A brief look into recommender systems

Author: Pablo Sanabria Quispe (psanabria@uc.cl)
Course: Recommender Systems - IIC3633, 2016

Introduction

Recommender systems or recommendation systems (sometimes replacing "system" with a synonym such as platform or engine) are a subclass of information filtering system that seek to predict the "rating" or "preference" that a user would give to an item.

Recommender systems have become extremely common in recent years, and are utilized in a variety of areas: some popular applications include movies, music, news, books, research articles, search queries, social tags, and products in general. However, there are also recommender systems for experts, collaborators, jokes, restaurants, financial services, life insurance, romantic partners (online dating), and Twitter pages [1].

In this work I have studied different Collaborative Filtering Algorithms in order to recommend and predict ratings for a custom dataset based on a predefined training set and try to achieve the best results in a competition organized for this course.

Dataset

The dataset consists in two files containing items, users and their ratings, one file has a training dataset for the rating test and the other one has a training dataset for rankings, this last one file is more than 10 times bigger than the rating dataset.

In order to analyze both datasets I built some custom classes to read the CSV files and classify it, the Classifier class has some utilities that can summarize both datasets and get the total submissions by user or item, also it can get the average, the average per item and the average per user. The analysis for the rating data set can be summarized in the Figure 1, Figure 2 and Table 1.

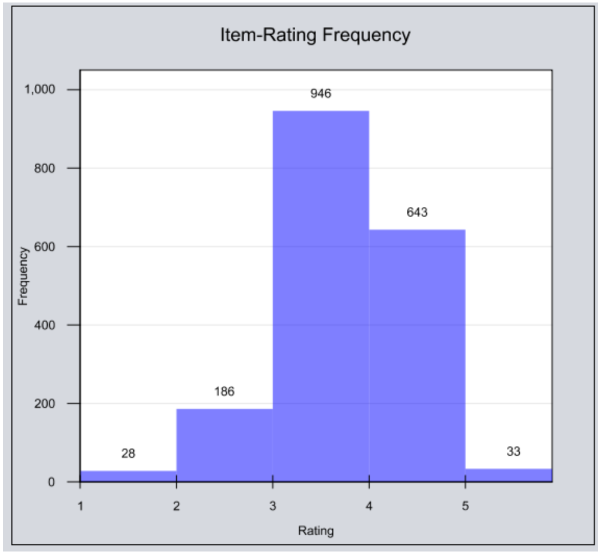

Figure 1: Item-Rating Frequency for Rating Dataset

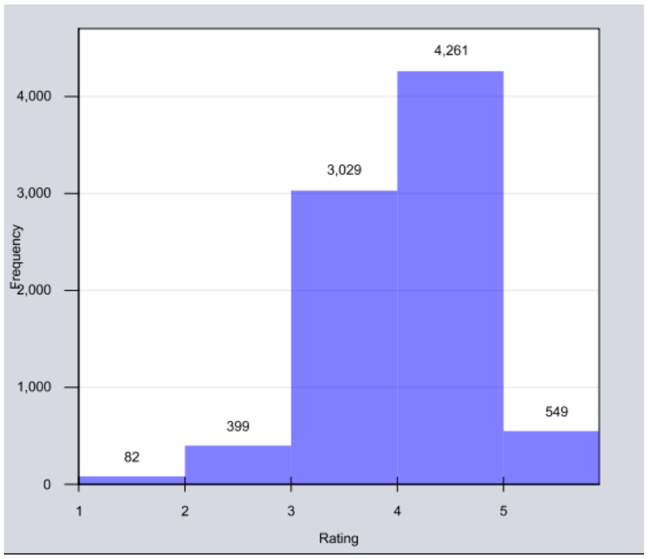

Figure 2: User-Rating frequency for Rating Dataset

Total Items	1836
Total Users	8320
Total Ratings	43945
Average rating for items	3.623
Standard Deviation for items	0.643
Average rating for users	3.911
Standard Deviation for users	0.643

Table 1: Summary of Rating Dataset

As we can see, with this dataset we can say that we have few ratings (comparing with other datasets) and we can say that items are medium rated (rounding the average, the average rating is 4 for user's perspective and items perspective). Looking into the graphics we can say that the ratings are nearby a normal distribution.

After that I proceed to the analysis for ranking dataset, this dataset was more than 15 times bigger than the rating dataset and this dataset can be summarized in.

Total Items	43968
Total Users	22531
Total Ratings	737386
Average rating for items	4.265
Standard Deviation for items	0.679
Average rating for users	4.509
Standard Deviation for users	0.338

