Goal-driven Collaborative Filtering
- A Directional Error Based Approach

Tamas Jambor and Jun Wang

Department of Computer Science
University College London
Malet Place, London, WC1E 6BT, UK
t.jambor@ucl.ac.uk, wang.jun@acm.org

Abstract. Collaborative filtering is one of the most effective techniques for making personalized content recommendation. In the literature, a common experimental setup in the modeling phase is to minimize, either explicitly or implicitly, the (expected) error between the predicted ratings and the true user ratings, while in the evaluation phase, the resulting model is again assessed by that error. In this paper, we argue that defining an error function that is fixed across rating scales is however limited, and different applications may have different recommendation goals thus error functions. For example, in some cases, we might be more concerned about the highly predicted items than the ones with low ratings (precision minded), while in other cases, we want to make sure not to miss any highly rated items (recall minded). Additionally, some applications might require to produce a top-N recommendation list, where the rank-based performance measure becomes valid. To address this issue, we propose a flexible optimization framework that can adapt to individual recommendation goals. We introduce a *Directional Error Function* to capture the cost (risk) of each individual predictions, and it can be learned from the specified performance measures at hand. Our preliminary experiments on a real data set demonstrate that significant performance gains have been achieved.

1 Introduction

Collaborative filtering (CF) is concerned with predicting how likely a specific user will like certain information items (books, movies, music items, web pages, etc). As the term "collaborative" probably implies, the prediction has to rely on a collection of other (similar) users' preferences, which have been collaboratively collected. One of the popular applications of collaborative filtering is personalized content recommendation. A typical example is movie recommendation, where a user is explicitly asked to rate what he or she liked or disliked in the past. After rating a few movie items, the recommendation engine would be able to produce a prediction about the users ratings of unseen movie items by looking at other (similar) users past ratings for the movies items in question. In this case, users have to explicitly provide their ratings for movie items beforehand, e.g., give 1 star for the lowest rating (most hated) and 5 stars for the highest rating (most liked). As a major recommendation technique, collaborative filtering has been widely used in practice.

The first Netflix competition [1] posed a challenge to develop systems that could beat the accuracy of Neflix in-house recommender by 10 percent. One of the importance of this challenge is that it specified an evaluation metric that is

to be used to measure the efficiency of the system. Therefore forcing developers to think along the line of this measure. This would result in outcomes that have the same shortfalls as the measure [2]. This paper attempts to take another point of view of designing recommender systems. The aim is to introduce a design pattern that takes into account user preferences which would define the system itself. Different measures emphasize different qualities with respect to how closely they are correlated with certain objectives that the system would achieve. Therefore the measure itself gives a good indication of the qualities that the algorithm should possess. This approach offers a different solution. It enables the algorithm to be adjusted to user needs flexibly given that these needs are already defined and do not change during the session. It attempts to optimize the algorithm to these user needs instead of a measure which results in greater flexibility and better user experience. To achieve this, we first critically examine the issues of using squared errors as a cost function in collaborative filtering. Based on this discussion, we propose a goal-driven optimization framework where the users' or system's goal can be specified as a weight function. This weight function will be optimized by a genetic algorithm [3]. Experimental results on a real data set confirm our insights with improved performance.

The paper is organized as follows. We will discuss the related work in Section 2, present our theoretical development in Section 3, give our empirical investigation on recommendation in Section 4, and conclude in Section 5.

2 Related work

The term, collaborative filtering, was first coined in [4] where the authors developed an automatic filtering system for electronic mail, called Tapestry. If we look at the collaborative filtering problem from a conceptual level, it is very much like Web retrieval in that it needs to calculate the correspondence (called relevance) between a user information need (in our case, a user preference or predefined preferable topics) and an information item (e.g., a movie or a book) [5]. In text retrieval, the correspondence is usually calculated by looking at content descriptions, e.g., how many and how frequent the query terms occur with a document. In contrast, when we make personalized recommendations, users unseen preferences can be predicted by aggregating the opinions and preferences of previous users.

Originally, the idea of collaborative filtering was derived from heuristics, assuming that users who have similar preferences in the past are likely to have similar preferences in the future, and the more similar they are, the more likely they would agree with each other in the future. The preference prediction is therefore calculated by weighted-averaging of the ratings from similar users.

