Documentation Programming Assignment #1
Recommender Systems 2019/1:
Collaborative Movie Recommendation

Ronald Pereira
Federal University of Minas Gerais
Belo Horizonte, Minas Gerais
ronald.pereira@dcc.ufmg.br

1 INTRODUCTION

Currently we deal with absurd amounts of data being generated at every moment on the Internet. The consumption of this data, many times, is outdated, and the overwhelming majority of this information is not even processed or made available and, even so, the amount of information available has proven to be humanly impossible to be totally consumed. For this problem to be solved, it is necessary to have some systems that indicate which contents are more significant to be consumed based on our preferences and it was in this context that the Recommender Systems were created.

2 IMPLEMENTATION

The programming language used in this project is C++ (specifically the C++11 version). This was chosen for a better understanding and readability of the code, as C++ brings with it classes to represent the various objects used with standard data structures libraries at almost no performance trade-off.

3 DATA STRUCTURES

Several data structures were used in this project as a way to optimize calculations and space occupation. But only just two standard data structure libraries were used with different combinations: $std :: unordered\_map$ and $std :: vector$.

The $std :: unordered\_map$ was used to optimize access and spend the minimum RAM space as possible, as it represents only the given input values, producing a dense representation. If a value is not found by it's key, then it returns a default value, that is 0. The main idea by using this data structure was to reduce the memory usage by the item-user ratings matrix, which was using much more memory than it needed, because it was using a sparse representation with $std :: vector$.

Therefore, the space complexity achieved by using the latter data structure was in order of $O((|U|*|I|)+|U|+|I|)$. The $|U|*|I|$ particle represents the sparse representation on the input ratings, as $|U|$ is the cardinality of the user set and $|I|$ is the cardinality of the item set. However, the matrix representation via vectors is a sequential and rigid allocation, for example the user with ID 123456 and item with ID 9239131, wouldn't be that IDs to access through the same keys. This happens because vectors have to have a sequential index starting with 0. In that way, it was needed an ID translation table to map the real user or item IDs to the vector index, which each one spent $O(|U|)$ for the user IDs and $O(|I|)$ for the item IDs.

Some research was made and with the main proposition of not having to sort by any order the input matrix, as we only needed to access specific inserted values and the ideal data structure for this optimization was found: $std :: unordered\_map$. This representation only spends $O(|U_i| * |I_j|)$, being represented only if the $user_i$ rated the $item_j$. This structure achieved to reduce the space by approximately 80% of the previous one produced by $std :: vector$ and simplified even more the access to a value given by two keys (user ID and item ID).

There's just one data structure that it not entirely composed by $std :: unordered\_map$, that it the $UserConsumedItems$ representation. It uses the $std :: unordered\_map < int, std :: vector < int >>$ structure, as we only want to get the item ID vector for a certain user ID. In that way, the complexity still unchanged ($O|U_i * I_j$), as we only insert items consumed by the user in this structure.

4 ALGORITHMS

This project approached the collaborative movie recommendation by using some different strategies.

4.1 Item-User Collaborative Filtering

As we do have more users than items in this project input ratings, an excellent computational optimization was to compute item-item similarities, instead of user-user, reducing the computational complexity from $O(|U|^2)$ to $O(|I|^2)$, as $|U| > |I|$. This strategy also showed some rating predictions improvement, because item vectors tend to be more dense than user vectors, as an item is possibly consumed by a much higher number of users than an user consumes various items, making the similarity computation much more accurate.

4.2 Matrix Mean Normalization

This is a common approach to collaborative recommender systems. It aims to normalize each user-item entrance by the user average rating, excluding the user rating tendencies for the predictions. This happens because each user may have a different way to evaluate the feedback to a certain item experience and this usually comes with an inclination to follow it's average rating. By doing that, you remove the latter, making the ratings much more informative and improving future predictions.

4.3 Adjusted Cosine Distance

The similarity computation was made by using Adjusted Cosine Distance between a certain item vector and another item vector. This was implemented because this distance metric has a built-in significance weighting as it's similarity is scaled by ratio

$$|U_i \cap U_j|/|U_i||U_j|$$

4.4 K-Nearest Neighbors

The K-Nearest Neighbors algorithm was implemented to optimize some rating predictions by picking the top-K item neighbors ranked by similarity. However, this KNN strategy gave a little RMSE (Root Mean Squared Error) increase each time we used a smaller K in the expense of making quicker predictions.

4.5 Item Cold-Start

If an item doesn't have at least one common user with any other items, no similarity can be computed. In that case, we use the item average rating as the <user, item> prediction. Even so, some items were never consumed by any user, making it's average impossible to compute. In that latter case, we use the global item average rating as the <user, item> prediction. When this was implemented, it gave the biggest improvement to the RMSE (Root Mean Squared Error) metric than any other strategy, showing the importance of a good cold-start heuristic in this project.

5 RESULTS DISCUSSION

The first submission was made by using a single value to all predictions given by the global average ratings to serve as a baseline comparison, achieving 3.98123 in the public leaderboard. All the other submissions gave a little relative improvement as some algorithms and computational optimizations was implemented in this project.

The best Kaggle submission was made using the previous algorithms with KNN disabled (all item-item similarities was used to predict the <user, item> rating) achieving 1.72739 in the public leaderboard. By not using a SVD (Singular Value Decomposition) matrix factorization strategy, this seems to be an excellent score, as it was significantly close to the top averages achieved by other students. The submission file was generated within three minutes by average, surpassing the execution limit established by the specification of maximum five minutes.