Table 2: Summary of Ranking Dataset

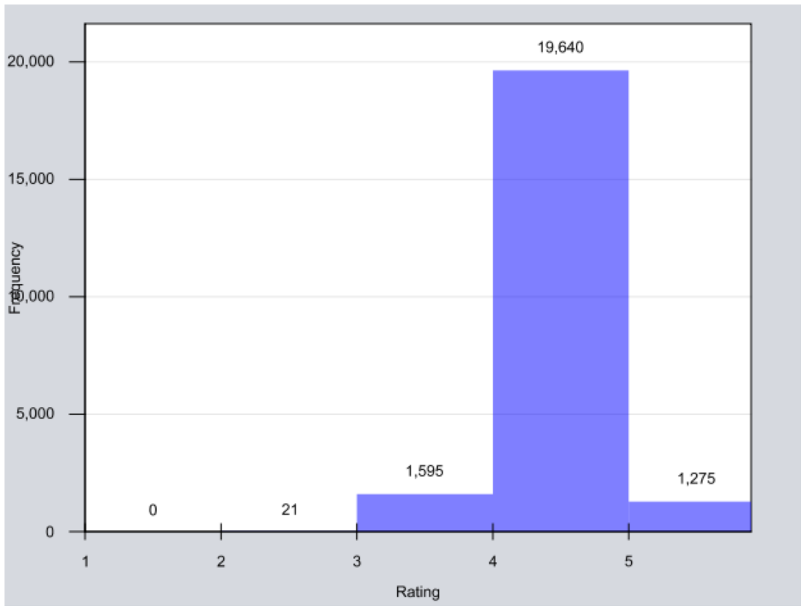

Figure 3: User-Rating frequency for Ranking dataset

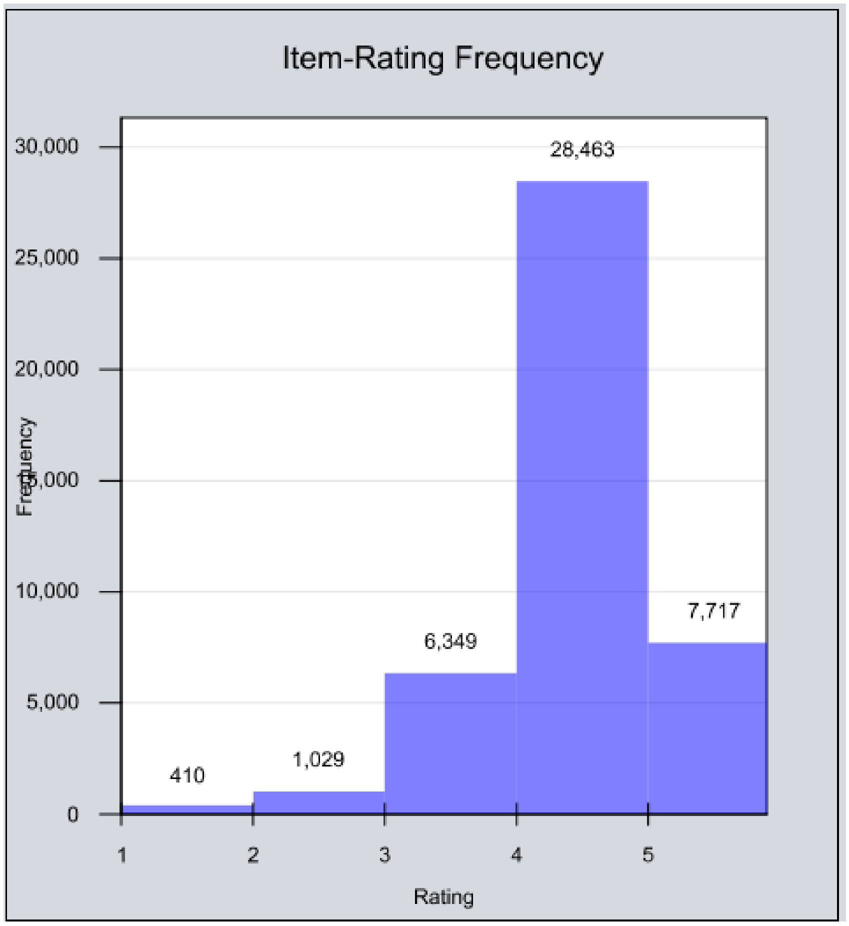

Figure 4:Item Rating frequency for Ranking

The ranking dataset has some interesting niches, first, the total items are bigger than users, contrary to the standard supposition that user's list size is bigger than the item's list size, so we can assume that if we use an Item-KNN algorithm it will use more memory than other algorithms. In this dataset also we can say that the ratings given to items, and the ratings given by the users are more averaged to 4-5, contrary to the Rating Dataset where the items are averaged to 3-4.

Framework and technology

For this work I have checked some reviews about recommender libraries, so based on Graham Jenson post on Github [2] I looked for the Open Source libraries written in Java, and I choose Lenskit because it was open source and the examples from the documentation [3] worked without problems. Also this library integrates well with automatic building systems (like Gradle and Maven). Also I used Java as my primary programming language for this task because I'm more experienced with this language than Python, R or .NET.

Finally, the built software uses the next technologies:

- Java 8 as Programming Language [4]
- Gradle as building system [5]
- Lenskit as recommender systems library
- Simple Logging Facade for Java as logging system [6]
- GRAL library for graphics [7]
- GSON library for JSON data management [8]

Rating

Tested Algorithms

For rating dataset, I tested the next Algorithms:

- User Based collaborative algorithm based on the Grouplens' work [9] and Shardanand's work [10]
- Item based collaborative algorithm based on Sarwar's work [11]
- Slope One Predictor based on Lemire's work [12]
- FunkSVD Matrix factorization as explained in the paper "Matrix Factorization Techniques for Recommender Systems" [13]

For the testing phase I used a sensitivity test similar to the test used on Sarwar's work, I used partitioned data (called the holdout parameter) and tested every algorithm listed, for user based and item based collaborative algorithms I have used Cossine, Pearson and Spearman correlation. For Slope one the test was realized using a normal Slope One algorithm and a Weighted Algorithm. For Matrix Factorization I tested the sensibility of the Latent factors parameter, testing the dataset with 10 to 50 latent factors, scaling 5 by 5. The learning rate for this algorithm was set with 0.001 (the default value for lenskit library), and 200 iterations. The baseline algorithm configured for all algorithms was a User Mean Item Scorer [14] that have a Item Mean scorer as the User baseline. The results from the different tests are visualized in Figure 5, Figure 6, Figure 7, Figure 8, Figure 9, Figure 10, Figure 11 and Figure 12.

