Using Mutual Influence to Improve Recommendations

Aline Bessa, Adriano Veloso, and Nivio Ziviani

Universidade Federal de Minas Gerais
Department of Computer Science, Belo Horizonte, Brazil
{alinebessa,adrianov,nivio}@dcc.ufmg.br

Abstract. In this work we show how items in recommender systems mutually influence each other's utility and how it can be explored to improve recommendations. The way we model mutual influence is cheap and can be computed without requiring any source of content information about either items or users. We propose an algorithm that considers mutual influence to generate recommendations and analyse it over different recommendation datasets. We compare our algorithm with the $Top - N$ selection algorithm and obtain gains up to 17% in the utility of recommendations without affecting their diversity. We also analyse the scalability of our algorithm and show that it is as applicable for real-world recommender systems as $Top - N$.

Keywords: Recommender systems, theory of choice, mutual influence, collaborative filtering.

1 Introduction

Consumers from widely varying backgrounds are inundated with options that lead to a situation known as "information overload", where the presence of too much information interferes with decision-making processes [1]. To circumvent it, content providers and electronic retailers have to identify a small yet effective amount of information that matches users expectations. In this scenario, Recommender Systems have become tools of paramount importance, providing a few personalized recommendations that intend to suit user needs in a satisfactory way. One type of such systems, known as Collaborative Filtering [2], generally works as follows: (i) prediction step - keeps track of consumers known preferences to predict items that may be interesting to other consumers; (ii) recommendation step - selects predictions, ranks and recommends them to consumers.

Traditionally, predictions are scores assigned to items with respect to a certain consumer. The higher the score the higher the compatibility between the items in question and consumer's known preferences. It is therefore intuitive to think that the N items with highest scores should be the ones chosen in the recommendation step. This approach though, known as $Top - N$ recommendation, does not consider the utility of the recommended list as a whole, focusing exclusively on individual scores. As we show in this work, items exert a mutual influence on their utilities, i.e. the quality of an item depends not only on its own score but also on which other items are presented in the *Top* − *N* recommendations.

This work is motivated by the theory of choice of Amos Tversky [3], which indicates that preference among items depends not only on the items' specific features, but also on the presented alternatives. In the context of movies, for instance, it is equivalent to state that an action fan may prefer a mediocre karate movie over romance titles, but would not interact with this same karate movie when presented with better action films. We investigate the influence alternatives exert on each other, and how this information can be used to improve recommendations utility. It could be either embedded on the predictions computation or weighed right after they are generated in the recommendation step. We here focus on the recommendation step.

We propose a novel algorithm that incorporates mutual influence to perform the selection of a set of N items, which we call *GSMI - Greedy Selection based on Mutual Influence*. We conducted a systematic evaluation of *GSMI* involving different recommendation scenarios and distinct datasets. In order to evaluate *GSMI*, we used the utility metric proposed in [4] and measured diversity using the framework proposed in [5].

In summary, the main contributions of this work are (i) a cheap way of modelling and computing the mutual influence items exert on each other's utility, (ii) a new algorithm that considers mutual influence to select items in the recommendation step of a collaborative filtering, (iii) a thorough evaluation of its benefits in recommendation tasks – we compare *GSMI* with *Top* − *N* and obtain significant gains in the utility of recommendations without affecting their diversity –, (iv) an analysis of the scalability of *GSMI*, which indicates that the algorithm is applicable for real-world recommender systems.

This paper is structured as follows. Section 2 discusses previous related work and connects it to our study. Section 3 outlines some basic concepts that are the foundations of this work. Section 4 details *GSMI* and our evaluation methodology. Section 5 presents experiments that demonstrate the efficiency and efficacy of *GSMI*, taking different datasets into account. Finally, Section 6 details our conclusions and future work.

2 Related Work

In this section, we present related work in *Top* − *N* recommendations, dependencies among items, and learning to rank, as described next.

Top − *N Recommendation.* The state of the art prediction algorithms for *Top* − *N* recommendations, when explicit feedback is available, are PureSVD and NNCosNgbr (Non-normalized Cosine Neighborhood) [6]. A *Top* − *N* recommendation step when predictions are generated by these algorithms performs better than some sophisticated Learning to Rank methods. PureSVD is based on latent factors, i.e. users and items are modelled as vectors in a same vector space and the score of user u for item i is predicted via the inner-product between their corresponding vectors. NNCosNgbr works upon the concept of neighborhood, computing predictions according to feedback of similar users/items. [6] is related to our work because we use both PureSVD and NNCosNgbr as our prediction algorithms and compare *GSMI* with $Top - N$ for the recommendation step.

