Recomenders Project 1: Collaborative Filtering Models

Emilio Tylson Baixauli

May 2020

1 Introduction

This document explains the implementation and the analysis of three recommender systems models based on the MovieLens (100K and 1M rows) data set. The implementation and evaluation processes follow previous studies [3] [2] [1], that are the three main papers that this project is based on. The code consist of python code and jupyter notebooks uploaded in a git repository: `https://github.com/EmilioTyl/recsys_movie_lens`

In the beginning, I explain the git repository structure besides the files and notebooks to visualize the final results.

Then in the Models section, I give a short explanation on how each of the models work and their respective implementations. Three models has been implemented. First, I present a simple and classic knn item based model [3], that computes the adjusted cosine similarity between items and selects the k nearest neighbours to predict the rating of an unrated item. The second model is called SLIM [2](Sparse Linear Method for top-N recommendations). Basically is a latent-factor model that tries to exploit the sparsity of the user-item matrix in order to compute a sparse item-item matrix. The third model is called PureSVD and it is presented in [1]. It is another latent-factor model that exploit the sparse item-user matrix in order to efficiently compute its SVD factorization. This model is interesting because it is not implemented by the minimization of the RMSE and still produces top-N recommendations.

In the next section I present three different metrics proposed in [3] [2] [1] to evaluate each of the models. [3] propose simply a mean absolute error (MAE) and root mean squared error (RMSE) to measure the performance of its memory based model. [1] suggests that neither MAE nor RMSE are representative metrics to evaluate the performance of a recommender system. The effectiveness of a recommender is to give a ranked list of proposals, it is not that important the score itself. Hence they propose evaluate the models with precision and recall on ranked lists. On the other hand [2] not only propose a model, they also propose a metric called hit rate, a variation of the aforementioned metrics, but they argue its more descriptive than others. In Evaluation and Results section,

all of the three metrics are implemented and evaluated with the models in order to study different properties of them.

Finally, conclusions of the metrics and the models.

2 Code structure

All the code is uploaded in `https://github.com/EmilioTyl/recsys_movie_lens`. In the root directory is placed two exploratory draft notebooks called **exploration_item_based** and **exploration_latent_factor**. Despite being a draft version it explains the progression of the development of the principal code. The core code of the project is placed in three python files called **movielens_dataset.py**, **recomenders.py** and **evaluation.py**.

In **movielens_dataset.py** is placed all the helper functions that compose the base dataset and functions that splits the dataset in different ways that are going to be explained later in this document. **recomenders.py** has the implementation of the three models: **ItemBasedCF**, **SLIM** and **PureSVD**. All the classes have the same method. **.fit()** is used for training the model. **all_useres_recomendation** and **all_useres_recomendation** are methods that recommend a top-N list of movies to all the users, and finally **estimate** is for estimate the rating of a given user and movie.

The final results and evaluation are shown in the **results_final_notebook** notebook. There, I proceed to train and evaluate each of the model with different metrics.

3 Models

The three presented models are collaborative filtering (CF) models, where recommendations are done based on past users behaviours. The classic item-based model [3] is memory based neighborhood approach and the other two models (SLIM [2] and PureSVD [2]) are latent-factor approaches. The neighborhood approach of item-based model relies on computing similarity between items (in this case movies) and estimate the rating of a pair user-movie by relating similar movies to the ones that the user has previously rated. The latent factor models transforms the users and items to a common latent factor space. This space tends to be smaller than the item-user matrix dimensions and has the advantage that users and item can be expressed on the same space. Hence, the rating between users and items is predicted as the inner product between user and item vectors.

3.1 Item-Based (neighborhood model)

Classic neighborhood models predicts rating based on similarity item-item ore user-user. There are different ways of computing those similarities, but in this work I focused on item-item similarity. Here, the rating of a user to an unseen/unrated item is based on the rating of the user on previous similar items.

Therefore, the training consist on pre-computing the item-item similarity matrix. In order to compute similarity of two items, the implemented model computes the cosine similarity between two item vectors. It is important to remark that this vectors are the columns of the user-item matrix. Each vector is composed on different user ratings. When the cosine similarity between vectors is computed only co-rated values by the same user are taken into account. As mentioned before the cosine similarity is used to compute similarity between vectors.

$$sim(i,j) = \frac{i}{\|i\|\|j\|} \tag{1}$$

Alternatively, [3] propose a adjusted cosine similarity. Computing simple cosine similarity does not take into account different difference rating scale between users. Hence there are some more influencing users than other. The adjusted cosine similarity subtract to each co-rated pair the mean rate of the user as following. [3] evaluated that this adjusted similarity helps the model to a better RMSE and MAE score.

