IMPROVING TOP-N EVALUATION OF RECOMMENDER SYSTEMS

by

Vaibhav Mahant

A thesis submitted to the Graduate College of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
August 2016

Committee Members:

Michael Ekstrand, Chair

Byron J. Gao

Vangelis Metsis

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

DEDICATION

Dedicated to my family and soon-to-be wife

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Dr. Michael Ekstrand for his patience and continuous support to me towards the completion of this work as well throughout my graduate study.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure **Page**

ABSTRACT

Recommender systems are used to provide the user with a list of recommended items to help user find new items they might prefer. One of the main task of the recommender is to provide such items that the user has not seen before. But while evaluating, if the recommender correctly predicts such items we penalize the recommender, usually because the relevance of the item for that user is unknown, and because of the unknown relevance the item being recommended was not present in the test set of the recommender. In recommender systems it is very hard to get the relevance of every item for every user. In this research we are trying to address this problem by randomly adding decoys into the recommender's test set. We will be measuring the performance of the recommender with different decoy sizes. We find that random decoys are exaggerating the advantage of popular-item recommenders, casting doubt on their usefulness.

I. INTRODUCTION

Recommender systems can now be found in many modern applications that provide the user to a huge collections of items. These systems are used to predict how much a user will like a particular item, or provide the user with a list of items he/she may like. These systems are very useful in helping the user find an item he may prefer which he has not seen before. These systems use several algorithms to generate the recommendation list for a particular user.

Problem Statement

Evaluation of the recommender systems is an active area of research, where there are a lot of open questions to answer. The common methodology of the evaluation is the use of precision oriented metrics which helps us analyze the quality of the items being recommended. While the traditional evaluation metrics are good, they also have some flaw. One of the main flaw is the violation of the assumption of *fully-coded corpus:* that for every item that is returned by the recommender or information retrieval system we know whether it is relevant. However, in recommender systems and most deployed Information Retrieval systems, we do not know the relevance of every item. We know some relevant items – the ones the user has already purchased on the test set – but we do not know whether other items are irrelevant, or relevant but unknown to the user. Most items are probably irrelevant, but if all unpurchased items are irrelevant, then the user would not need a recommender. The best recommendations for many applications will be exactly those items that are relevant, but that the user did not purchase because they did not know about them without the recommender. But a recommender that recommends such items will be judged

to be inaccurate, because it suggests items that is the not in the test set.

Current Work

In this research we are trying to address this problem, by adding random-item set while testing the recommender algorithm. These random item sets are called 'Random Decoy Set'. The algorithm being tested does not have any knowledge of the relevance of the items. These sets are constructed irrespective of the popularity of the item.

Our goal in this research is to make the evaluation not penalize the recommender for recommending an unseen item, (for which the relevance is unknown) by providing the recommender a random set of item from which it can pick an unseen good recommendation for the user (Koren, 2008).

A recommender can only recommend from the candidate set, or from the list of candidate items to consider for the recommendation. Usually the set of possible candidate items consist of everything except the item present in the train set. In our work we restrict the recommender's candidate set. Instead of putting all the item say universe of the item minus the train set, we ae putting the test set plus the randomly selected decoys, so that the recommender will have the item of test set plus the decoy set to recommend from.

This use of random decoys has been found in (Koren, 2008) but there are not many research that tell us how the decoys should be selected and what the size of the set should be, and how will the different decoys size could affect the recommender. Our goal here is to get some insight on how this decoys can be selected and the choice of the decoy set size.

Our main goal here is to measure the performance of the recommender accurately with the help of the random decoy set. We are trying to know what size of the random decoy set is good for the performance measurement of the recommender. So, we wanted

to get to size of the decoy set where we can say that this is a fair size where there is less chance that the metric will penalize the recommender for recommending good but unknown item, and we also want to balance that chance with other potential problems.

Overview of the Following Chapters

- A survey of the literature of the recommender systems evaluation as well as the information retrieval system evaluation. The goal here is to figure out the current practices being used in the context of information retrieval and recommender system evaluation.
- The next section we will be explaining the selection of the metrics used for evaluation, the data set to be used and the algorithms being used. Then we will be explaining the recommender experiment. How did we selected the random decoy set size.
- Then we present our results and at last, the conclusion and future work.

For this research we have used LensKit recommender toolkit (Michael D. Ekstrand, Michael Ludwig, Joseph A. Konstan, and John T. Riedl. 2011). LensKit is an open source toolkit for recommender systems research, which provides several evaluation technique that have historically been used for recommender systems research. It also supporting reproducible research in recommender systems. LensKit also supports reproducible research in recommender systems.

II. BACKGROUND AND RELATED WORK

Recommender Systems

Recommender systems are being used in various fields:

- **E-Commerce:** In e-commerce, recommender systems are being used to provide a list recommendation to the user that the user may like. This is the most common use of the recommender systems.
- **Social Network:** Recommender systems are now also being used in the social networking field. It is being used to recommend people to people. For example a social networking website suggest friends based on the history of the user. Some apps also suggest or recommend a person you may want to have a date with.
- **Entertainment:** There are also apps and website which recommend entertainment stuff like movies, music. In case of music there are so many apps that uses some features of recommender systems to recommend songs to their users. Book recommendation is also one of the main use of the recommender systems.

  Recommender systems also being used other fields like research papers or articles, news, restaurants etc.

Evaluation in Recommender Systems

A crucial component of recommender systems research and development is evaluation: determining whether the recommender is useful and effective. Offline evaluation using public data sets is a valuable tool for evaluating recommender system performance (Shani and Gunawardana, 2010). Offline evaluation is usually built on one or both of two different families of metrics derived from machine learning and information retrieval evaluation methodologies.

Prediction accuracy metrics compare the recommender's predicted rating to the user's actual rating. Top-N evaluation metrics compare the items recommended by the recommender to the user's ratings or purchases (Bellogin and Castells, 2010). Each of these is typically applied in a train-test methodology, where part of a data set of user preference data, such as a list of user ratings of movies, is hidden and the recommender is asked to reproduce the missing data. Cross-validation helps ensure statistical validity.

Evaluation is the recommender systems has been a prime area for research. Over the past few years researchers have proposed several methods to measure the accuracy, and to evaluate the performance of different algorithms used in recommender systems. Methods from information retrieval evaluation were also being used and discussed in recommender system domain (Barbieri et al., 2011; Breese et al., 1998; Cremonesi et al., 2010; Herlocker et al., 2004; Shani and Gunawardana, 2011). A precision-oriented evaluation is also addressed in (Bellogin, Castells, and Cantador, 2011) which compares three recommenders. A performance prediction approach has been introduced in (Bellogín, and Castells, 2010) which uses the clarity-based query based predictor to predict the neighborhood size in collaborative filtering.

Researchers like Bellogín, and other have primarily discussed the adaptation of information retrieval evaluation methodologies in the recommender system domain (Bellogin et al 2011). To get the knowledge of current best practices of evaluation we did a study of the information retrieval and recommender system evaluation literature. We took Bellogin's work as our base literature. In Bellogin's work we came across the NDPM metric which were not implemented in LensKit recommender toolkit.

The effect of popularity is talked in the recent research like (Cremonesi et al., 2011;

Cremonesi et al., 2010; Steck, 2011), where they have discussed the effect of popular items in the test set that results as a bias toward non-personalized recommenders for precision and recall. This has been also observed in (Bellogin et al 2012) as well, where they also proposed two alternative methods to overcome this bias. Our work is an extension to (Bellogin et al 2012).

III. METHODOLOGY

Metrics Used

While analyzing the literature we found that some precision oriented metrics like Mean Reciprocal Rank (MRR), Mean Average Precision (MAP) etc, can be used in a variety of ways. In this research we focused our attention on random decoy sets. We used these IR metrics with our configuration style. The metrics used in this research are the following:

Top-N Metrics:

- **Mean Reciprocal Rank (MRR) :** Prefers the ranking list in which the first good item occurs near the top of the result list (Baeza-Yates and Ribeiro-Neto, 2011).

  It is defined as:

  $$\boldsymbol{MRR = \sum_u 1/S_r(U)}$$

  Where $\mathbf{S_r(u)}$ returns the position of the first relevant item returned for user **u.**

- **Mean Average Precision:** Gives the summary of the user's ranking by averaging the precision obtained after each new relevant item is obtained (Baeza-Yates and Ribeiro-Neto, 2011).

  $$\boldsymbol{MAP = \frac{1}{u} \sum_u 1/Real_u \sum i \epsilon Real_u P@Rank(u,i)}$$

  Where (u,i) outputs the ranking position of item in the user's list. So, precision is computed at the position where each relevant item has been recommended.

- **Normalised Discounted Cumulative Gain (nDCG):** Normalised discounted cumulative gain (nDCG) uses graded relevance that is accumulated starting at the top of the ranking and may be reduced, or discounted, at lower ranks (Järvelin and

Kekäläinen, 2002):

$$\boldsymbol{nDCG = \frac{1}{u}\sum_{u}\left(\frac{1}{IDCG_u^{pu}}\right)\sum_{p=1}^{ps} fdis(rel(u, i_p), p)}$$

where the discount function is usually defined as, **dis** depending on the emphasis required on retrieving highly relevant items (Croft et al., 2009).

- **Precision and Recall :** Precision is defined as the fraction of recommended item that are relevant whereas Recall fraction of relevant item that has been recommended (Bellogin et al 2011). They are defined as follows (Baeza-Yates and Ribeiro-Neto, 2011):

$$\boldsymbol{p@k = \frac{1}{|u|}\sum_{u\epsilon U}\frac{|Rel_u@k|}{k}}$$

And,

$$\boldsymbol{p@k = \frac{1}{|u|}\sum_{u\epsilon U}\frac{|Rel_u@k|}{|Rel_u|}}$$

where Rel_u represents the set of relevant items for user , and Rel_u @k is the number of relevant recommended items up to position k.

These metrics listed above are all TopN metrics. These metrics are used over the recommended list, since over main goal here is to implement new ways to use the TopN evaluation metrics.

While configuring the TopN metrics we need to provide few things:

- List Size: Length of the recommendation list.

- Candidate Items: The list of items to be considered for recommendation.

The list size in this experiment were first set as 10, which we changed to 25 after some initial run of the experiment.

With these TopN metrics we have also used two Prediction Accuracy metrics.

Prediction Accuracy Metrics:

- **Root Mean Squared Error:** This metrics measure the difference between the score/rating predicted by a recommender and the actual rating given by the user.
- **Predict.nDCG:** Measures the rank accuracy of the recommender's rating with normalized discounted cumulative gain.

The configuration of these Predict metrics is different than that of TopN metrics. Unlike the TopN metrics we do not have to provide the list size and the candidate items.

The Data Set Used

- ML-Latest-small: A MovieLens Dataset, is the dataset we used for the initial run of our experiment. This dataset is consist of 5-star rating and free-text tagging activity. It contains 105339 ratings and 6138 tag applications across 10329 movies.
- ML-20M: Also a MovieLens Dataset, It contains 20000263 ratings and 465564 tag applications across 27278 movies (Harper and Konstan, 2015).
- Book-Crossing: It contains 278,858 users providing 1,149,780 ratings (explicit / implicit) about 271,379 books (Ziegler et al., 2005)

Algorithms

In this experiment we have used some popular recommender system algorithms, and they are:

- **User-User Collaborative Filtering:** This is the process of finding recommendation for a user based on the similarity of that user to other users. This is built on the principle that a particular user's rating records are not equally useful to all other users as input for providing personal item suggestions (Herlockeret al., 2002). For example if user A and B have similar taste in an item M it is more likely that they will have the same taste for item N.