In the memory-based approaches, all user ratings are indexed and stored into memory, forming a heuristic implementation of the "Word of Mouth" phenomenon. In the rating prediction phase, similar users or (and) items are sorted based on the memorized ratings. Relying on the ratings of these similar users or (and) items, a prediction of an item rating for a test user can be generated. Examples of memory-based collaborative filtering include user-based methods [6], item-based methods [7] and combined methods [8].

In the model-based approaches, training examples are used to generate an abstraction (model parameters) that is able to predict the ratings for items that a test user has not rated before. In this regard, many probabilistic models have

been proposed. For example, to consider user correlation, [9] proposed a method called personality diagnosis (PD), treating each user as a separate cluster and assuming a Gaussian noise applied to all ratings. On the other hand, to model item correlation, [10] utilizes a Bayesian Network model, in which the conditional probabilities between items are maintained. Some researchers have tried mixture models, explicitly assuming some hidden variables embedded in the rating data. Examples include the cluster model [10] and the latent factor model [11]. These methods require some assumptions about the underlying data structures and the resulting 'compact' models solve the data sparsity problem to a certain extent. However, the need to tune an often significant number of parameters has prevented these methods from practical usage.

Alternatively, collaborative filtering can be considered as a matrix factorization problem and it has emerged as the clear favorite in the Netflix competition [12]. In general, the approach aims to characterize both items and users by vectors of factors inferred from item-rating patterns. The approximation is usually found such that it minimizes the sum of the squared distances between the known entries and their predictions. One possibility of doing so is by using a Singular Value Decomposition (SVD) [13]. The main reason of its success may be due to the fact that the objective function of the approach is equivalent to the performance measure (Root Mean Squared Error) that has been targeted in the competition. The drawback of using the RMSE performance measure is studied in [14]. However, it is also important to point out that many collaborative filtering models use the squared errors as the objective function, either implicitly or explicitly. Part of this paper is intended to increase the awareness and provide a future study about the issues of using RMSE as the objective function.

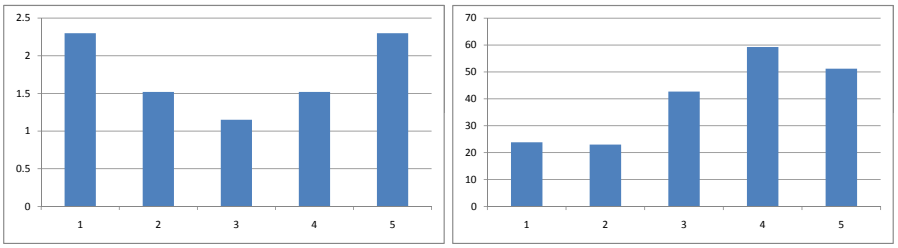
On the other hand, researchers also argued that understanding collaborative filtering as a rating prediction problem has some significant drawbacks. In some of the cases, a better view of the task is of generating a top-N list of items that the user is most likely to like [15, 16]. Thus, this paper attempts to develop a flexible goal-driven optimization framework so that the algorithm can be tailored to meet individual system requirements.

3 A Goal-driven Optimization Framework

3.1 Problem Definition and Analysis

One of the most used performance (error) measures for rating-based recommender systems is RMSE (Root Mean Squared Error), which measures the difference between ratings predicted by a recommendation algorithm and ratings observed from the users. It is defined as the square root of the mean squared error: $\sqrt{E\big((\hat{r}-r)^2\big)}$, where E denotes the expectation, r is the true rating and $\hat{r}$ is the predicted value. In [14], researchers have already systematically examined many performance measures for collaborative filtering. In this section, we critically examine the RMSE measure as an objective function.

The RMSE metric measures recommendation error across different rating scales, and the error criterion is uniform over all the items. RMSE squares the error before submitting it which puts more emphasis on large errors. Naturally large errors can occur at the end of the rating scales. To see this, suppose we have a recommendation algorithm which predicts the rating of an item randomly



(a) (b)

Fig. 1. (a) The expected value of RMSE per rating. (b) Percentage of RMSE improvement over random recommendation by a common collaborative filtering algorithm that optimizes RMSE.

from rating 1 to 5. Fig. 1(a) shows that it is more likely to get higher error at both the ends of the rating scale if a random algorithm is used.