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \overline{R_i})(R_{u,j} - \overline{R_j})}{\sqrt{\sum_{u \in U}(R_{u,i} - \overline{R_i})^2}\sqrt{\sum_{u \in U}(R_{u,j} - \overline{R_j})^2}} \tag{2}$$

The implementation of the code during training tried to reduce as much as possible the queries to the data frame in order to get item vectors. In a first version, the training was done by extracting each item vector from the training dataframe, and then computing each similarity value of the item-item matrix. That process was time intensive due to the times the training dataframe was consulted. For that purpose, in a second version of the model, in the beginning of the training it is computed the sparse matrix user-item (by using a dataframe pivoting function in pandas). After that the similarities are computed using the **pairwise_distances** function of **numpy**. This function efficiently computes the distance among the input vectors. Although it has built in function that computes cosine distance with sparse vectors, using this function would not be useful because it does not compute co-rated pairs. Hence, a custom cosine similarity has been implemented in order to detect the co-rated values before computing the distance. The implementation of this methodology critically improved the training time of the algorithm with respect to manually computing similarity value of the item-item matrix by querying the item vectors from the training dataframe.

For predicting new values, the model uses what in [3] is called weighted sum. It computes the prediction of an item i for a user u by computing the weighted sum of the ratings of the user on similar items to i. The weights are the similarity values between i and the rated items.

$$P_{u,i} = \frac{\sum_{\text{all similar items,N}} s_{i,N} R_{u,N}}{\sum_{\text{all similar items,N}} |s_{i,N}|} \tag{3}$$

The computation of the sparse user-item matrix was also helpful during prediction, it was only necessary to accessing to the user row and simply computing the columns that has value in order to get the rated movies by the user. This helped a lot with respect to the first iteration where the dataframe was consulted multiple times in order to get the rated movies. Additionally, the class has a parameter that uses only the k nearest movies to predict the rating.

3.2 SLIM

SLIM [2] is a latent factorization model that exploits the sparsity of the user-item user and allows to compute the item-item W matrix in parallel fashion. More over, SLIMS forces the W matrix to be sparse in order to compute the top-N recommendations efficiently and in high-speed. SLIM computes the recommendation score of an unrated item i of a user u by a sparse aggregation of items that have been rated by u

$$\tilde{r}_{i,u} = a_u^T w_i \tag{4}$$

Where a_u^T is the sparse row of the m x n item-user matrix A and w_i is the sparse column from learned sparse n x n matrix W. Therefore, the learning task can be summarized as minimizing the following system (matrix item user A is ground truth)

$$\begin{aligned} \underset{W}{\text{minimize}} \quad & \frac{1}{2}\|A - AW\|_F^2 + \frac{\beta}{2}\|W\|_F^2 + \lambda\|W\|_1 \\ \text{subject to} \quad & W \geq 0, diag(W) = 0 \end{aligned}$$

Where AW is the estimated recommendation matrix, and l_1 and $\|\|_F$ norms are the regularization terms that force the learned W to be sparse (β and λ are regularization parameters). The non negative constraint makes the algorithm to learn positive relations.

The problem can be decoupled into smaller optimization problems since columns of W are independent.

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \frac{1}{2}\|a_{u,j} - Aw_j\|_2^2 + \frac{\beta}{2}\|w_{\|2}^2 + \lambda\|w_j\|_1 \\ \text{subject to} \quad & W \geq 0, w_{j,j} = 0 \end{aligned}$$

The main difference with respect to the aforementioned methods is that it is not pre-specified a similarity function, the model learns the similarity between elements. This can potentially produce richer relations between items. Finally, the addition of l_1 and the regularization terms makes the matrix W sparse and helps to capture informative signals making it more invariant to noise.

In order to learn the weights the code applies **ElasticNet** from **sklearn** library. It applies the regularization terms along with stochastic gradient descent (SGD). The code applies eq. with the **ElasticNet** model to learn each column of the sparse matrix W. In that way it exploits the sparse matrix user-item. For top-N recommendation simply applies the sparse matrix operation AW. Using

scipy sparse matrix implementation, the user-matrix A is stored in compressed sparse row format (csr), making easy and fast to consult what are the already rated movies of each user (only need to consult to the *indices* parameters of csr matix). Those movies are filtered from the predicted recommendation of each user and sorted in a way that retrieves the top-N recommendations.