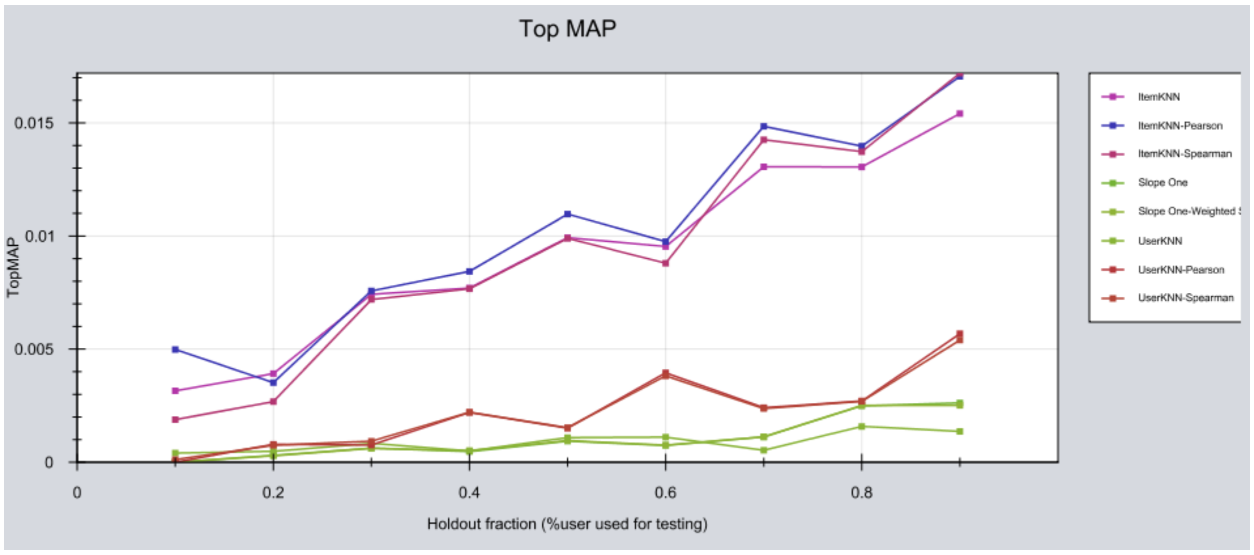

Figure 5:MAP Sensibility for Rating Dataset, Collaborative Algorithm

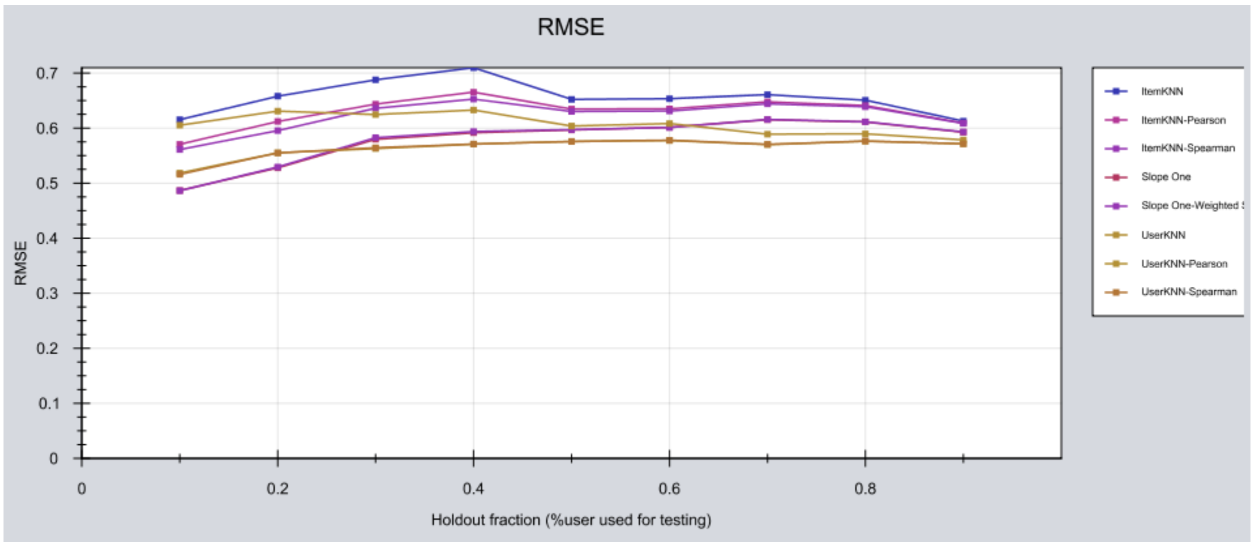

Figure 6:RMSE Sensibility for Rating Dataset, Collaborative Algorithm

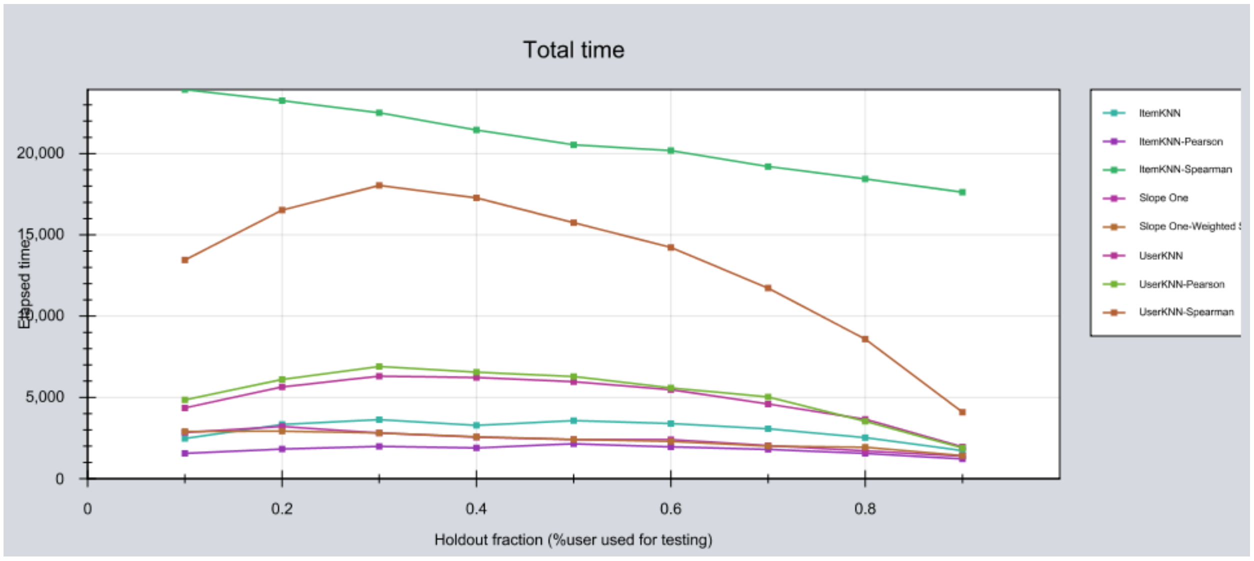

Figure 7:Test Total time (in seconds) elapsed for Rating Dataset, Collaborative Algorithm