Thus, the question arises if we should adopt RMSE as the measure of customer satisfaction. It measures the error across the system even for items that are not that important for users to be correctly predicted. Therefore the system might not want to penalize predictions that are not important for the user. If the user is only interested to get relevant recommendations, RMSE as a measure is not sufficient. Even if the user is interested in items that he or she would dislike, it is arguable whether the middle range of the rating spectrum is interesting to the user at all. If we take rating three out of five as middle, that range cannot help to explain why an item was recommended, neither can it explain why the item was not recommended.

Another issue may also arise if one directly optimizes RMSE. This is due to the fact that the training samples are not uniformly distributed across rating scales. To demonstrate this, Fig. 1(b) shows RMSE improvement in percentage over random recommendation by using a common recommendation algorithm (in this case an SVD-based approach is used to optimize the metric [12]). Since users are likely to rate items that they liked, in most cases, they give them a rating of four. So the algorithm has more data to make a prediction at that range. This is the reason why we have higher improvements for rate four.

Improving accuracy on items that the user would like may be desirable from a user point of view, but if the prediction falls into the middle range the error does not matter as much as if the prediction falls into the lower range. It is similar with items that are rated low, reducing the error rate is more desirable as the error rate increases since the item gets a higher prediction. In addition to that, highly accurate predictions on uninteresting items (perhaps rated 3 out of 5) can drown out poor performance on highly/lowly ranked items. Therefore depending on the rating we need to pay attention to the direction of the offset between the rating and the prediction. Fig. 2 shows that the SVD algorithm tends to overpredict items in the middle range. Also, it is more likely to overpredict lower rated items than underpredict higher rated items.

Therefore a distinction can be made between items that are interesting to the user and items that are neutral. Within the interesting category we can differentiate between liked and disliked items. To decided which one is the most important to us, let us consider two different type of recommendations. Since the performance of a recommender system should be evaluated with respect to a

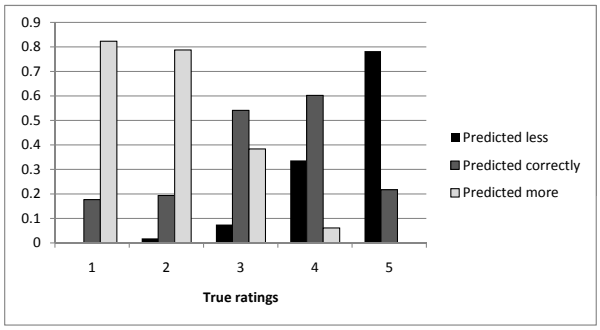


Fig. 2. Asymmetric Rating-prediction error offset.

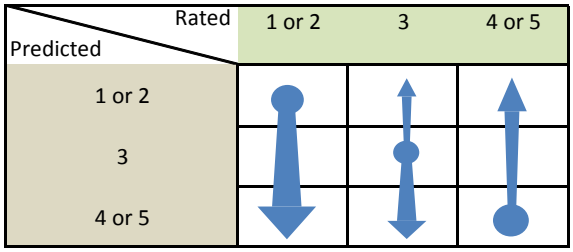


Fig. 3. Directional Risk Preference of Recommendation Prediction.

specific task, it is useful to define the two main tasks that a typical recommender system fulfills. If the output of the recommender system is the first n items then RMSE is not an appropriate objective to optimize, since it is not important to measure the system performance on items that do not fall into the first n good items. As long as the system correctly identified that these items do not fall into the n good items the accuracy is irrelevant. Users might be interested in exploring movies, looking through the database or checking particular movies. In this case everything matters, because users are interested in the justification on how movies are made. Clearly, in both scenarios we can differentiate between two separate kind of risks. First, the risk of recommending something that is not relevant to the users, second, the risk of not recommending something that is relevant to the users. These are two kind of errors, that should be separated when it comes to measuring the error rate. For example assume that the system predicts a movie four, and the user watches that movie, which he or she would have rated only three. This is clearly different from the case where the system rates a movie three, which would have been rated four is the user took the time and watched it. Since the error of the algorithm in the second scenario would never be found out, because the user would never watch a movie that is rated three, from a user point of view this error would be hidden. Therefore the system that makes errors like that would not considered better by the user than a system that makes errors illustrated in the first scenario.

Therefore it is important to introduce two concepts here. *Taste boundaries* and the *direction of taste*. Taste boundary could be defined as the interval that is between liked and disliked items. In a rating scale from one to five this boundary would be three. Direction would represent whether the predicted rating is towards the taste boundary or not, at one level, on another level it would rep-

Table 1. The two dimensional weighting function, where p is the predicted value of the item and r is the ground truth.