3.3 PureSVD

PureSVD proposed in [1] is a simple SVD factorization that exploits the sparsity of the user-item matrix. Their hypothesis is that minimizing RMSE and MAE does not necessary makes the algorithm to recommend useful top-N recommendations. Hence using the simple latent-factor SVD model by imputing the non rated items with 0 is not necessary a problem, it may not produce precise ratings but will learn good relations between items in order to bring useful top-N recommendations. The algorithm simply factorize the A user-item using sparse implementation of SVD.

$$A = U \cdot \Lambda \cdot Q^T \tag{5}$$

Where U is m x t matrix. Λ is a diagonal matrix with singular values and Q^T is a t x n matrix. Then user factors are $P = U \cdot \Lambda$ and item factors are Q^T. With algebraic manipulation we can derive that

$$P = U \cdot \Lambda = R \cdot Q \tag{6}$$

Therefore the predicted rated matrix is

$$\bar{A} = A \cdot Q \cdot Q^T \tag{7}$$

The code uses **scipy.linag.svds** implementation. This project will use PureSVD with 40 latent factors.

4 Metrics

Following I will talk deeply about the metrics used in this work.

4.1 Hit Rate

Hit rate is the evaluation metric proposed by [2]. The dataset is split using Leave One Out strategy where for each user one item is moved to the test set. Therefore, the test set has one movie rating for each user. After training the model, it generates a top-N list. A hit is computed for each user if the test item is in the top-N, otherwise it is a miss. The hit rate HR metric is then computed as:

$$HR = \frac{\#\text{hits}}{\#\text{users}} \tag{8}$$

In addition, in the evaluation hit rate is evaluated separately to each ground truth rating. It is to say, the HR is computed for 5 star ground truth rating, for

4 stars rating and so on. I have found valuable to observe the HR performance on each ground truth rather aggregating 1 star movies with 5 stars in one metric. Authors aregue that this metric is more meaningful than the precision/recall, and RMSE/MAE, since it measures the performance on top-N lists computed from all the item sets(and not from a pre selection of random 1000 items like I will show in precision/recall)

4.2 Precision/Recall

[1] presents precision and recall metric as an alternative to the RMSE/MAE, arguing that what is meaningful of a recommender to learn a relative order in a top-N list rather than precision on the estimated score. The authors agree that precision recall metrics gives information about how meaningful the top-N list is to the user. The first point for applying this metric is the way they split the dataset. First the dataset is split into train and test T (more precisely 1.14% to the test set, they also uses the MovieLens dataset). From the test dataset, only 5 rating movies are kept in order to state that test set has only relevant items to the users. After training the model with train dataset, for each item i from the test set, the followings steps are done:

- Select randomly 1000 unrated items for the user u. This assumes that probably most of the items are of less interest than the item i
- Predict ratings for the 1000 unrated items and item i
- Form a ranked N list with the item and compute the parameter p that is the position of item i in the ranked list
- Compute a hit if p is less than N (the item is inside the top-N list)

After computing hits, precision and recall is computed as follows.

$$\text{recall}(N) = \frac{\#hits}{|T|} \tag{9}$$

$$\text{precision}(N) = \frac{\#hits}{N \cdot |T|} = \frac{\text{recall}(N)}{N} \tag{10}$$

4.3 MAE and RMSE

Both metrics measure the average error between the predicted metric and the ground truth. This metric does not give information about how the model produces top-N recommendations.

$$\text{RMSE} = \frac{1}{n}\sum_{t=1}^{n} e_t^2 \tag{11}$$

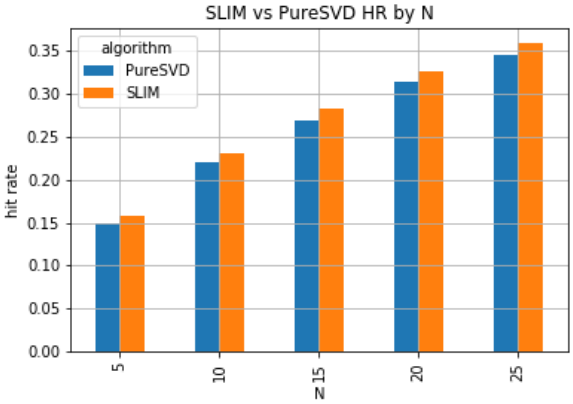
$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n} |e_t| \tag{12}$$