- **Item-Item Collaborative Filtering:** Instead of user similarity, Item-Item Collaborative Filtering focuses on getting the item similarity. It's based on the patterns of similarity between the items themselves. In general, item-based recommenders look at each item on the target user's list of rated items, and find other items that seem to be similar to that item (Shardanand and Maes, 1995; Sarwar et al., 2001).
- **Funk SVD:** This is another version of collaborative filtering based on matrix factorization. This algorithm uses gradient descent for an SVD-style matrix factorization.
- **Popular Item Recommender:** This algorithm rank items based on their popularity. This recommender scores item between 0 to 1, 1 being most popular item and 0 being an unknown item.

- **Popularity Blending Item-Item:** This is a hybrid-type recommender that uses weighted score of the ranks provided by two recommender algorithms. (Ekstrand et al, 2015). This recommender blends the item-item rank with the popularity rank by asking both algorithm to provide a rank for an item and then combine them by weighted score of each rank score.
- **Popularity-Blending User-User:** This is also a hybrid-type recommender that blends the ranks given by two recommenders, but in the context for User-User Collaborative Filtering (Ekstrand et al, 2015).
- **Popularity Blending Funk SVD:** A hybrid-type recommender that blends the ranks given by two recommenders, in the context for Funk SVD (Ekstrand et al, 2015).
- **Random Item Recommender:** Recommends items randomly irrespective of their popularity or similarity.

Recommender Experiment

A typical LensKit evaluation experiment is composed of a collection of tasks.These tasks are depended on each other, for example the 'evaluate' task which does the train-test evaluation is depended on the 'crossfold' task which does the crossfolding of the data set. The three main task of this evaluation experiment are:

- **Fetch-data:** This task fetches the data set which is going to be used for the evaluation.
- **Crossfold:** The crossfold task does the partition of the data set received by the fetch data, for its cross validation. This generates separate train-test data sets for each fold. The method we used in this experiment for partition is the 'Holdout', which

holds a selected number of ratings for each user. These ratings to hold were selected randomly.

For this experiment we have set the total number of partition as 5. For efficiency we used a crossfold method that splits the users into some disjoint samples, instead of partitioning the entire user base. The sample size used here is 5000.

- **Evaluate:** This is the main task of the evaluation experiment. This task builds the algorithms, test them over the metrics provided and gives us the results in a csv file. This is the task where we have to explicitly tell the evaluator, which algorithms and metrics are going to be used in the experiment.

Algorithm Configuration

- **Item-Item Collaborative Filtering :** For Item-Item the ModelSize was set to 5000, the Neighborhood size was set to 20 and, the userVectorNormalizer was bind to BaselineSubtractingserVectorNormalizer
- **User-User Collaborative Filtering:** In User-User configuration the Neighborhood size was set to 30, the MeanDamping was set to 25 and the NeighborFinder was bind to SnapshotNeighborFinder.
- **Funk SVD :** For Funk SVD the IterationCount was set to 125 and the FeatureCount to 25.
- **Popular Item Recommender:** For Popular Item Recommender's configuration the ItemScorer is bound to PopularityRankItemScorer.
- **Popularity Blending Item-Item:** It is the same configuration as item-item with some more parameters. Like the item-item configuration the ModelSize is set to

5000, the Neighborhood size is set as 20 and, the userVectorNormalizer was bind to BaselineSubtractingserVectorNormalizer. Plus the ItemScorer is boud to RankBlendingItemScorer and the BlendWeight is set as 0.5. The BlendWeight is computed by mixing the scores of the two recommenders.

- **Popularity-Blending User-User:** For this the ITemScorer is bind to RankBlendingItemScorer, the BlendWidth is set as 0.5, the Neighborhood size was set to 30, and the MeanDamping was set to 25.
- **Popularity Blending Funk SVD:** For Popularity Blending Funk SVD the IterationCount was set to 125 and the FeatureCount to 25, the BlendWidth is set as 05, and ITemScorer is bind to RankBlendingItemScorer.
- **Random Item Recommender:** For Random Item Recommender configuration the ItemScorer is bind to RandomItemRecommender.

Experiment Runs

We ran the experiment 4 times with the different data sets described above.

- **Initial Run:** We initially ran the experiment over the small data set (ML-Lettest-Small). Our goal here was to make sure that the experiment is setup properly. In this run we included the Item-Item and Funk SVD algorithm with MRR, MAP, TopN.nDCG and Precision and Recall metrics.

  The list size was set to 10 and the candidate items were given with 5, 10, 20, 50, 100, 250, 500, and 1000 random decoys.
- **Second Run:** In the second run we include the Popularity-Item-recommender. This time we ran the experiment with the ML-20M data set. The goal here was to check whether the popularity of the item effects the evaluation or not. The metrics used

were the same as previous run. Here we added one more decoy set that uses all the items as decoys.

- **Third Run:** This time we ran the experiment separately with ML-20M and BookCrossing data set. With ML-20M we added Popularity-Blending-Item-Item, Popularity-Blending-FunkSVD and Pupular-Item-Recommender to get more insight of the popularity effect.

  With BookCrossing data set we included User-User and Popularity-Blending-User-User but we did not include both versions if Item-Item recommender. Here we also added the two prediction accuracy metrics. The decoy size was the same as previous.

- **Fourth Run:** In the fourth run we added more decoy sets. We used 10 and 25 percent of both the data sets. For ML-20M we added 1500, 2700 (10%), 3000, 5000, 67500 (25%) items as decoys.

  For BoockCrossing data set we added 10000, 20000, 270000 (10%), 50000, 660000 (25%). In this run with BookCrossing we didn't include the RMSE and Predict.nDCG metrics.

IV. RESULTS

Overview

After the initial run of the experiment we saw that the performance of the recommender tends to decrease as we add more and more decoys (Fig 1). This trend was observed for all the other metrics also. Here we will be discussing the MRR, MAP and nDCG. By observing the graph of the metrics we can say that there is no clear insight regarding the good decoy set size.

After this, in the next run of the experiment we added the popular item recommender. In the results of this run we observed that the metrics are showing a strong bias for the popular item recommender as compared to the other two. By this bias we mean that the values obtain for the popular item recommender for different decoy set size are higher than other recommender like Item-Item and FunkSVD (Fig 14), specially when the decoy set size are relatively small say 5 to 20 the difference between the values are high. After seeing this, we decided to add more popularity based recommenders to check weather this trend is common to all popularity based recommenders or not.

After adding more popularity based recommender we saw that the values obtain for other popularity based recommenders are also high as compared to the non-popularity recommenders. There is a clear advantage given to the popularity based recommender. Also we saw that there is no clarity regarding the good decoy set size. We also saw that there is no sweet spot for popularity and non-popular recommender. To get more insight of this, we ran the same experiment with the different data set.

After observing this bias we decided to test the recommender by providing 25% of the items in data set as random decoys. Here also we notice that while the decoy sets are

small the bias towards the popularity of the item is strong. Even though after adding all the items as decoys the gap between the popularity blended recommender and the others can be seen clearly. This proves that the popularity of the items being recommended does affect the evaluation of the recommenders. This bias towards the popularity is also the result of skewness in rating distribution. (Bllogin et al, 2010).

Our goal here was to figure out a good decoy set that prevents the recommender from being penalized for recommending an unseen item for which the relevance is unknown. We have seen in our experiment results that the method of using random decoys is giving advantage to the popularity recommender. It is hard to tell that what is a good decoy size. Also, the advantage being given to the popular item recommenders creates doubts on the usefulness of the random decoys.

Following are the chats created with results of different experiment run:

Here, the axis represents the following:

- **X Axis:** The decoy set,
- **Y Axis:** The value of the metrics evaluation, (This will always between 0-1)
- **Colored Series**: Represents different algorithms.