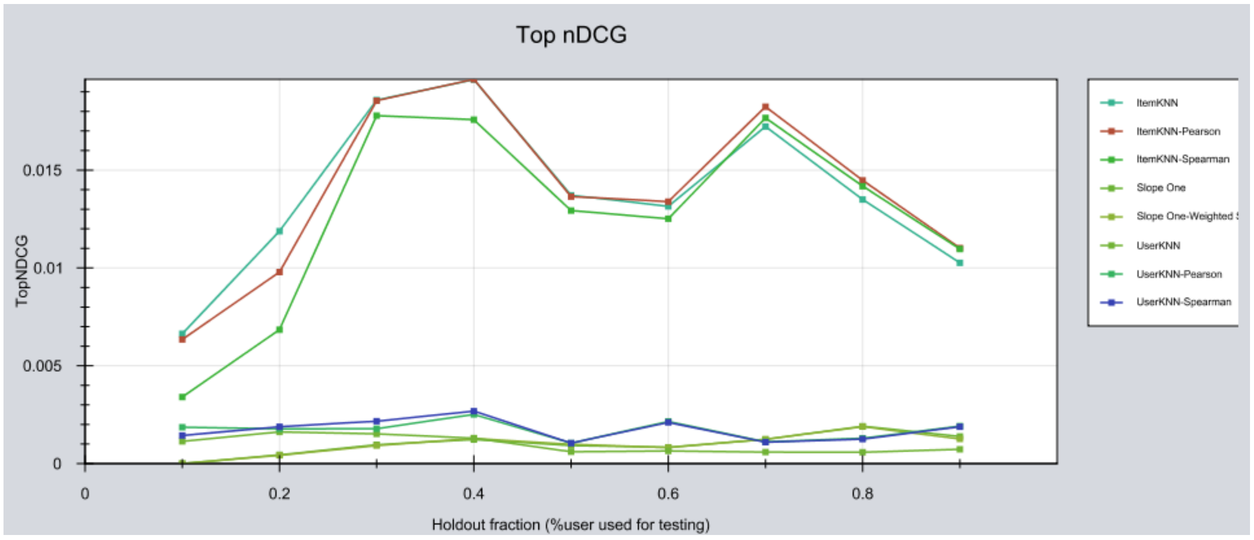

Figure 8:Top nDCG Sensibility for Rating Dataset, Collaborative Algorithm

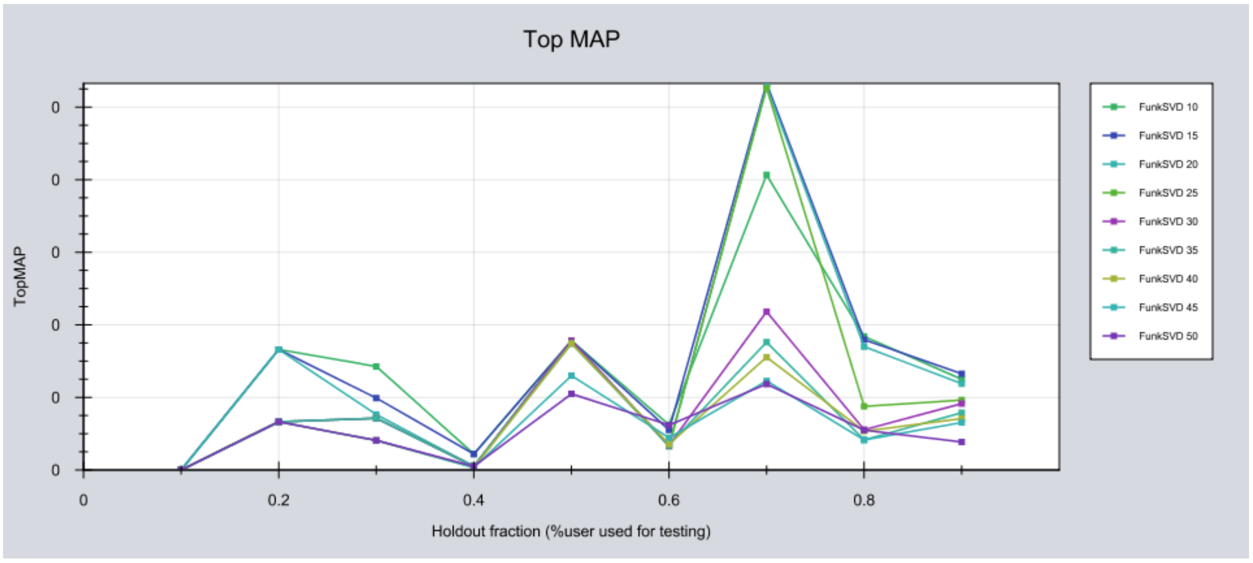

Figure 9:MAP Sensibility for Rating Dataset, SVD Algorithm

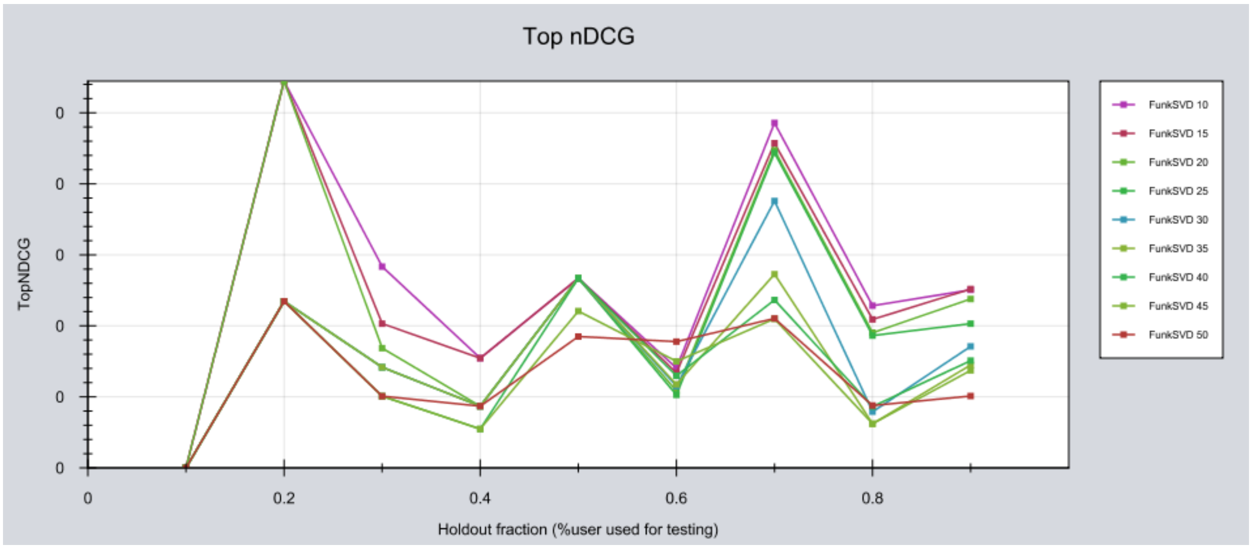

Figure 10:nDCG Sensibility for Rating Dataset, SVD Algorithm

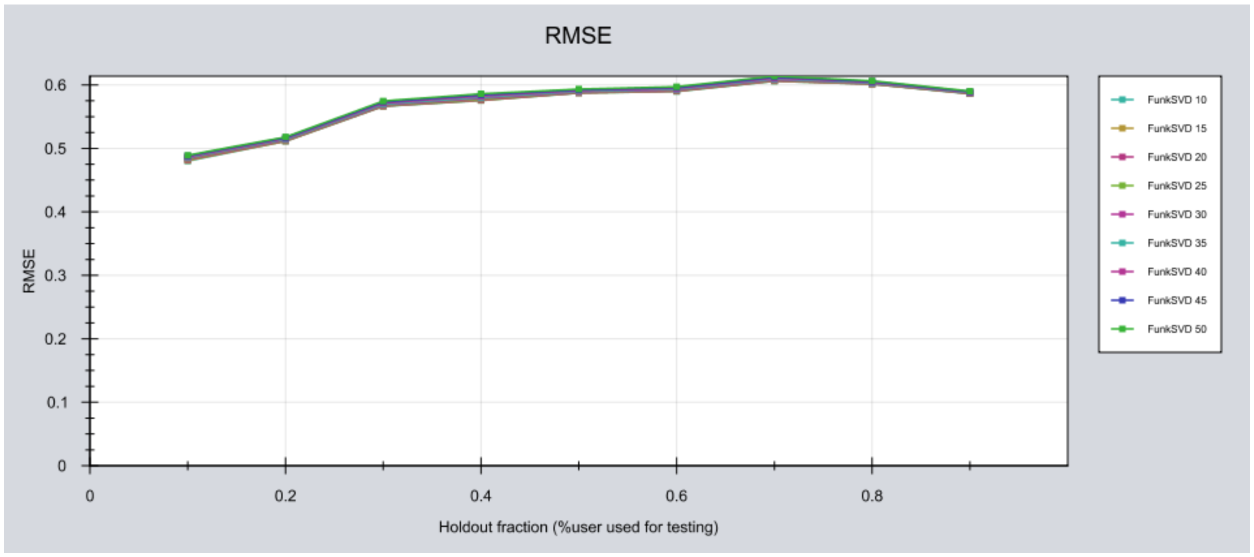

Figure 11:RMSE Sensibility for Rating Dataset, SVD Algorithm

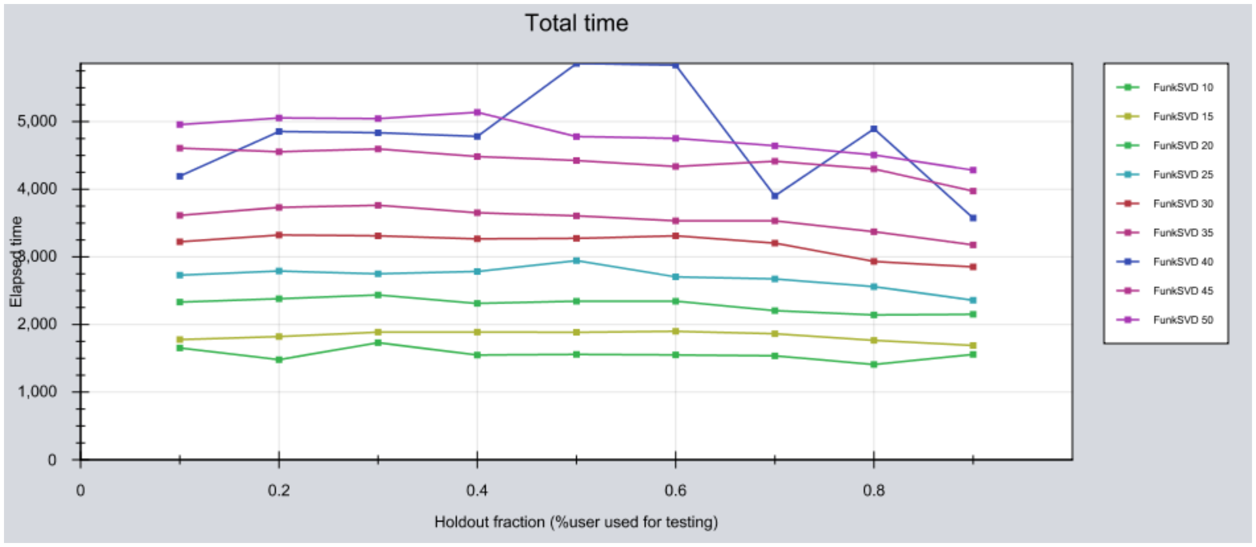

Figure 12:Total time elapsed (in seconds) for Rating Dataset, SVD Algorithm

As we can see with this tests the more stable algorithm for the RMSE is the Matrix Factorization, so I chose that algorithm to run the Rating predictions for the rating dataset, the software is built taking this in mind, so the user must choose what is the rating dataset and the output file, select the algorithm and wait for the output, the output is ready to submit to CodaLab.

In terms of memory usage all algorithms use between 1.2 to 1.5 Gb of RAM memory, the algorithms were tested on a computer with Windows 10 x64, Java 8u102, 16Gb RAM.

Ranking

The ranking dataset as we can see is too big, so I had some problems about time for the testing, at least I could test Matrix Factorization Algorithms, so I could test the FunkSVD Matrix factorization algorithm and a UserKNN with cosine correlation algorithm. For the testing purpose I have used the same logic as the rating testing, I partitioned the dataset and I have tested the sensibility to the holdout parameter, I have tested the sensibility of the latent factors parameter with the same conditions as the rating test. The results are shown in Figure 13, Figure 14, Figure 15 and Figure 16.