	$r = 1, 2$	$r = 3$	$r = 4, 5$
$p <= 2.5$	w_1	w_2	w_3
$2.5 < p <= 3.5$	w_4	w_5	w_6
$p > 3.5$	w_7	w_8	w_9

resent whether the error is large enough to cross the taste boundary or not. In other words, whether the algorithm suggests that the user would like the item when it is not the case and vice versa. These boundaries are illustrated in the matrix shown in Fig. 3. It shows that we would like to minimize errors where the prediction is correct and as we go further from the correct prediction we take higher risks depending on the direction (the risk is illustrated by the size of the arrows). Fig. 3 can also be applied to a ranking problem since higher predicted items represent higher risk. For example in ranking an error should be penalized more if an item is ranked higher than if it happens the other way around.

3.2 Optimizing the Weighted Errors

Based on our discussion, we should penalize more for more risk given a specified recommendation goal. Also, risk is directional as shown in Fig. 3. Previous recommender systems considered the absolute value of the error, taking equally into account negative distance and positive distance from the ground truth. Here, we propose an optimization framework that would differentiate between negative and positive distance between the prediction and the ground truth rating, assigning a higher penalty for positive distance than negative distance. To achieve this, we assign a weight for each type of error. As shown in Fig. 3, the weights are two dimensional, depending on both the prediction and the ground truth. Mathematically, we optimize the following proposed objective function in order to obtain the parameters of a recommendation model:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \sum_{u,i} w\big(f(\theta), r_{u,i}\big)\big(f(\theta) - r_{u,i}\big)^2 \tag{1}$$

where $f(\theta)$ denotes the recommendation model parameterized by θ. u and i are the user and item index respectively. $w\big(f(\theta), r_{u,i}\big)$ denotes the cost (or risk) weighting function. We can solve the optimization problem by applying a Gradient Descent method [17], which requires to differentiate the objective function as follows:

$$\sum_{u,i} 2w(f(\theta) - r_{u,i})f'(\theta)d\theta + w'\big(f(\theta) - r_{u,i}\big)^2 f'(\theta)d\theta \tag{2}$$

A discrete form of the weighting function is adopted in this paper (see in Table 1) – the risk preference of the system is thus captured by the nine weights. Because the weight w is constant for the three regions defined by f, the expression can be further approximated by:

$$\sum_{u,i} 2w(f(\theta) - r_{u,i})f'(\theta)d\theta \tag{3}$$

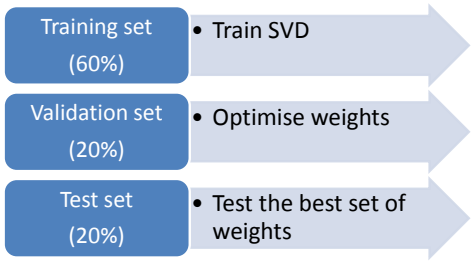


Fig. 4. Two-level optimization

To demonstrate the optimization framework, we adopt an incremental SVD (Singular Value Decomposition) factorization method [12], which is defined as follows:

$$\underset{q,p}{\operatorname{argmin}} \sum_{u,i} w(r_{u,i} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2) \tag{4}$$

where $f = q_i^T p_u$, where q and p are the model parameters. This algorithm factors the matrix using only user and item pairs where $r_{u,i}$ is known. As mentioned earlier we introduced a weight w depending on the given criteria that is added to the equation. Therefore w is introduced, aiming to control the magnitude how conservative the system is to be in a given rating sector.

The system learns the model by fitting the previously observed rating. In order to avoid overfitting the second half of the equation regularizes the learning parameters and the constant λ is set to control the extent of regularization. Stochastic gradient descent is used to optimize the equation [12] introduced by [18].

The next question is how to obtain the optimal weighting w given a recommendation goal. Normally, a recommendation goal can be defined by a performance metric. For example, if the output is a ranked recommendation list, rank-based metrics such as NDCG [19] might be suitable. We adopt a Genetic algorithm to obtain the optimal weights. Genetic algorithms are search algorithms that work via the process of natural selection. They begin with a sample set of potential solutions which then evolves toward a set of more optimal solutions. Within the sample set, solutions that are poor tend to die out while better solutions remain in the population, thus introducing more solutions into the set. The genetic algorithm does its best when there is a smooth slope of fitness over the problem space towards the optimum solution. This approach requires a two-level optimization illustrated in Fig. 4.