5 Evaluation and Results

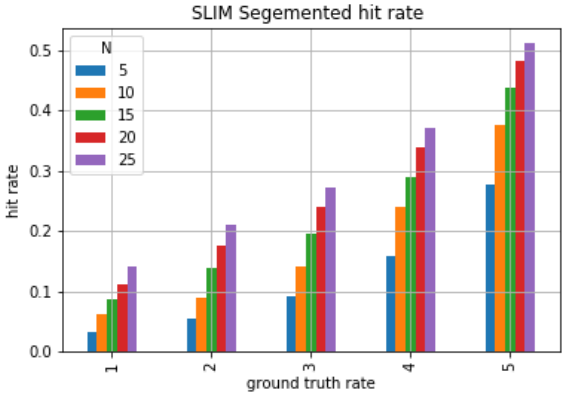
In this section I will evaluate models according to the presented metrics. SLIM and PureSVD are trained and evaluated on MovieLens-1M (one million of ratings). Due to memory and computation restrictions, the item-based neighborhood model is trained and evaluated on the MovieLens-100k (100k of rows). In order to compare item-based model with SLIM and PureSVD, both are also trained on MovieLens-100k. SLIM parameters have been found using grid search. PureSVD uses 40 latent factors similar to the 50 used in [1]. The item-based used same parameters as [3].

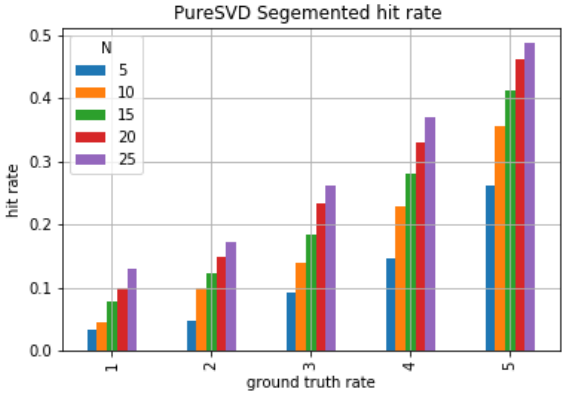
5.1 Hit Rate analysis

The dataset is split with leave one out strategy, as explained above. First, I evaluated SLIM and PureSVD hit rate on different top-N sizes. SLIM model reaches similar results as [2]. Reasonably the hit rate metric increases with N as the movie has better chances to be included in the list. Globally SLIM has better hit rate than PureSVD.

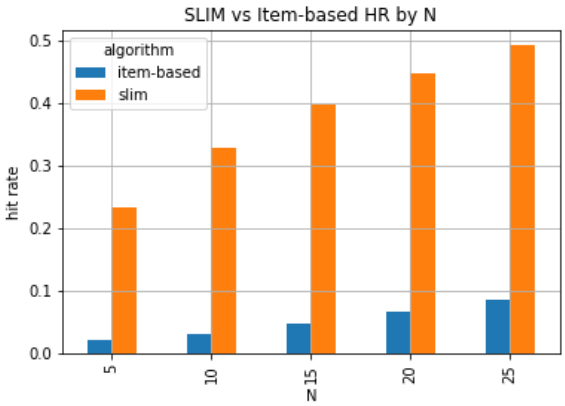


The aggregated hit rate evaluates hits of 1 rated movies as the same level as 5 rated movies. What is desirable of an recommender algorithm is to have a low hit rate of 1 star movies and a high hit rate on 5 start movies. Therefor, in order to observe the hit rate evaluation on each ground truth rating I proceed on segmenting the test set and compute hit rate of SLIM and Pure SVD.

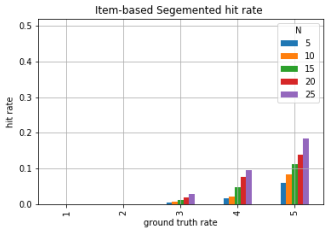


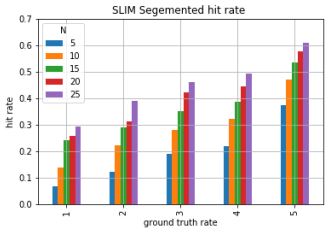


The above plots better explains the performance of the models, the hit rate of 1 or 2 rated movies is less than 0.1 in top-10 recommendations in both algorithm. On the other hand 4 or 5 star movies has grater HR, above 0.4. This tells the models are better on recommending high star movies than low rated movies. Again, despite segmenting on the ground truth ratings, SLIM is still better than PureSVD in all categories. In order to include the item-base neighborhood algorithm I trained and evaluated it with MovieLens-100k that has fewer ratings. Additionally SLIM model was trained and evaluated with the same dataset in order to perform comparisons. The following compares two models in aggregated HR metric. Item-based model has a poor HR performance with respect to SLIM that clearly surpasses on every N value.