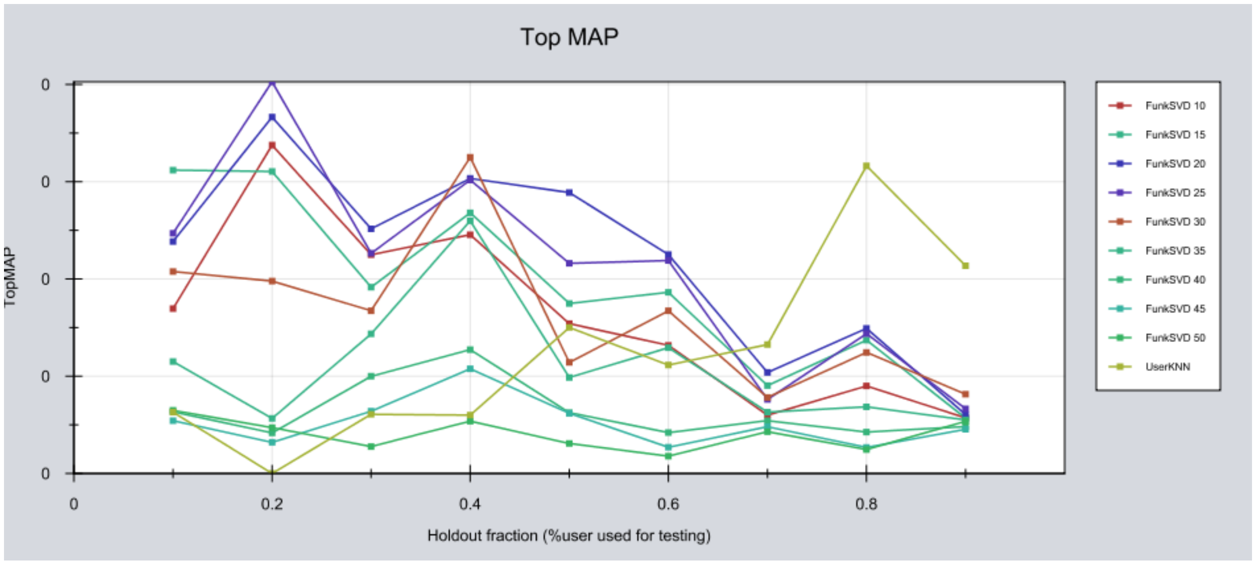

Figure 13:MAP Sensibility for Ranking Dataset, SVD Algorithm

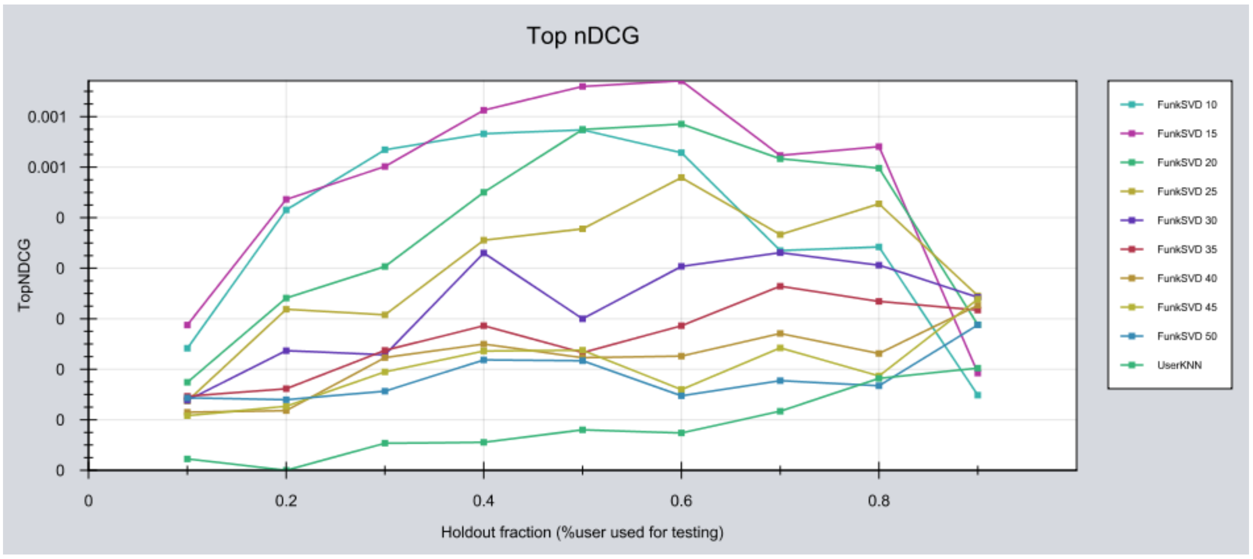

Figure 14:nDCG Sensibility for Ranking Dataset, SVD Algorithm

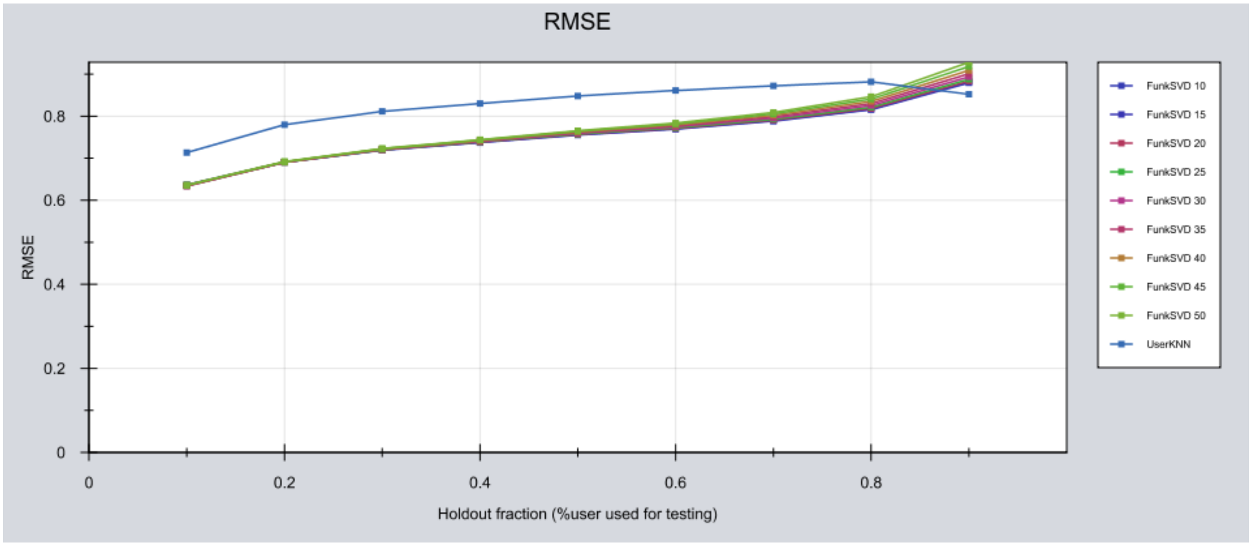

Figure 15:RMSE Sensibility for Ranking Dataset, SVD Algorithm

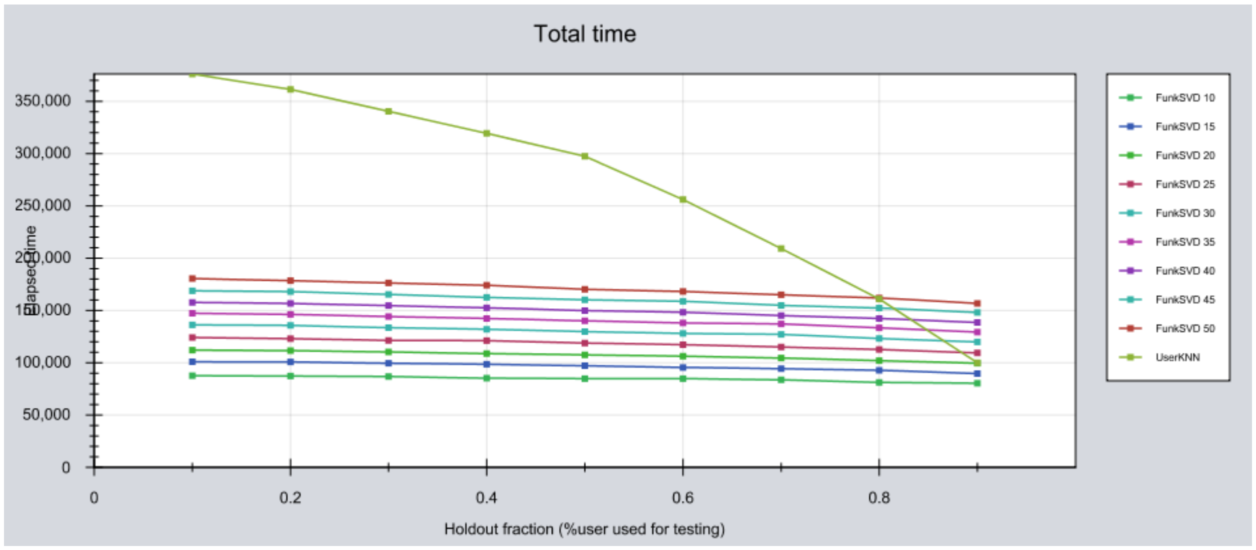

Figure 16:Total time Sensibility for Ranking Dataset, SVD Algorithm

As we can see the Matrix factorization is the best algorithm in terms of all the tested factors, as RMSE (lowest RMSE is SVD), MAP, nDCG (SVD has the highest MAP and nDCG) and in terms of time, UserKNN is too slow compared with SVD. In terms of memory the UserKNN algorithm uses approximately 4GB of RAM memory, and SVD uses 2.5 to 3Gb of RAM memory, so I used Matrix Factorization to give the top10 recommendations. The software similar to the rating predictors needs an input file with the users to test and an output file ready to upload to CodaLab.

Discussion and Conclusion

Concerning to these datasets I can say that collaborative algorithms are fine with a well populated dataset, but in this case a lot of users and items suffer of the cold start problem, checking the output that a UserKNN rating predictor. In the case of the ranking dataset the Item KNN couldn't be used because the RAM usage, this was caused because this dataset has more items than users, as opposed the first supposition on the ItemKNN algorithm.