4 Evaluation

4.1 Experiment Setups

We empirically investigated the relationships between the taste boundaries and the CF performance, using the MovieLens dataset. This publicly available dataset

consist of 100,000 ratings for 1682 movies by 943 users. We divided the dataset into three parts (Fig. 4) making sure that ratings from any given user are in all of the sets. Every user in the dataset rated at least 20 movies and the movies from each user distributed randomly when the dataset was divided. This is an important criterion since the performance measures that are discussed below consider users as a point of evaluation. The result is cross-validated using a five-fold cross-validation method and the outcomes are averaged.

The algorithm measures system effectiveness based on two assumptions. First, we consider recommendation as a ranking problem. Second, we define risk in nine different sectors (Fig. 3) which can be adjusted based on the desired outcome of the system. Even if the goal is to measure the effectiveness of the system across all users and items, from a user point of view there are items that are more important than others. If we consider recommendation as a ranking problem, it is sensible to optimize the algorithm using some of the measures from IR. Therefore two main concepts from IR should be defined in the domain of recommendations. Relevance shows whether an item is relevant to the query issued. However, the query is hidden in a recommender system, since it is defined by the user's preference which is usually not expressed explicitly. In this paper we make an assumption that users would watch a movie if it is rated four or five on a five point scale (relevant), but we acknowledge that this might be different for individual users. Therefore relevance is defined on a binary scale. Movies that are rated four or five are considered relevant, the rest of the movies are considered irrelevant. Retrieved items represent a list of items that are presented to the user. This concept might be important if the task of the system is to return the first-N relevant items. In order to reach the desired effect we evaluated the system using measures from IR. For a given user the algorithm ranks unseen movies such that the movies he or she likes most are suggested first. The following performance measures are used in this experiment.

The Mean reciprocal rank (MRR) [20] is the average of the reciprocal ranks of results for all users in the dataset. This measure only takes into account the first relevant item in the list. So the algorithm would achieve a high score if all the items that are relevant are predicted correctly.

Mean average precision (MAP) [20] obtains the precision score after each relevant document is retrieved. The mean of this score is calculated for all users to obtain the MAP score. The algorithm would achieve a higher score if it improves the precision in the retrieved list. So in this case all the documents that are retrieved count toward the score.

Normalized discounted cumulative gain (NDCG) [20] measures the gain based on the items position in the recommended list. This measure was introduced in [21]. It penalizes the system if it returns highly relevant documents lower in the ranking list but penalizes less if the lower end of the ranking list was retrieved incorrectly. NDCG is normalized by the perfect permutation of all the documents in the set. One of the problems if we apply it to recommendation is that the average number of ratings by user is relatively low. The aspect of picking the right k elements from a big dataset is lost here. Therefore all users are evaluated on a fixed number of items which is set. So most of the time all of the considered items are retrieved. Thus, there is no penalty on having the wrong elements within the retrieved documents, the only penalty can arise from the wrong order. In this experiment we used the formula defined in [22].

Table 2. Baseline SVD

	$r = 1, 2$	$r = 3$	$r = 4, 5$
$p <= 2.5$	0.05175	0.01935	0.0106
$2.5 < p <= 3.5$	0.0904	0.1461	0.1391
$p > 3.5$	0.02995	0.10125	0.4115

Table 3. SVD with weights where $w7 > w8 > w4$

	$r = 1, 2$	$r = 3$	$r = 4, 5$
$p <= 2.5$	0.0759	0.04075	0.0264
$2.5 < p <= 3.5$	0.0837	0.16765	0.23815
$p > 3.5$	0.0125	0.0583	0.29665

Since we used a small set it was also important to define the best solution to the problem and use a measure that is relative to the best solution. This is particularly important for MRR. As mentioned above in collaborative filtering the algorithm is tested in a relatively small set compared to sets used in IR. Therefore it is more likely that the algorithm is tested on users where all the items are non-relevant. So the algorithm does not have a chance to return relevant documents from a set where there is not any relevant documents. This would decrease the performance of the algorithm. Therefore, in this experiment the algorithm disregards users where there is no relevant documents in the test set. This is a reasonable assumption, because if the algorithm runs on the whole database it is very likely that there will be at least one item that is relevant to the user.