Following the plots that explains deeply both model performances with the segmented HR metric. Clearly item-based model has a better HR as the ground truth label increases, meaning that recommenders more higher rated movies than low rates, but still its hit rate is poor with respect to the SLIM model in the same dataset.





6 Precision and Recall

In this evaluation I split the dataset as [1] explains. The test dataset is 1.4% of the training, and only 5 rated movies are kept. The dataset has small set of popular movies and a along tail of movies with not many ratings. It is important to understand the performance on both groups because movies from the long tail provides novelty to the top-N recommendations. Hence, the test dataset is also split between T_{head}, movies that represent the 33% of ratings and T_{long}, movies from the long tail. The first pair of plots is the evaluation of the SLIM and PureSVD recall against N, the size of top-N recommendations. The second row of plots shows the relation between precision and recall.

In terms of recall in function of N, the plot shows recall in different splits of the test dataset. In general aspects SLIM model performs slightly better than PureSVD. For instance, the recall of SLIM at N=10 is about 0.47, meaning that the model has a probability of 47% to provide an interesting movie in a top-10 whereas the PureSVD has a probability of 44% recommending an appealing movie at top-10 list. It is also interesting the different recalls on differents splits of test set. Testing with the full test dataset gives a general idea of the models.

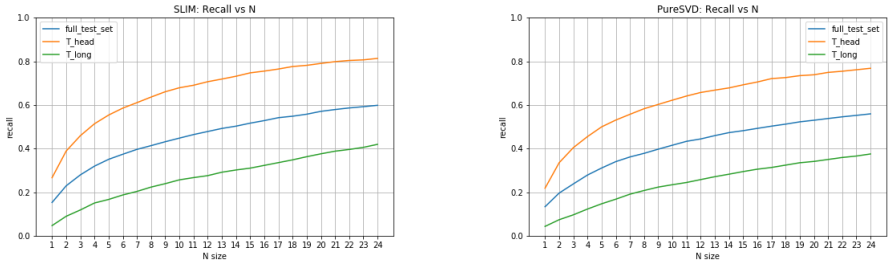


Figure 1: Left: SLIM recall against N Right: PureSVD recall against N

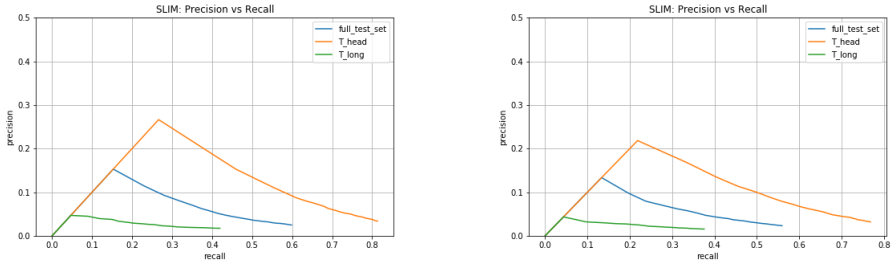


Figure 2: Left: SLIM precision against recall Right: PureSVD precision against recall

But also we can conclude that each model is better in recommending popular movies than non popular. SLIM algorithm has a recall over 0.6 for popular movies since the top-6 whereas PureSVD surpasses the same recall since top-10. In term of non popular movies, both algorithms are more tied, having a recall of around 0.25 in top 10. A better look on the aforementioned differences can be seen in the following plots that compare the T_{head} and T_{long} recall of both models in the same plot.

In terms of precision of recall trade-off both models have almost the same behaviour in any of the splits, the maximum precision is about 0.15 and it decreases as the recall increases.