The SVD algorithms in this case worked fine and they didn't have problems with the dataset, the only trouble with this algorithm was the case of training, and try to get the latents factor number. Once we get this info, the algorithm performance is ok for most of this use cases in this work.

References

[1] W. Foundation, "Recommender Systems," 2016. [Online]. Available: https://en.wikipedia.org/wiki/Recommender_system.

[2] G. Jenson, "List of Recommender Systems," Github, 21 02 2016. [Online]. Available: https://github.com/grahamjenson/list_of_recommender_systems.

[3] L. Contributors, "Lenskit Documentation," Lenskit, 2014. [Online]. Available: http://lenskit.org/documentation/.

[4] Oracle, "Java," Oracle, 2016. [Online]. Available: https://www.oracle.com/java/index.html.

[5] G. community, "Gradle User Guide," Gradle, 2016. [Online]. Available: https://docs.gradle.org/current/userguide/userguide.html.

[6] QOS.ch, "Simple Logging Facade for Java," QOS.ch, 2016. [Online]. Available: http://www.slf4j.org/docs.html.

[7] E. Seifert, "GRAL Java Graphics," 2016. [Online]. Available: https://github.com/eseifert/gral.

[8] Google, "GSON," Google, 2016. [Online]. Available: https://github.com/google/gson.

[9] P. a. I. N. a. S. M. a. B. P. a. R. J. Resnick, "GroupLens: An Open Architecture for Collaborative Filtering of Netnews," in *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, New York, NY, USA, ACM, 1994, pp. 175-186.

[10] U. a. M. P. Shardanand, "Social Information Filtering: Algorithms for Automating "Word of Mouth''," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, ACM Press/Addison-Wesley Publishing Co., 1995, pp. 210-217.

[11] B. a. K. G. a. K. J. a. R. J. Sarwar, "Item-based Collaborative Filtering Recommendation Algorithms," in *Proceedings of the 10th International Conference on World Wide Web*, New York, NY, USA, ACM, 2001, pp. 285-295.

[12] D. a. M. A. Lemire, "Slope One Predictors for Online Rating-Based Collaborative Filtering," in *Proceedings of the 2005 SIAM International Conference on Data Mining*, 2005, pp. 471-475.

[13] Y. a. B. R. a. V. C. Koren, "Matrix Factorization Techniques for Recommender Systems," *Computer,* vol. 42, no. 8, pp. 30-37, 2009.

[14] Lenskit, "UserMeanItemScorer," 2016. [Online]. Available: http://lenskit.org/apidocs/org/grouplens/lenskit/baseline/UserMeanItemScorer.html.