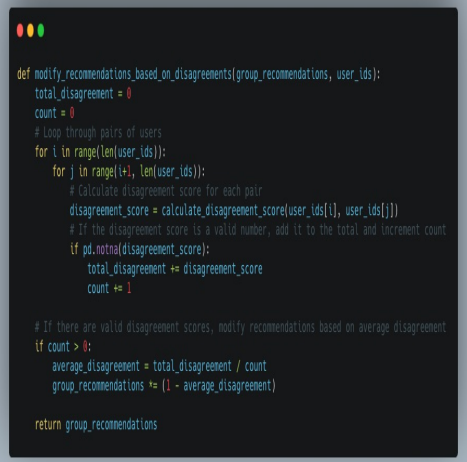
```
def recommend_movies(user):
    # Find users similar to the given user
    similar_users = find_similar_users(user)
    recommendations = {}

    # Iterate over each similar user and their similarity score
    for similar_user, similarity in similar_users:
        # Get movies rated by the similar user
        movies_rated_by_similar_user = user_item_matrix.loc[similar_user]
        for movie, rating in movies_rated_by_similar_user.items():
            # Consider the movie for recommendation if it's positively rated by the similar user and not rated by the original user
            if rating > 0 and user_item_matrix.loc[user, movie] == 0:
                if movie in recommendations:
                    # Append the similar user and their similarity score to the movie's recommendation list
                    recommendations[movie].append((similar_user, similarity))
                else:
                    # Start a new recommendation list for this movie
                    recommendations[movie] = [(similar_user, similarity)]

    recommended_movies = []
    # Calculate predicted rating for each movie based on similar users' ratings and their similarities
    for movie, similar_users in recommendations.items():
        total_similarity = sum(similarity for _, similarity in similar_users)
        if total_similarity != 0:
            weighted_rating = sum(similarity * user_item_matrix.loc[similar_user_id, movie] for similar_user_id, similarity in similar_users)
            predicted_rating = weighted_rating / total_similarity
            recommended_movies.append((movie, predicted_rating))

    # Sort the recommended movies by their predicted ratings in descending order
    recommended_movies.sort(key=lambda x: x[1], reverse=True)
    # Return the recommendations as a DataFrame
    return pd.DataFrame(recommended_movies, columns=['MovieID', 'Predicted Rating'])
```

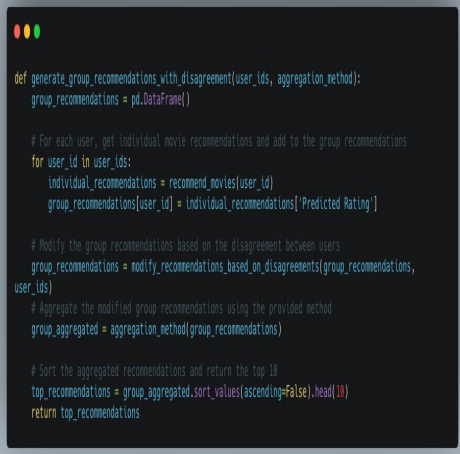
```
def calculate_disagreement_score(user1, user2):
    # Find common movies rated by both users and multiply their ratings, filling missing values with 0
    common_movies = user_item_matrix.loc[user1].mul(user_item_matrix.loc[user2], fill_value=0)
    # Calculate the absolute difference in ratings for each movie
    disagreement = (user_item_matrix.loc[user1] - user_item_matrix.loc[user2]).abs()
    # Keep the disagreement scores only for movies rated by both users
    disagreement = disagreement[common_movies != 0]
    # Return the mean disagreement score
    return disagreement.mean()
```



```
def modify_recommendations_based_on_disagreements(group_recommendations, user_ids):
    total_disagreement = 0
    count = 0
    # Loop through pairs of users
    for i in range(len(user_ids)):
        for j in range(i+1, len(user_ids)):
            # Calculate disagreement score for each pair
            disagreement_score = calculate_disagreement_score(user_ids[i], user_ids[j])
            # If the disagreement score is a valid number, add it to the total and increment count
            if pd.notna(disagreement_score):
                total_disagreement += disagreement_score
                count += 1

    # If there are valid disagreement scores, modify recommendations based on average disagreement
    if count > 0:
        average_disagreement = total_disagreement / count
        group_recommendations *= (1 - average_disagreement)

    return group_recommendations
```



```
def generate_group_recommendations_with_disagreement(user_ids, aggregation_method):
    group_recommendations = pd.DataFrame()

    # For each user, get individual movie recommendations and add to the group recommendations
    for user_id in user_ids:
        individual_recommendations = recommend_movies(user_id)
        group_recommendations[user_id] = individual_recommendations['Predicted Rating']

    # Modify the group recommendations based on the disagreement between users
    group_recommendations = modify_recommendations_based_on_disagreements(group_recommendations, user_ids)
    # Aggregate the modified group recommendations using the provided method
    group_aggregated = aggregation_method(group_recommendations)

    # Sort the aggregated recommendations and return the top 10
    top_recommendations = group_aggregated.sort_values(ascending=False).head(10)
    return top_recommendations
```

Conclusion

We can see that the top 10 recommendations for the average method and the least misery method are mostly the same for the selected user group.

However, for our method with disagreement handling, the recommended movies are different, as the method is different.

```
Assignment 2 part (a)
Top 10 Recommendations (Average Method):
[0, 12, 21, 1, 19, 18, 17, 16, 15, 14]

Top 10 Recommendations (Least Misery Method):
[0, 12, 21, 20, 19, 18, 17, 16, 15, 14]
Assignment 2 part (b)

Top 10 Recommendations Considering Disagreements:
[779, 751, 749, 748, 747, 746, 745, 744, 743, 742]
```