4.2 Results

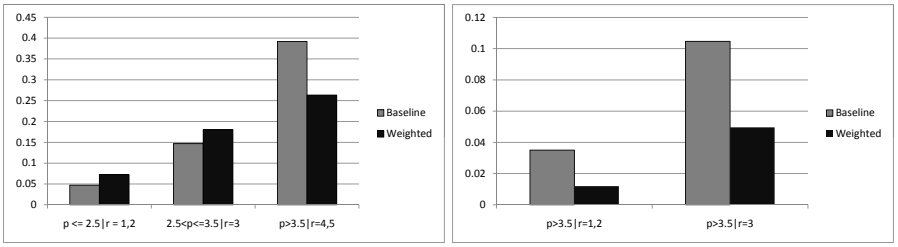
Introducing weight for the error would penalize unwanted categories therefore the recommender prefers to have higher error rate in categories that are not penalized. For example if an item was rated five by the baseline recommender with the ground truth of one, the weighted recommender in Table 3 would more likely to rate it less than three instead, reducing the error, but increasing the error on items that are not that important (e.g. items where the ground truth is one).

The first experiment aimed to demonstrate that introducing weight in different sectors would reduce the number of items that fell into those sectors. We introduced weights in sectors where the algorithm would make a higher prediction than the ground truth ($w4, w7, w8$) and set the magnitude of the weight in the order of risk illustrated in Fig. 3. Table 3 shows that the probability that an item would fall into those sectors is reduced. However, this is a trade-off since it reduces the number of items in the sector (items that are rated five and predicted five). On the positive side, it increases the accuracy in the middle and lower range.

In the second experiment weights are set to one by default at points where the prediction should be the most accurate as defined by Fig. 3. In this case only weights that fall into the interval where prediction were higher than ground truth considered ($w4, w7, w8$). The rationale behind this choice is that the combination of this force (enabling the algorithm to modify only these weights) and the measure would result in an optimal solution for the user where higher rated items

Table 4. Experimental results

	Measure(Test)	Baseline(Test)
MAP	0.450	0.447
MRR	0.899	0.889
NDCG@10	0.726	0.720
NDCG@5	0.574	0.570
NDCG@3	0.450	0.447



(a) (b)

Fig. 5. (a) The probability of correct predictions within sectors described in Table 1. (b) The probability of predicting an item relevant when it is not.

are considered more important and items that are overpredicted are penalized. Table 4 shows the result of the four performance measures that are used in our experiment. The first column indicates the score that is computed using the optimal weights and the second column is the baseline score (without weights). This shows that weights in fact improved the algorithm. The samples in the table are tested and found statistically significant. The reason why this method would provide a more robust recommendation from the user point of view is that it is reorganizing the ranking in a way that would take into account our initial criteria defined by the weights and re-rank it in a way that is ideal for these criteria. The advantage of the second approach is that it dynamically chooses the parameters for a given measure, however, as we will discuss below the measures do not cover all the possibilities given in our initial criterion. In contrast, the first approach can be tuned to reach a result that satisfies these criteria, but it cannot reach an optimal solution for all users.

Essentially this approach aims to minimize the error for the predefined sectors which inevitably results in the increase of error in other sectors. Fig. 5(a) shows the probability that true ratings are correctly predicted within our predefined taste boundary by the optimized versus the baseline approach using the weights obtained in the second experiment (Table 4). As expected the baseline approach predicts higher ratings better than our optimized approach, since the optimized approach does not penalize this type of error (high ratings predicted less), whereas we have some improvement in the lower range where we aimed to reduce the error. This approach takes the low risk approach therefore it hurts the performance at the higher range of the spectrum where it is less risky to predict something less, in exchange it reduces the error for item that are rated low. This means that it is less likely that users get items that are not relevant to them (Fig. 5(b)).

It is also important to investigate how the improvement of this evaluation metrics can be translated into improvement in user experience. Using MRR as a measure would reduce the probability that an irrelevant item would be presented to the user at the first position in the list. That implies that it would reduce the chance that lower rated items are rated higher for all items and it would also reduce the chance that higher rated items rated lower given that they are relevant items. The only place where it does not fit to our initial specification is that it does not differentiate between item and item within the irrelevant category, therefore there is not any difference in the score if an item rated one or and item rated three was ranked higher. Therefore parameter $w4$ does not add anything extra to this measure since it only penalizes low rated (one or two) items being predicted as uninteresting items. The same applies to NDCG, however it is a more subtle measure so it is able to differentiate between the order of the items in the ranked list. Therefore an NDCG score can tell us how well relevant items are ranked, which would be an optimal solution for the user.