First Run

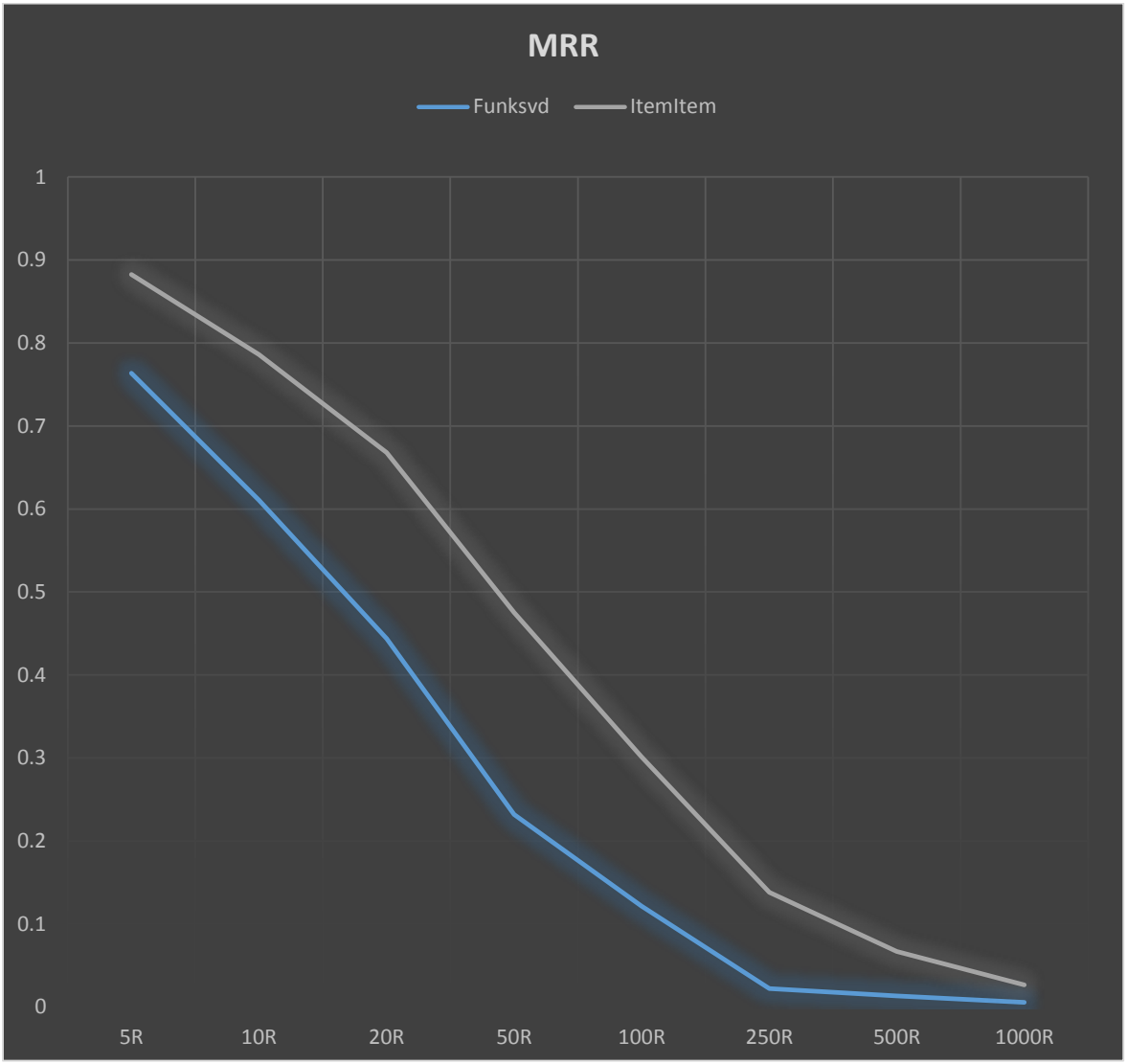

Fig 1. Performance of the recommender with MRR

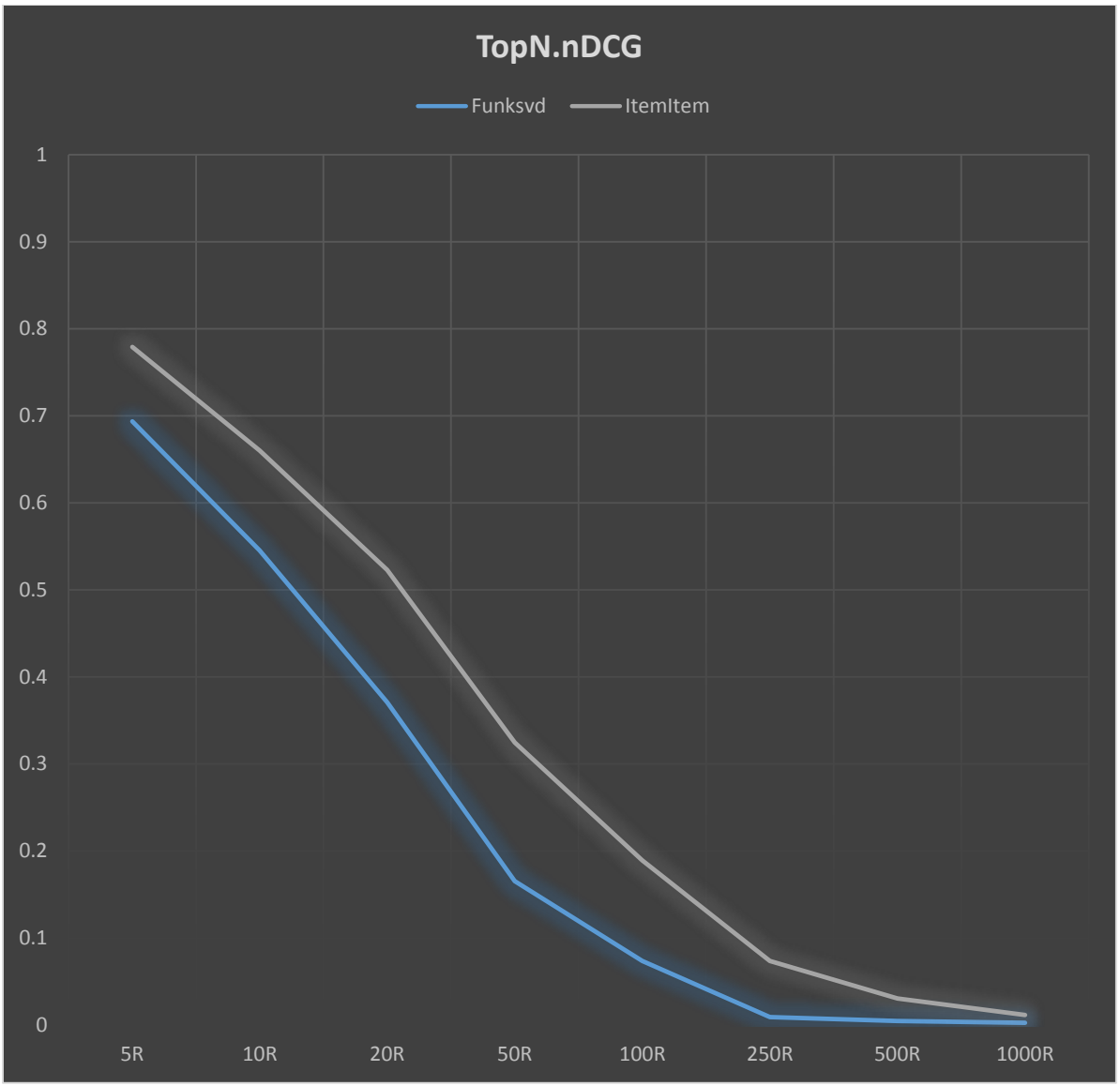

Fig 2. Performance of the recommender with nDCG

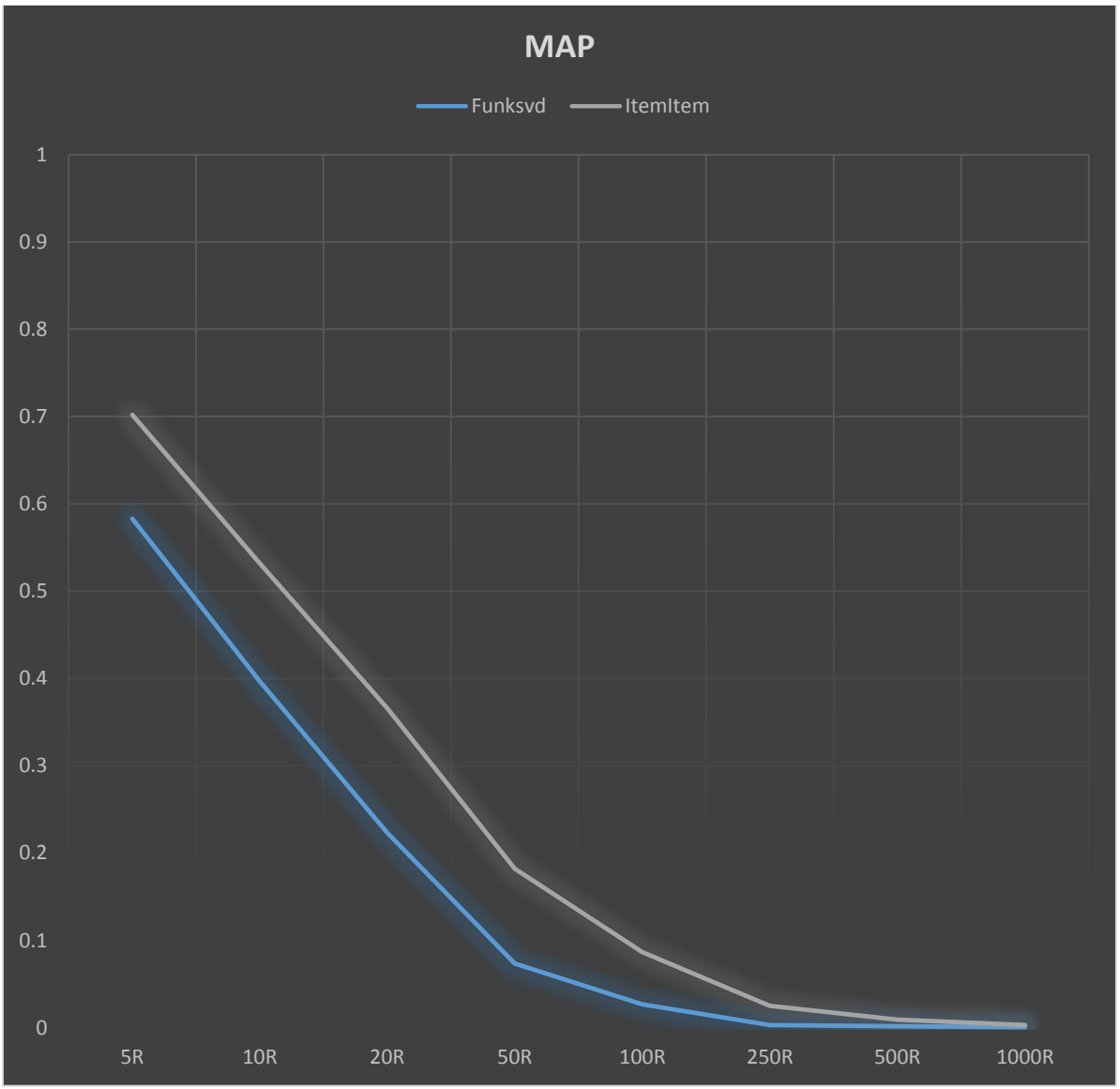

Fig 3. Performance of the recommender with MAP

Second Run