7 MAE and RMSE

In this final evaluation all algorithms are trained and evaluated with MovieLens-100k. As we can see in the following plots, knn item-based with adjusted cosine similarity reached the best performances among all models and the performance as [3]. Although SLIM is trained in order to reduce the RMSE, the restrictions make it not perform well in the test detest. I showed in the hit-rate section that the knn item-based model does not perform as good top-N recommendations as

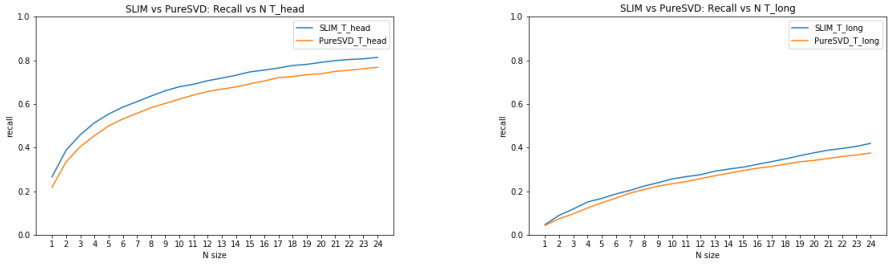


Figure 3: Left: SLIM vs PureSVD recall in T head Right: SLIM vs PureSVD recall in T long

other models, but is capable on estimating an more accurate possible rating for the movies with a +/- 0.7 error.

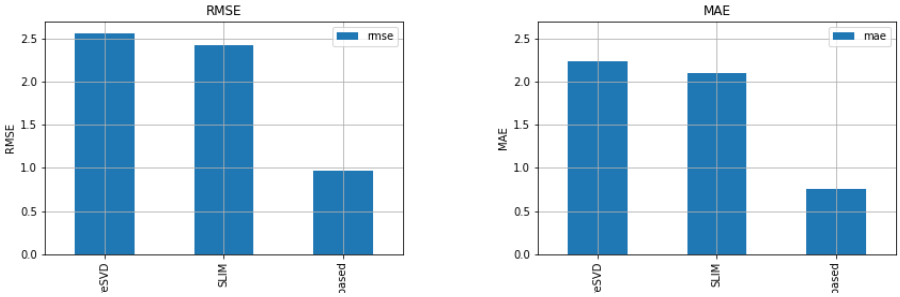


Figure 4: Left: RMSE comparison Right: MAE comparison

8 Conclusion

This report provides a short overview of the implementation of memory-based and model-based models for an item-based recommendation. I tried to show the advantages and properties of the models along with different methods of evaluation. One important aspect of the models to deal big datasets is the ability to deal and exploit sparsity of the matrix. SLIM has the advantage that can be parallelized and produces a sparse similarity matrix. On the other hand PureSVD takes advantages of the sparsity of the user-item matrix and produces a fast low dimension factorization, because it centres on capturing the relative relation between of elements rather predicting an accurate score. As I have shown, models such as knn item-based that minimize RMSE and MAE metrics does not necessary provide good quality top-N recommendations. Hence, the necessity of other metrics that better reflect the performance of the model on recommending a list of items. This work implements Hit Rate and

Precision/Recall as a way for studying critical points on its implementation and general properties to take into attention when selecting a metric. In a general aspects, the split of the train and test dataset has an important role in the training and evaluation. I have shown that HR needs to separate at least one movie for each user using a leave one out strategy while precision recall suggest a small partition of the test set and keep only five stars ratings. The train dataset should also take into account all users and all items, and a careful study on the popularity of the items should be done in order to understand how they biased the predictions. HR metric is simple metric to implement for understanding the top-N recommendations. In the aggregated form has the disadvantage that counts the same weight the 1 ranked hits as the 5 raked hits. Hence, a better visualization of the metric is done by segmenting the ground truth ranks. In this way one can observe the behaviour of the model on different ground truth metrics, and check the low hit rate on low ranked movies. On the other hand Precision/Recall requires keeping only 5 star ranked movies, and assume they are the most relevant movies for the users. Since ratings are subjective and each user has its own scale one user can rate what he consider relevant with other scales. Hence I find Precision/recall more optimistic than HR, but I find the segmented HR metric describe the behaviour of the model in different rate scales. One interesting analysis is the split into popular and non popular movies proposed in the Precision/Recall section. In that way one can have evaluate how probable the model can suggest novel movies to the user. Interestingly, the gap between SLIM and PureSVD is smaller in the long tail split than in the head split. In that way one can understand that SLIM is skewed toward popular movies. Both metrics provides different insights of the model and can be complemented to have a deeply understanding on the top-N recommendation lists.

References

[1] Gabriel Vargas Carmona and Eduard Heindl. Performance of recommender algorithms on top-n recommendation tasks. 2012.

[2] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pages 497–506. IEEE, 2011.

[3] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295, 2001.