5 Conclusion and future work

This paper presented a simple approach to optimize the outcome of collaborative filtering algorithm from the user point of view. This approach put an emphasis on the risk of making an incorrect recommendation. It considered recommendation in a more flexible way by taking into account predefined taste boundaries where users receive a list of items and are only interested in those items that are presented to the user. Another criterion is that items that are presented to the user are only interesting if they are within our predefined taste boundary. Therefore the algorithm aims to optimize its performance on those items. This approach can be fine-tuned further by considering how the items would be presented to the user. For example if a user would like to have just one recommendation, the algorithm is best optimized by MRR, or if the user would like to have more items recommended it would be better to optimized it by NDCG. The choice of parameters can be tailored to users need penalizing sectors that are more important to predict correctly to a specific user. For example calculating the mean of all the ratings for a particular user would suggest where the taste boundaries lie, so it can be determined for each individual user.

As it was discussed above all the measures only care about relevant items, but for our purposes it is also important to minimize error on disliked items (rated one or two). So we would like to measure how the algorithm performs on both sides of the rating scale. In both cases the middle range (items rated three) would be considered non-relevant. These two scores could be combined taking the high rated list more into account than the low rated one.

It is a widely discussed topic that accuracy alone is not a sufficient to measure whether a recommender system provides an effective and satisfying experience [2]. It is also important to note that a data is not homogeneous. In terms of prediction we can differentiate between easy and difficult items as well as easy and difficult users.

References

1. Bennett, J., Lanning, S.: The netflix prize. In: Proceedings of KDD Cup and Workshop. Volume 2007. (2007)

2. Herlocker, J., Konstan, J., Terveen, L., Riedl, J.: Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems **22**(1) (2004) 5–53
3. Mitchell, M.: An introduction to genetic algorithms. The MIT press (1998)
4. Goldberg, D., Nichols, D., Oki, B.M., Terry, D.: Using collaborative filtering to weave an information tapestry. Commun. ACM **35**(12) (1992) 61–70
5. Manning, C.D., Raghavan, P., Schtze, H.: Introduction to Information Retrieval. Cambridge University Press (2008)
6. Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In: SIGIR '99. (1999)
7. Deshpande, M., Karypis, G.: Item-based top-N recommendation algorithms. ACM Trans. Inf. Syst. **22**(1) (2004) 143–177
8. Wang, J., de Vries, A.P., Reinders, M.J.T.: Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In: SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, New York, NY, ACM Press (2006) 501–508
9. Pennock, D.M., Horvitz, E., Lawrence, S., Giles, C.L.: Collaborative filtering by personality diagnosis: A hybrid memory and model-based approach. In: UAI '00. (2000)
10. Breese, J., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98). (1998)
11. Canny, J.: Collaborative filtering with privacy via factor analysis. In: SIGIR '02. (2002)
12. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8) (2009) 30–37
13. Weimer, M., Karatzoglou, A., Smola, A.: Adaptive collaborative filtering. In: RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems, New York, NY, USA, ACM (2008) 275–282
14. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. **22**(1) (2004) 5–53
15. Wang, J., Roberston, S.E., de Vries, A.P., Reinders, M.J.T.: Probabilistic relevance models for collaborative filtering. Journal of Information Retrieval (2008)
16. Liu, N.N., Yang, Q.: Eigenrank: a ranking-oriented approach to collaborative filtering. In: SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval, New York, NY, USA, ACM (2008) 83–90
17. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley Interscience, Wiley, New York (2001)
18. Funk, S.: Netflix update: Try this at home. http://sifter.org/simon/journal/20061211.html (2006)
19. Jrvelin, K., Keklinen, J.: Cumulated gain-based evaluation of ir techniques. ACM Trans. Inf. Syst. **20**(4) (2002) 422–446
20. van Rijsbergen, C.J.: Information Retrieval. Butterworths, London, London, UK (1979)
21. Jarvelin, K., Kekalainen, J.: IR evaluation methods for retrieving highly relevant documents. In: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval, ACM New York, NY, USA (2000) 41–48
22. Vassilvitskii, S., Brill, E.: Using web-graph distance for relevance feedback in web search. In: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, ACM New York, NY, USA (2006) 147–153