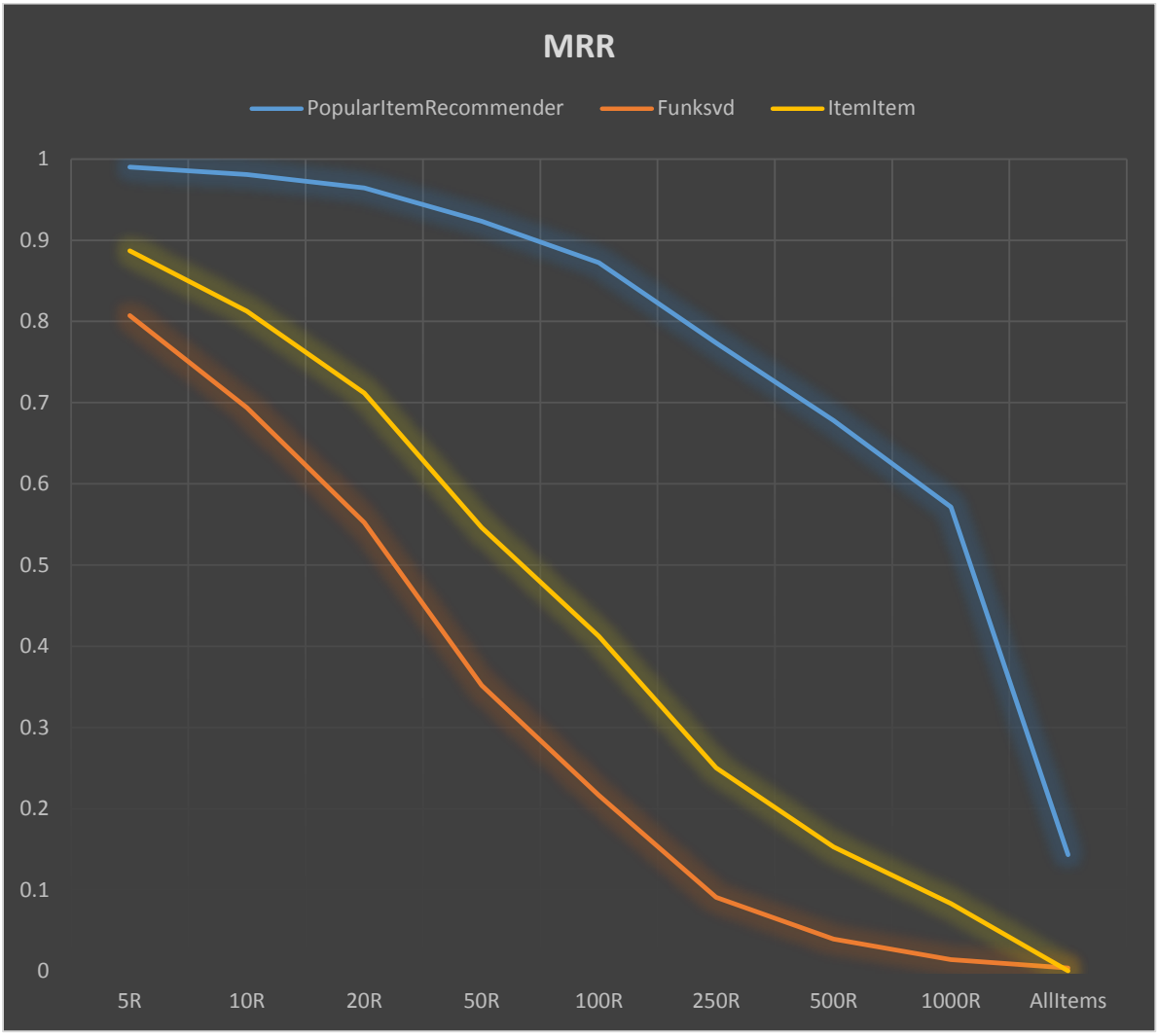

Fig 4. Performance of the recommender with MRR

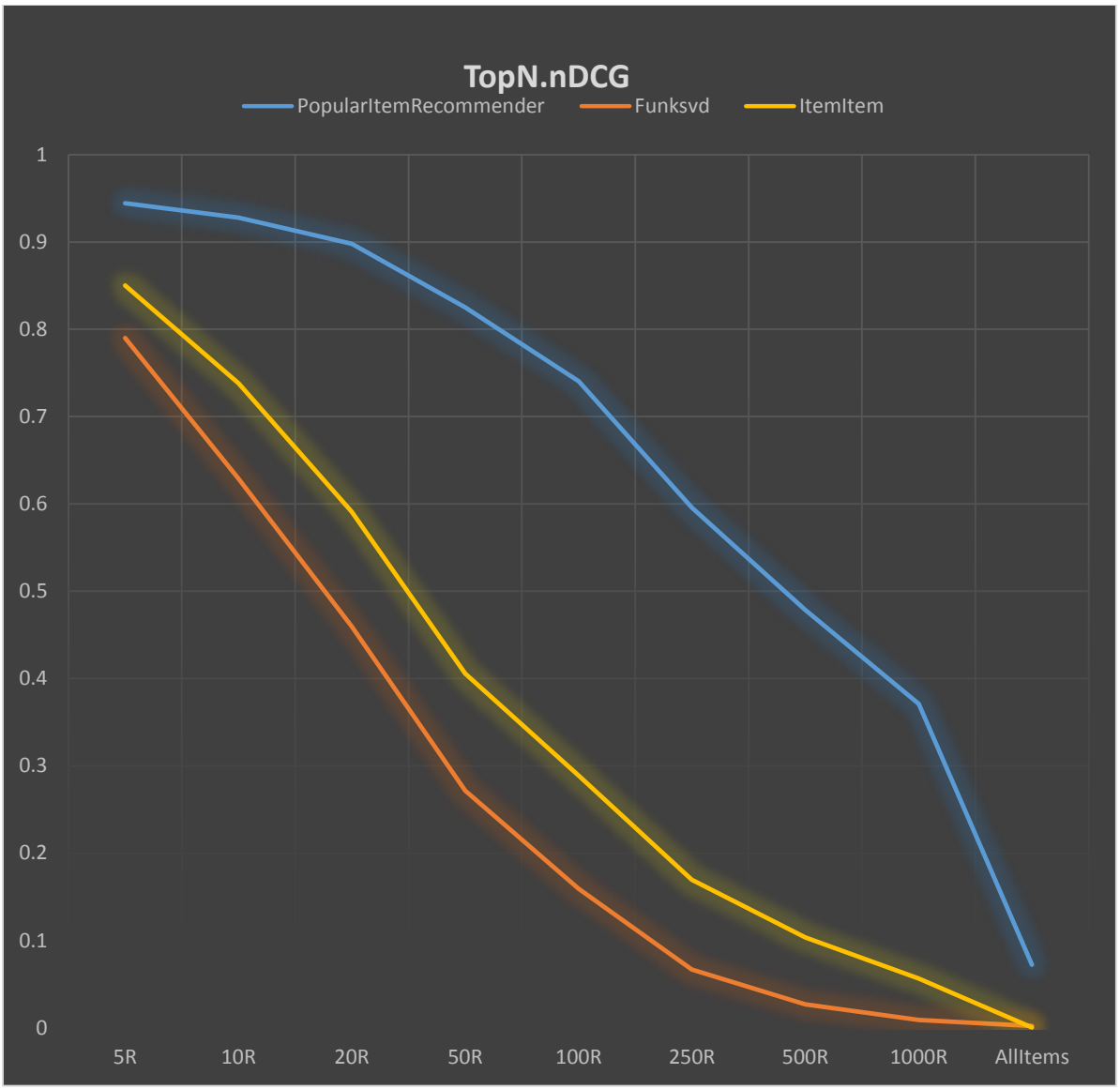

Fig 5. Performance of the recommender with nDCG

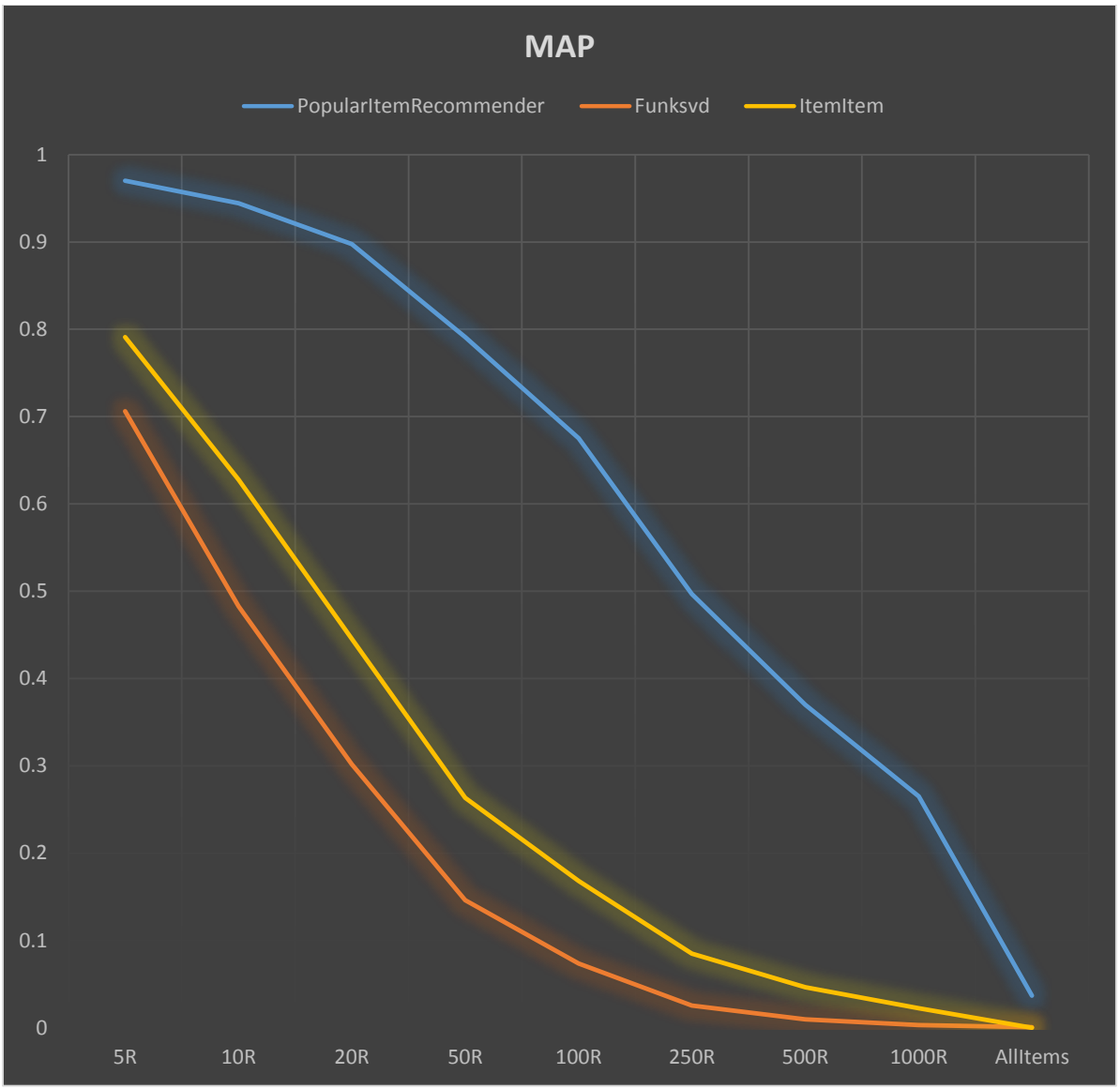

Fig 6. Performance of the recommender with MAP

Third Run (With ML-20M)

