Documentation Programming Assignment #2
Recommender Systems 2019/1:
Content-based 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 The consumption of this data, many times, is outdated, and the overwhelming majority of this information is not even processed or made available and, even so, the amount of information available has proven to be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.

Although, not every recommendation is great for the user for several reasons. The biggest and most impactful of them is recommending items that does not follow the target user preferences. To solve that, we usually use a content-based approach, which tries to represent item features and user preferences on each feature to calculate the "fit"of a target user with a target item, possibly improving the recommendation.

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 only one standard data structure libraries were used with different combinations: $std :: unordered\_map$.

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 previously using a sparse representation via $std :: vector$.

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 90% 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).

4 ALGORITHMS

This project approached the content-based movie recommendation by using some different strategies.

4.1 User-item and Item-user Ratings

As I could not use any ratings for collaborative filtering ratings prediction, I built two ratings data structures to optimize Rocchio calculations and User and Item average ratings. These optimizations were just to optimize the needed for loop to access every segmentation of this data sequentially on the second inner loop, without having to iterate several times over it.

4.2 Item Features

The content-based recommendation algorithm is based in some item features to describe it. In this project, I have tried many combinations and representations of it, many times achieving worse results than the very first few tries.

The first feature selection was very simple, because it used only the "Year"item feature to describe items. This similarity improved rating predictions by a lot, just by clustering similar items by its release year in a 2D dimensional space.

The second feature selection was an inclusion of more content information about the items, by using not only the "Year"information, but also using "Metascore", "imdbRating", and "imdbVotes". This similarity was better by the latter one, but not by much. This can be explained by the preferences of users by having movies within the same time period rather than the score or popularity itself.

The third and onwards feature selections was an attempt to make embeddings to "Director", "Author", "Writer", "Actors", "Language"and "Genre"categorical features with the item description. All these attempts have made the predictions much worse than

the most basic ones. This can be only explained by the fact of the number of available items is very small to make an efficient n-dimensional representation, while maintaining close items together becomes increasingly difficult as this representation dimensions is getting bigger by aggregating more and more features.

So, the choice was to represent items by only a few information about it, given by "Year", "Metascore", "imdbRating", and "imdbVotes"informations. This was made in order to achieve a better space 5-dimensional representation of similar items, consequently achieving better rating predictions and recommendations.

4.3 Rocchio Recommendation

The user-features preferences was built by making calculations using user ratings and item features, in a way that the algorithm could express the relative force of a given feature by analyzing the ratings of similar items in that dimension. The user vector calculation can be expressed as

$$\vec{u} = \frac{1}{|R_u|} \sum_{j \in R_u} r_{uj} \, \vec{j} \tag{1}$$

by $\vec{u}$ being the user vector representation, R_u being the set of items rated by user u, r_{uj} being the rating of the user u given to the item j, and $\vec{j}$ being the item vector representation.

Then, the <user, item> rating was made by using the cosine similarity between the target user and the target item. This calculation can be expressed as

$$\hat{r}_{u,j} = \cos(\vec{u}_u, \vec{j}_j) = \frac{\vec{u}_u \cdot \vec{j}_j}{||\vec{u}_u|| ||\vec{j}_j||} = \frac{\sum_{i=1}^{n} \vec{u}_{u_i} \, \vec{j}_{j_i}}{\sqrt{\sum_{i=1}^{n} (\vec{u}_{u_i})^2} \sqrt{\sum_{i=1}^{n} (\vec{j}_{j_i})^2}} \tag{2}$$

and then, the algorithm multiply this cosine similarity by the user average to get a rating prediction between 0 and 10.

4.4 User Cold-Start

If a target user has no similarity to at least one common item with any other items, caused by the fact that the user has not consumed any item, no recommendation from Rocchio can be computed. In that case, we use the global item average rating as the <user, item> prediction. When this last three strategies was implemented, it gave the biggest improvement to the RMSE (Root Mean Squared Error) metric than any other mentioned 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 items global average ratings to serve as a baseline comparison, achieving 3.98123 in the public leaderboard. Then, it was made by using the same output from the collaborative filtering algorithm used on the first project assignment, achieving 1.72739 in the public leaderboard. But it was expected that the content-based recommendations would be worse than that latter, based on the fact that just a minor subset of contents could be effectively used as description for an item.

The best content-based Kaggle submission was made using the previous section algorithms achieving 1.78873 in the public leaderboard. By not using a SVD (Singular Value Decomposition) matrix factorization strategy nor an external source of data contents to boost item descriptions, this seems to be an excellent score, as it was significantly close to the collaborative filtering score achieved. The submission file was generated within 2 seconds by average, surpassing the execution limit established by the specification of maximum five minutes and the challenge of making it run under 5 seconds.