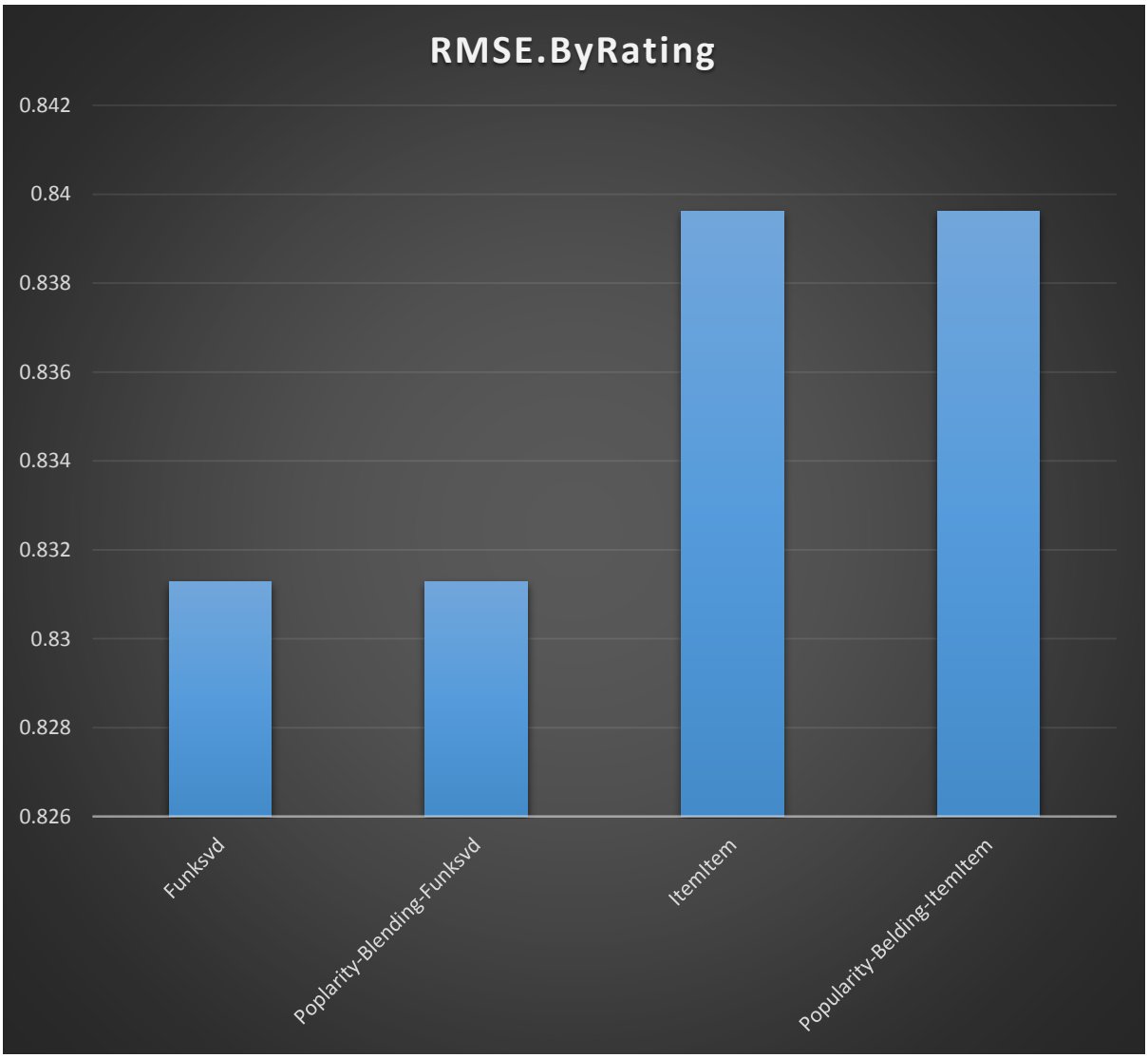

Fig 7. Performance of the recommender with RMSE

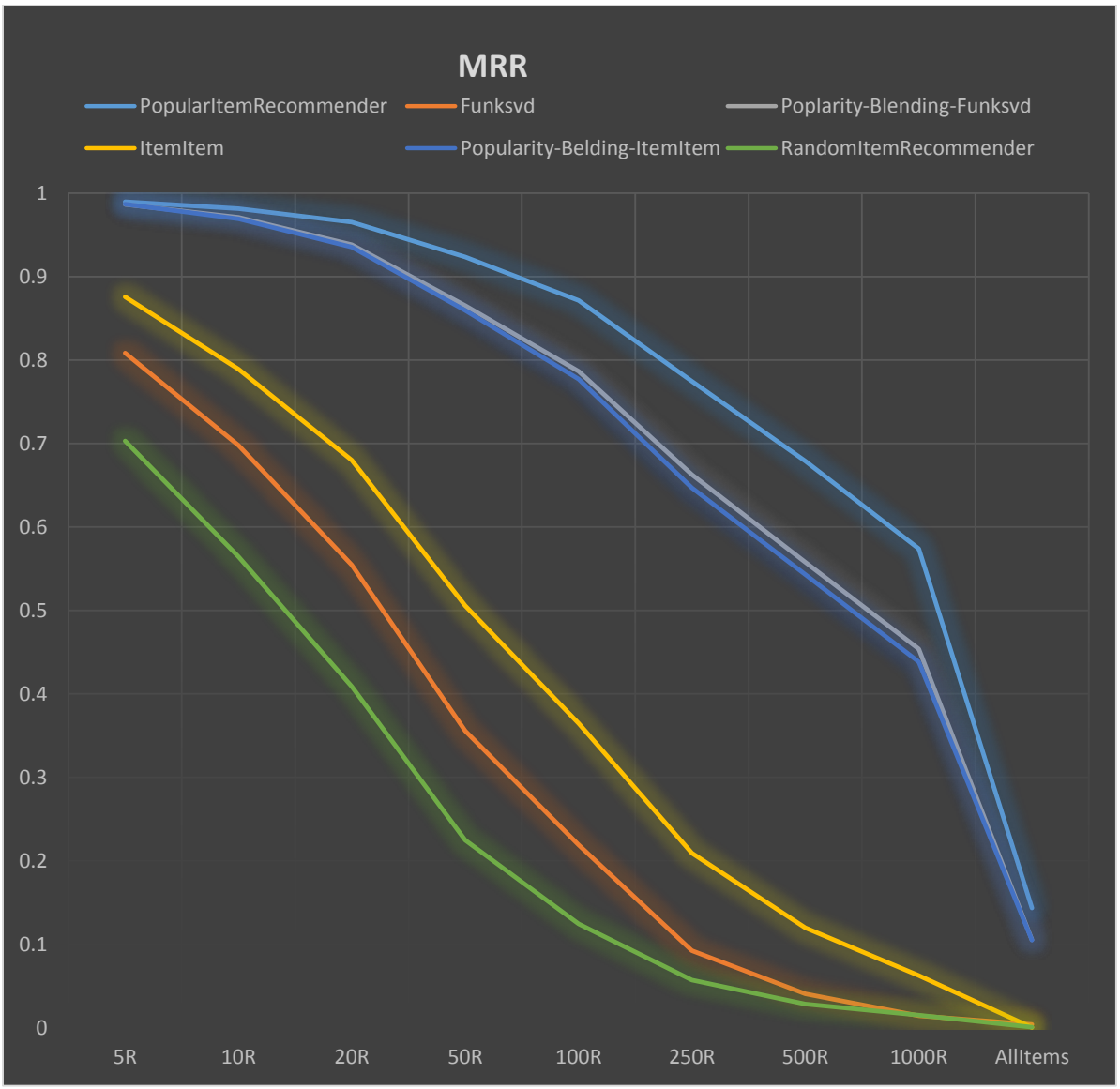

Fig 8. Performance of the recommender with MRR

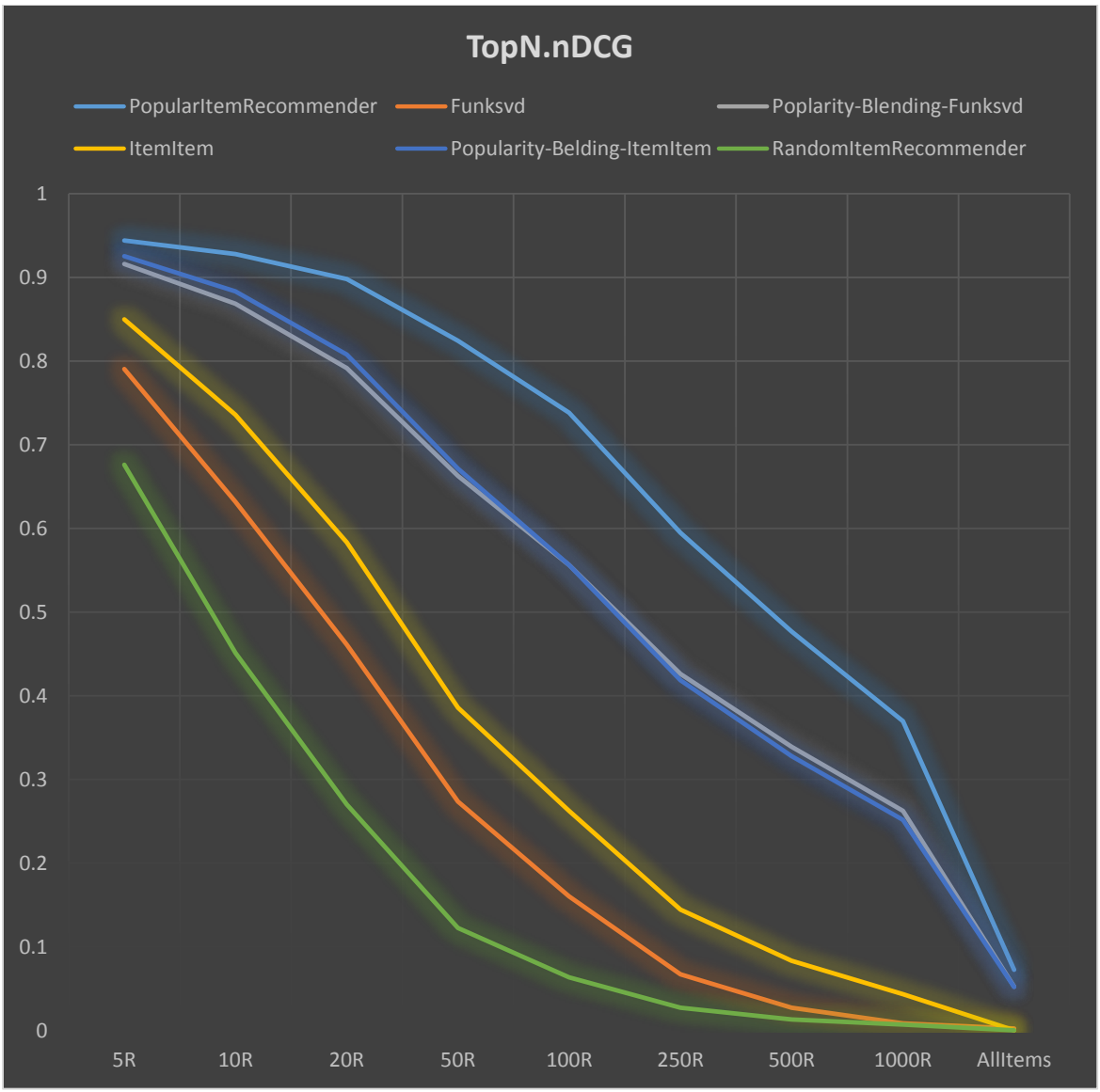

Fig 9. Performance of the recommender with nDCG

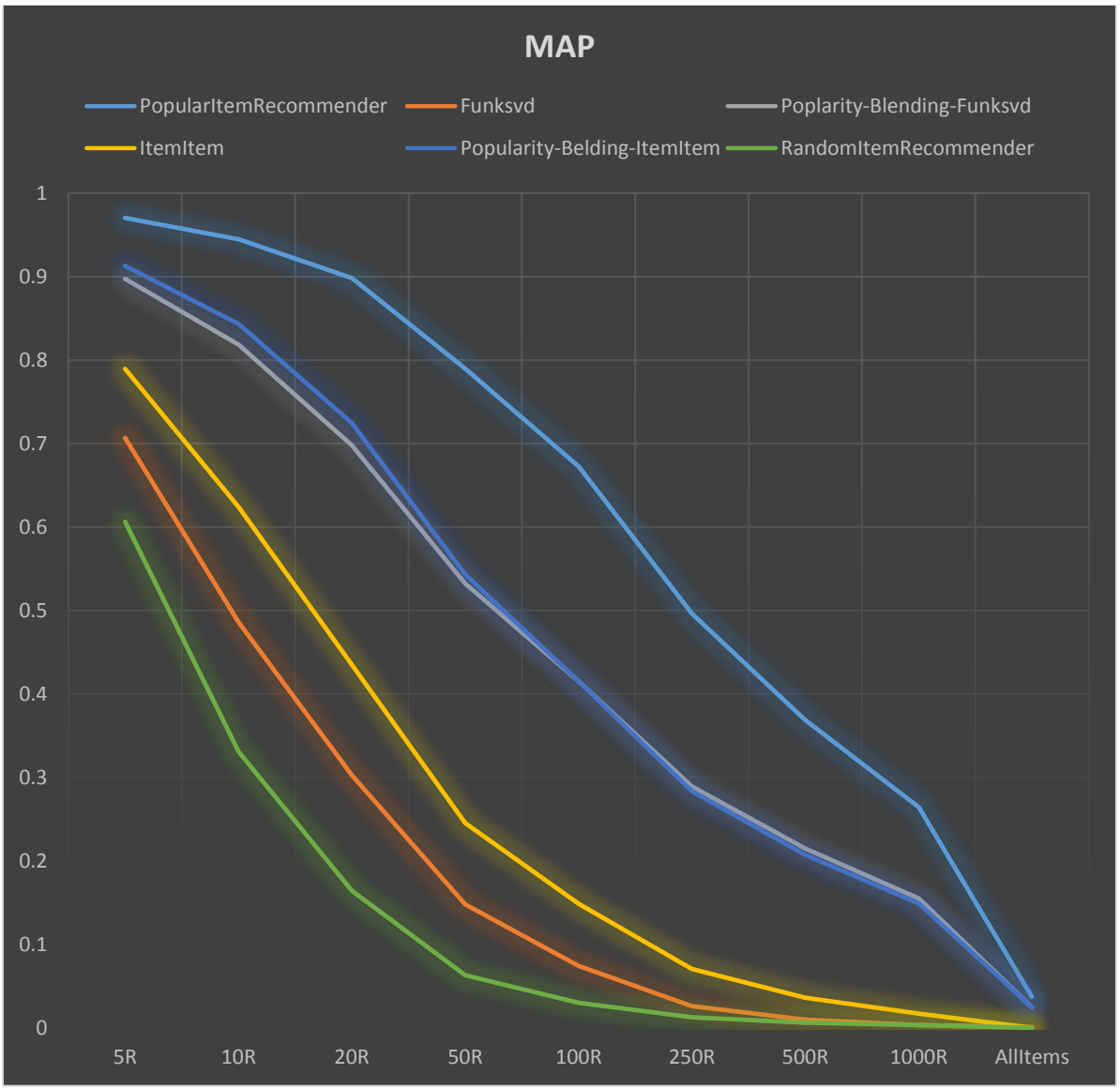

Fig 10. Performance of the recommender with MAP

Third Run (With BookCrossing)

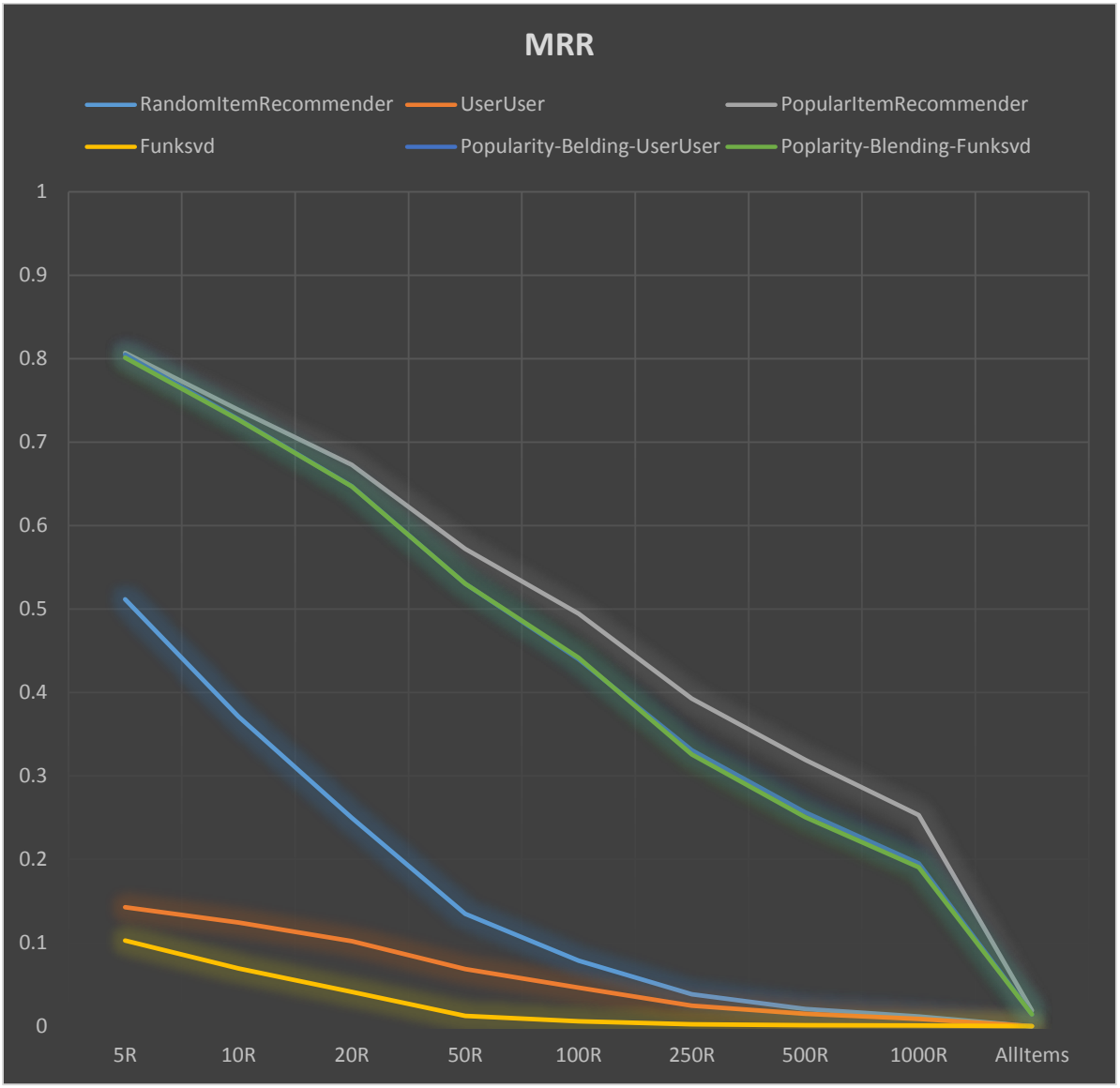

Fig 11. Performance of the recommender with MRR

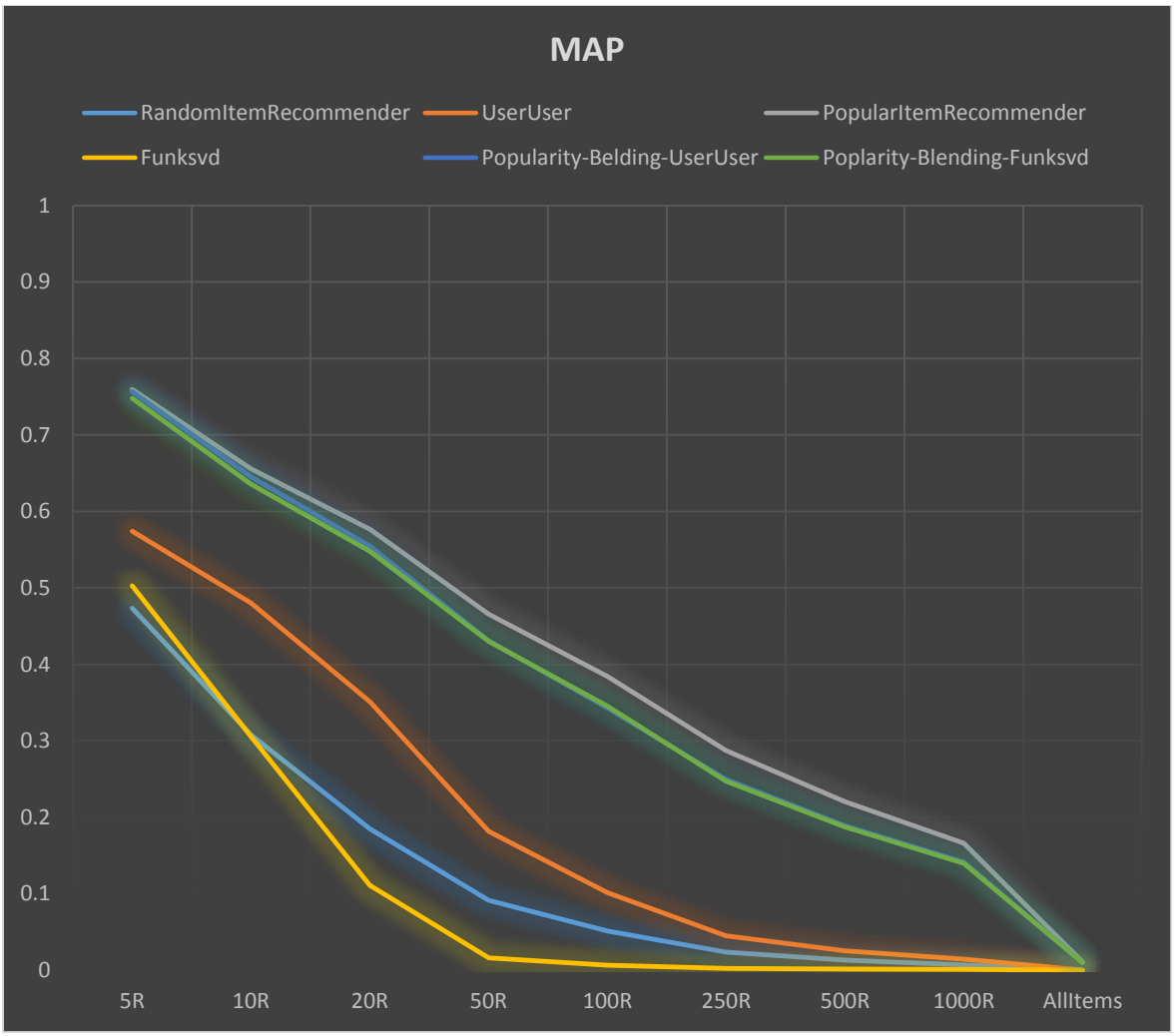

Fig 12. Performance of the recommender with MAP

Fourth Run (With ML-20M)

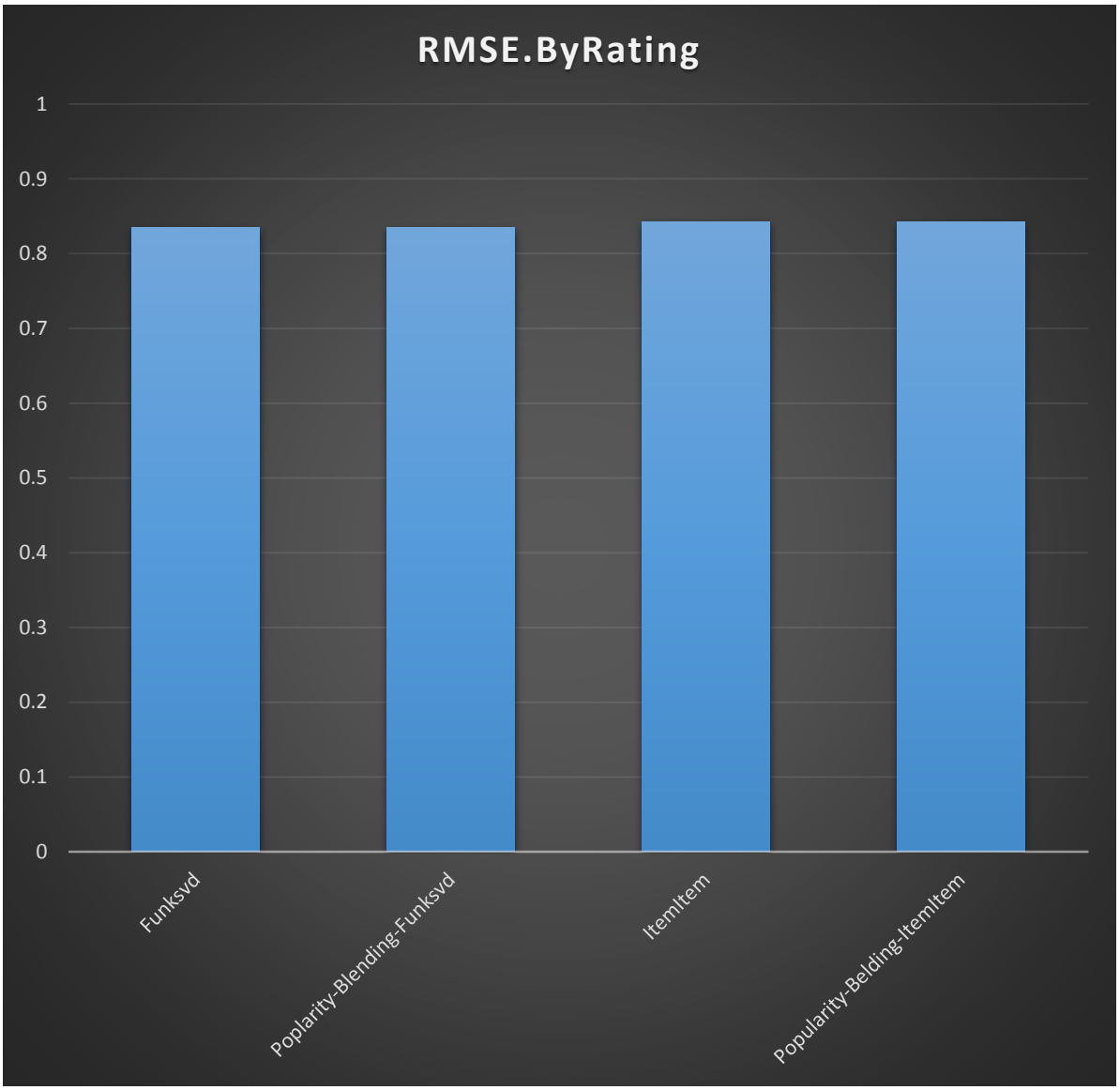

Fig 13. Performance of the recommender with RMSE

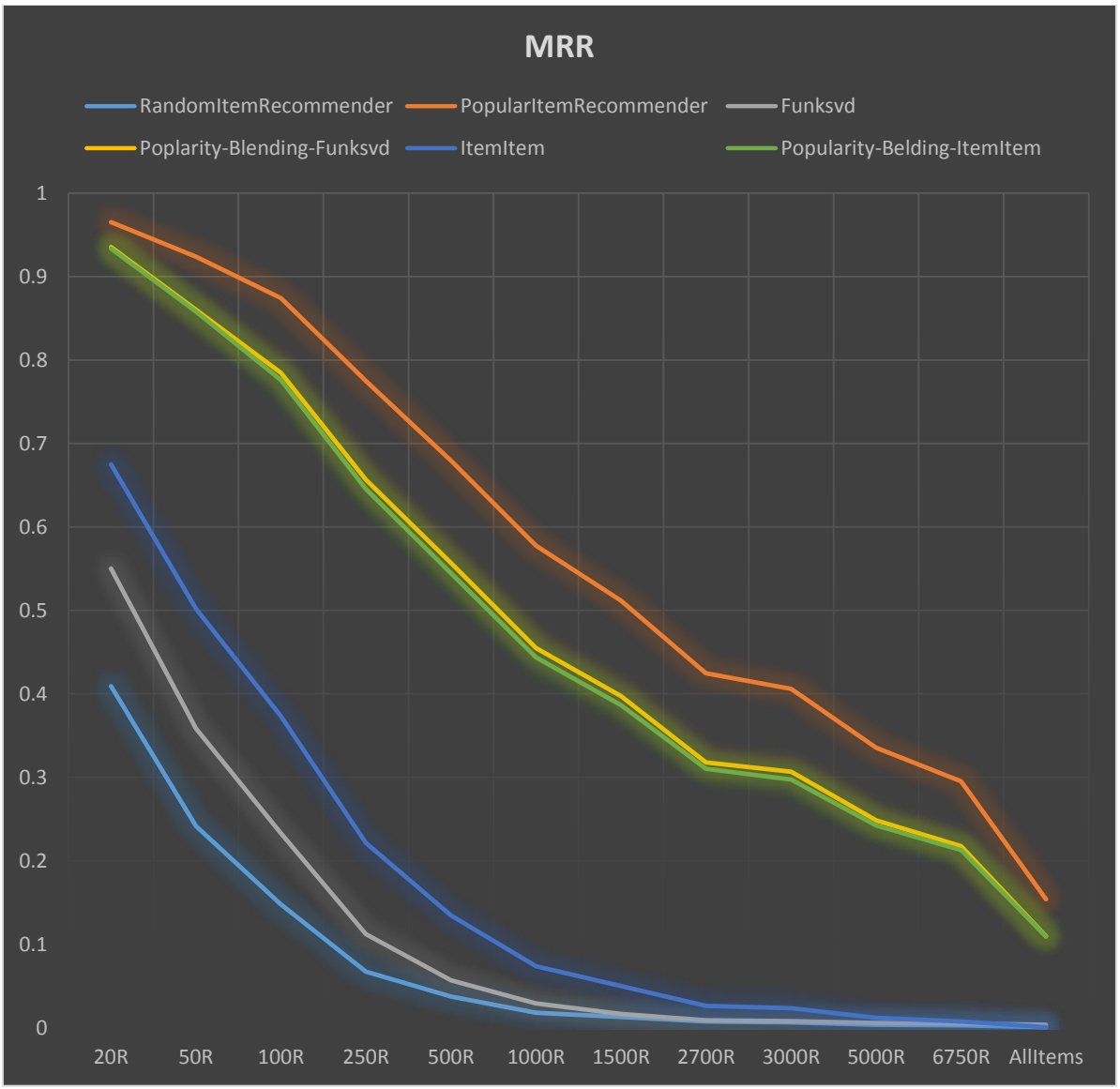

Fig 14. Performance of the recommender with MRR

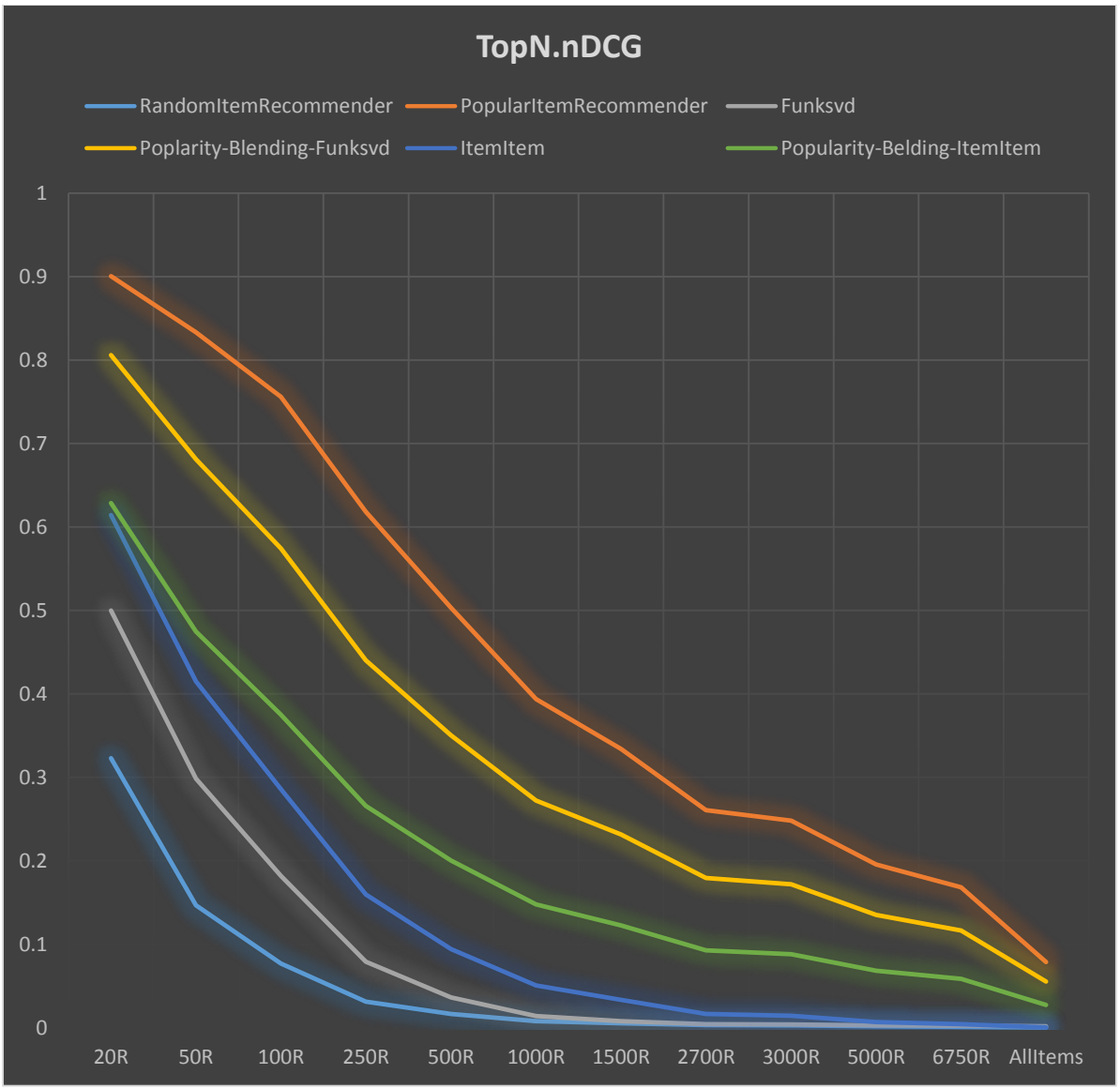

Fig 15. Performance of the recommender with nDCG

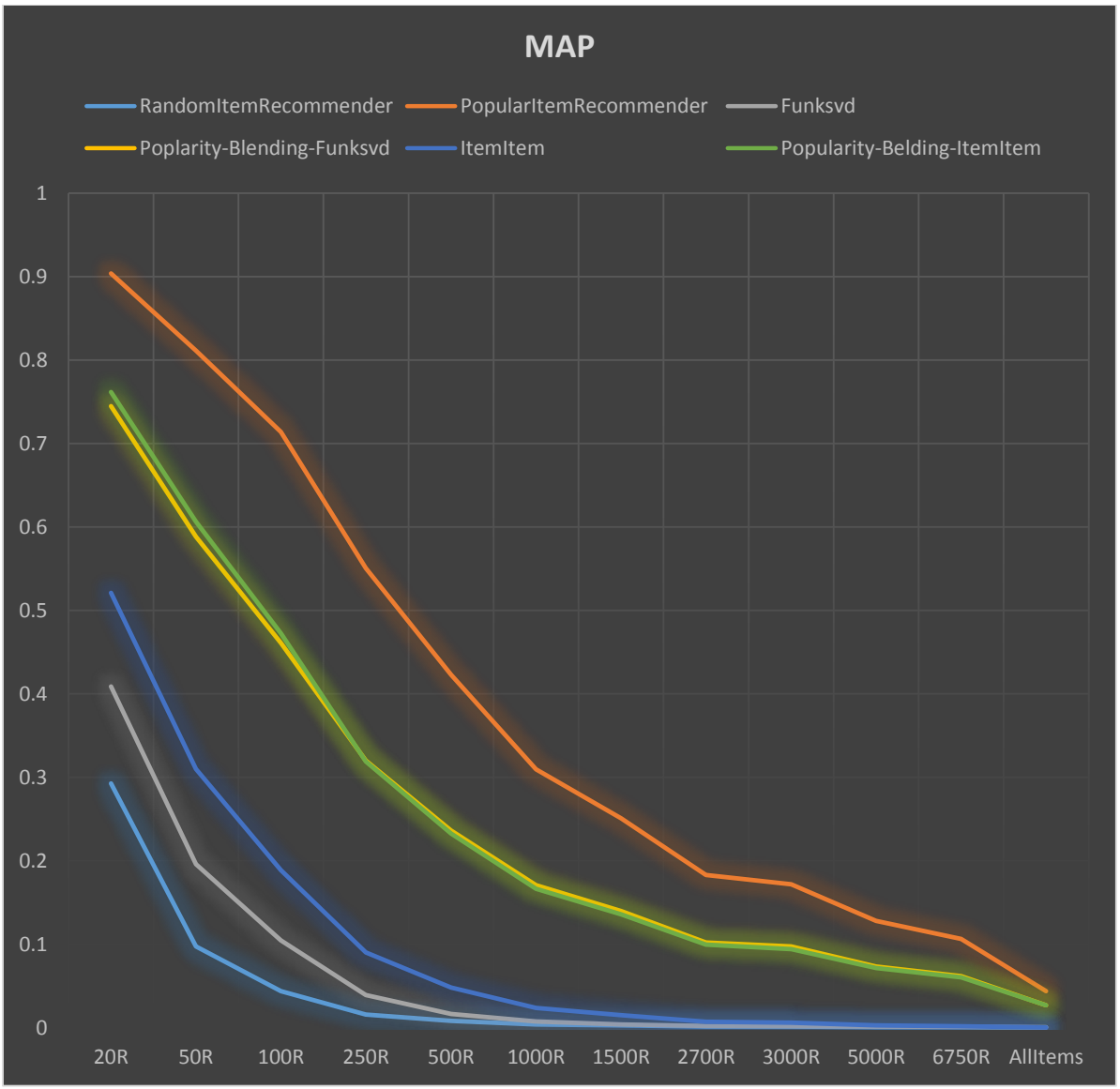

Fig 16. Performance of the recommender with MAP

Fourth Run (With BookCrossing)

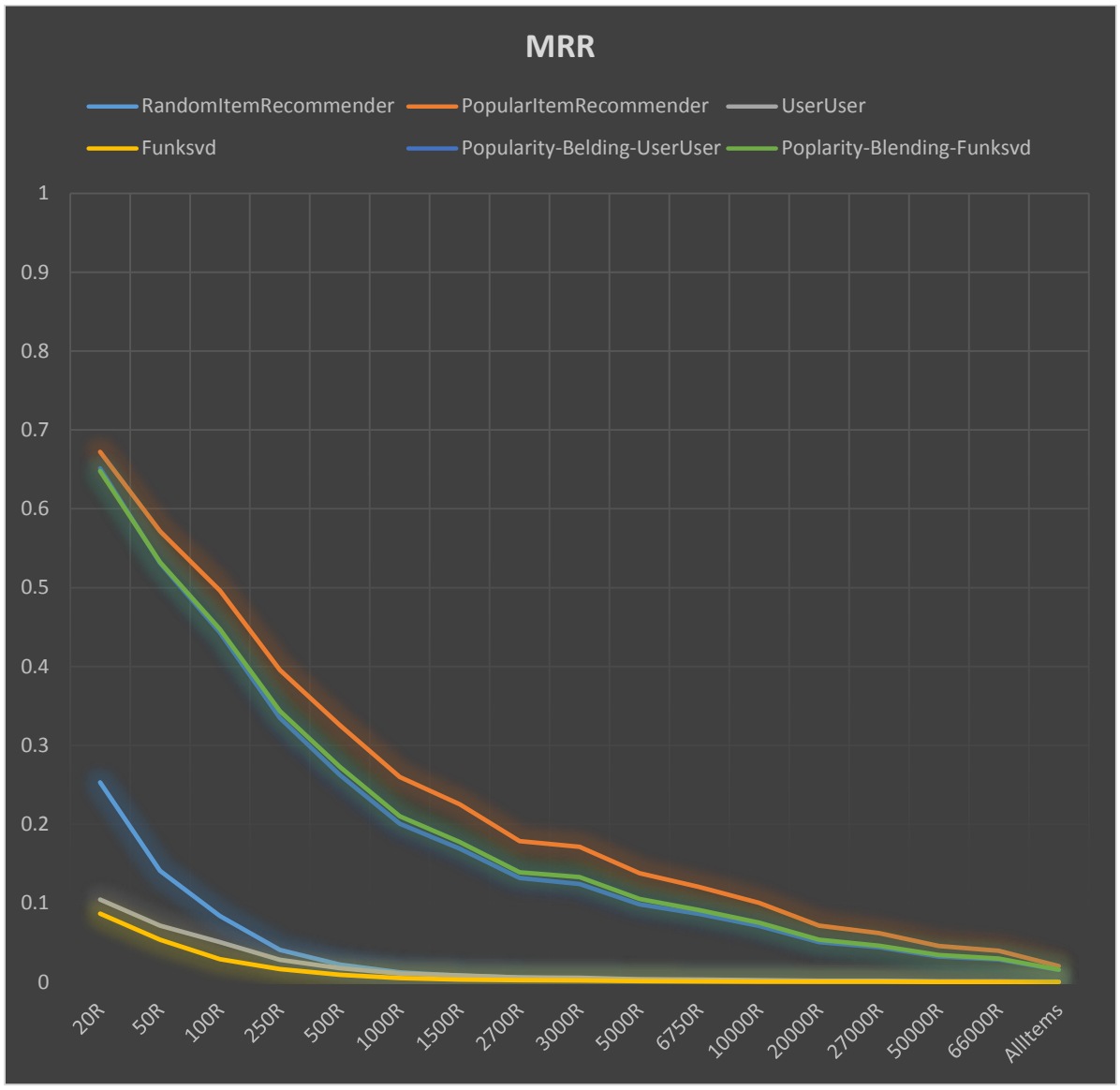

Fig 17. Performance of the recommender with MRR

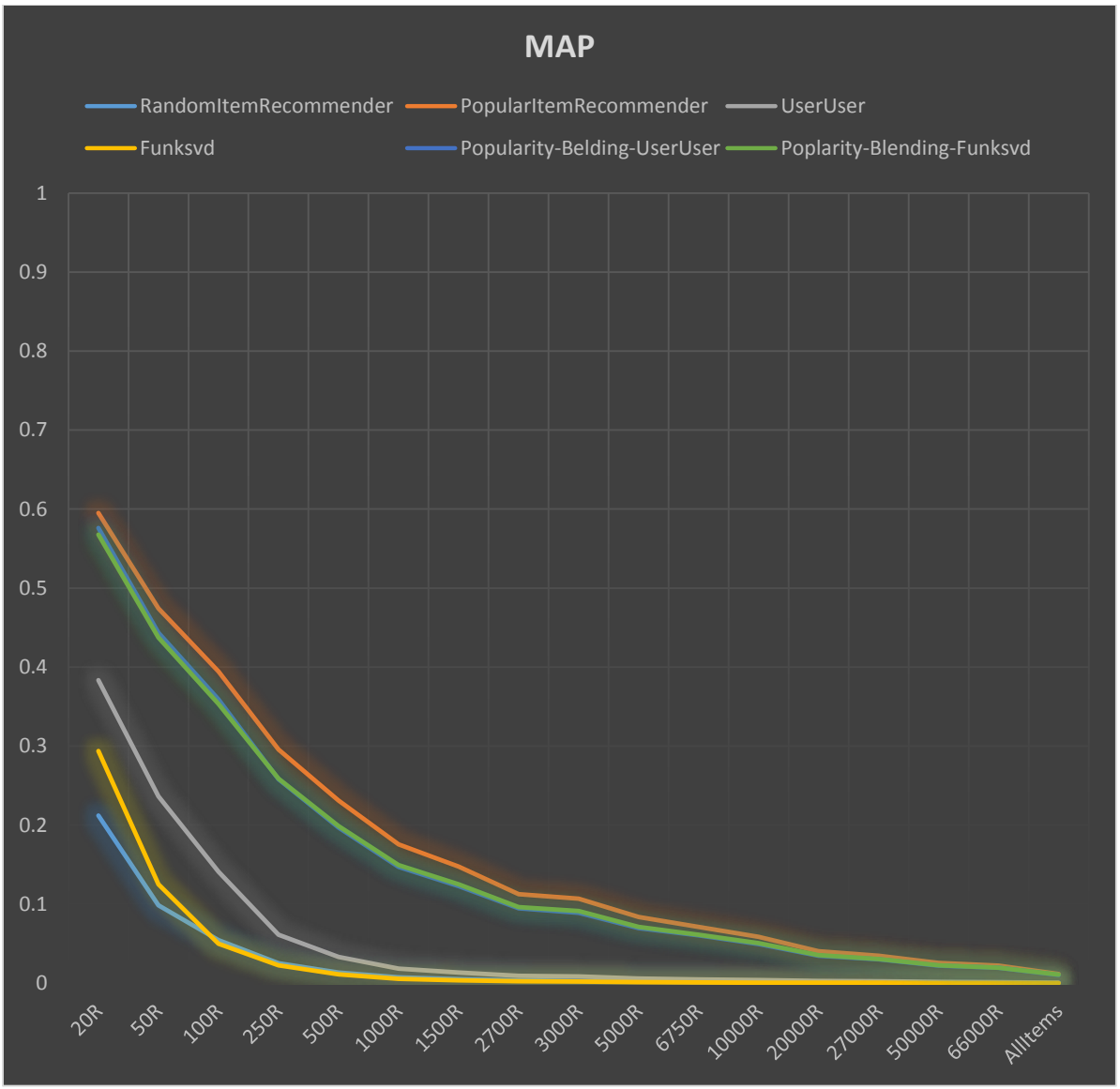

Fig 18. Performance of the recommender with MAP

V. CONCLUSION AND FUTURE WORK

We wanted to solve the problem of fully-coded-corpus with the method that uses random decoys in the test set of the recommender in order to keep the predicted unseen item in the test set, so that the recommender will not get penalized for recommending that item. Our results here show this method of adding random decoys are actually giving advantage to the popular item recommender. What is happening here is for the relatively small seize of the decoy set the it more likely that the item being picked by the popularity based recommender comes from the test set. Because for smaller decoy set there is less chance that it may contain a good amount of popular item. The popularity based recommenders always picks the most popular items. So, the item being picked for recommendation is probably belongs to the test set, that's why it can pick it, because for popular item they usually have more ratings than the other items.

As we increase the size of the random decoy set, the probability that it may contain some popular item is increasing. Even after selecting 25% of the data as decoys the performance of the popularity based recommender is measured to be a bit better than other recommenders, and that is because of the strategy of the popularity based recommender which is 'pick the most popular item', which are likely to be from the test set, because the recommender can pick them from there reliably.

Future extension of this work can include implementation of Bellogin et al 2011, method of reducing the popularity bias. Also different decoy selection strategies can also be used, for example random selection of decoys based on the popularity of the item.

REFERENCES

Adams, R. A. (2007). Music Recommendation using Collaborative Filtering with Similarity Fusion. Master's thesis, Department of Computer Science, The University of York.

Aslam, J. A., Pavlu, V., & Yilmaz, E. (2006, August). A statistical method for system evaluation using incomplete judgments. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 541-548). ACM.

Bellogín, A. (2009). Performance prediction in recommender systems: application to the dynamic optimisation of aggregative methods. Master's thesis, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Madrid, Spain.

Bellogín, A., & Castells, P. (2010). A performance prediction approach to enhance collaborative filtering performance. In *Advances in Information Retrieval* (pp. 382-393). Springer Berlin Heidelberg.

Bellogín, A., Wang, J., & Castells, P. (2011). Text retrieval methods for item ranking in collaborative filtering. In *Advances in Information Retrieval* (pp. 301-306). Springer Berlin Heidelberg.

Bellogin, A., Castells, P., & Cantador, I. (2011, October). Precision-oriented evaluation of recommender systems: an algorithmic comparison. In*Proceedings of the fifth ACM conference on Recommender systems* (pp. 333-336). ACM.

Bellogín, A. (2012). *Recommender System Performance Evaluation and Prediction: An Information Retrieval Perspective* (Doctoral dissertation, PhD thesis, Universidad Autónoma de Madrid).

Balabanović, M., & Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Communications of the ACM*, *40*(3), 66-72.

Cremonesi, P., Koren, Y., & Turrin, R. (2010, September). Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems* (pp. 39-46). ACM.

Cremonesi, P., Garzotto, F., Negro, S., Papadopoulos, A. V., & Turrin, R. (2011, September). Looking for "good" recommendations: A comparative evaluation of recommender systems. In *IFIP Conference on Human-Computer Interaction* (pp. 152-168). Springer Berlin Heidelberg.

Cremonesi, P., Garzotto, F., & Turrin, R. (2012). Investigating the persuasion potential of recommender systems from a quality perspective: An empirical study. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, *2*(2), 11.

de Máster, T. F. (2009). Performance prediction in recommender systems: application to the dynamic optimisation of aggregative methods.

Ekstrand, M.D., Kluver, D., Harper, F.M. and Konstan, J.A. (2015) 'Letting users choose Recommender Algorithms', *Proceedings of the 9th ACM Conference on Recommender Systems - RecSys '15*, . doi: 10.1145/2792838.2800195.

Michael D. Ekstrand, Michael Ludwig, Joseph A. Konstan, and John T. Riedl. 2011. Rethinking The Recommender Research Ecosystem: Reproducibility, Openness, and LensKit. In *Proceedings of the Fifth ACM Conference on Recommender Systems* (RecSys '11). ACM, New York, NY, USA, 133-140. DOI=10.1145/2043932.2043958.

Gunawardana, A., & Shani, G. (2009). A survey of accuracy evaluation metrics of recommendation tasks. *The Journal of Machine Learning Research*, *10*, 2935-2962.

Harper, F. M., & Konstan, J. A. (2016). The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, *5*(4), 19.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, *22*(1), 5-53.

Herlocker, J., Konstan, J. A., & Riedl, J. (2002). An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms.*Information retrieval*, *5*(4), 287-310.

Konstan, J. A. (2004). Introduction to recommender systems: Algorithms and evaluation. *ACM Transactions on Information Systems (TOIS)*, *22*(1), 1-4.

Koren, Y. (2008, August). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*(pp. 426-434). ACM.

Shani, G., & Gunawardana, A. (2011). Evaluating recommendation systems. In*Recommender systems handbook* (pp. 257-297). Springer US.

Urbano, J., Schedl, M., & Serra, X. (2013). Evaluation in music information retrieval. *Journal of Intelligent Information Systems*, *41*(3), 345-369.

Wu, W., He, L., & Yang, J. (2012, August). Evaluating recommender systems. In *Digital Information Management (ICDIM), 2012 Seventh International Conference on* (pp. 56-61). IEEE.

Zaier, Z., Godin, R., & Faucher, L. (2008, November). Evaluating recommender systems. In Automated solutions for Cross Media Content and Multi-channel Distribution, 2008. AXMEDIS'08. International Conference on (pp. 211-217). IEEE.

Ziegler, C. N., McNee, S. M., Konstan, J. A., & Lausen, G. (2005, May). Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web* (pp. 22-32). ACM.