Dependencies among Items. In 1972, Amos Tversky proposed a model acording to which a user chooses among options by sets of item aspects – an example would be $\{price < \$100.00\}$ [3]. We do not assume that items' features are available and therefore do not model aspects in our approach, but we do rely on the idea that user choice depends on all presented alternatives – i.e. such alternatives interfere with each other's utility. Another work that relies on Economics principles to model dependencies among items is that of Wang [7]. Inspired by the Modern Portfolio Theory in finance, Wang derives a document ranking algorithm that extends the Probability Ranking Principle by considering both the uncertainty of relevance predictions and correlations between retrieved items. This work is the closest to ours.

Learning to Rank. LTR (*Learning to Rank*) are supervised methods to automatically build ranking models for items [8]. Although we do not generate an ordering among the items selected in the recommendation step, we do use supervised learning to compute mutual influence and perform selections. An LTR work that is somewhat close to ours is that of Xiong et al [9]. In an advertisement scenario, they observed that the CTR (*Click-Through Rate*) of an ad is often influenced by the other ads shown alongside. Based on it, they designed a Continuous Conditional Random Field for click prediction focusing on how ads influence each other. Another work that models influence among items and explores it to perform an LTR is [10]. In this paper, influences are modelled as similarities among items and embedded in a latent structured ranking method afterwards.

3 Basic Concepts

In this work, there are two fundamental sources of evidence that are used to select which items should be recommended to a certain user: (i) individual scores ϕ generated in the prediction step by either PureSVD or NNCosNgbr, and (ii) pairwise scores θ that quantify mutual influence among items. Both ϕ and θ are real values in the interval $[0, 1]$.

The pairwise scores θ work in a positive way: the higher they are, the higher the utility of the items in question when selected to a same recommendation list. Given items a and b, $\theta(a, b)$ should ideally be computed considering only cases where they are simultaneously selected to recommendation. Unfortunately, it is not possible to reconstruct the recommendation lists in none of the studied datasets. As a consequence, it is not possible to know which items were presented to users together and therefore we compute $\theta(a, b)$ considering users historical data as a whole, as detailed in Section 3.2.

In this work, we assume that predictions are generated to K items, and then N, $N \leq K$, items must be selected to compose a recommendation list. Typical values for N are 5 and 10, and depending on the prediction algorithm K can be equivalent to the total number of items in the dataset [8]. We also assume that users explicitly give feedback to items, and depending on the system it can be a rating, a purchase signal (0/1), a click (0/1) etc. Next, we detail how we compute individual scores ϕ, pairwise scores θ, and how they are combined.

3.1 Individual Scores ϕ

Individual scores are generated by prediction algorithms that are divided into two categories: neighborhood-based and model-based [8]. The latter have recently enjoyed much interest due to related outstanding results in the Netflix competition, a popular event in the recommender systems field that took place between 2006 and 2009[1]. Nonetheless, neighborhood-based prediction algorithms usually provide a more concise and intuitive justification for the computed predictions, and are more stable, being little affected by the addition of users, items, or ratings [8]. The predictors used in this paper are PureSVD (model-based) and NNCosNgbr (neighborhood-based), both state of the art methods for $Top - N$ recommendations when explicit feedback is available.

The input for PureSVD is a User $\times$ Item matrix M filled up as follows:

$$M_{ui} = \begin{cases} \text{numerical feedback, if consumer } u \text{ gave feedback about item } i, \\ 0, \text{if not.} \end{cases} \tag{1}$$

PureSVD consists in factorizing M via SVD as $M = U \times E \times Q$, where U is an orthonormal matrix, E is a diagonal matrix with the first γ singular values of M, and Q is also an orthonormal matrix. The prediction of an individual score $\phi(i)$ given a user u is thus given by:

$$\phi(i) = M_u \times Q^T \times Q_i \tag{2}$$

where M_u is the u-th row of M corresponding to user u latent factors, Q^T is the transpose of Q, and Q_i is the i-th row of Q corresponding to item i latent factors.

NNCosNgbr is a neighborhood model that bases its predictions on similarity relationships among either users or items. Working with items similarities usually lead to better accuracy rates and more scalability [11]. In this case, recommendations can be explained in terms of the items that users have already interacted with via ratings, purchases, likes etc [11]. Due to these reasons, we focus on item-based NNCosNgbr. The prediction of an individual score $\phi(i)$ given a user u is computed as follows:

$$\phi(i) = b_{ui} + \sum_{j \in D^k(u;i)} d_{ij}(r_{uj} - b_{uj}) \tag{3}$$

[1] `http://en.wikipedia.org/wiki/Netflix_Prize`

where b_{ui} is a combination of user and item biases as in [12], $D^k(u;i)$ is the set of k items rated by u that are the most similar to i, d_{ij} is the similarity between items i and j, r_{uj} is an actual feedback given by u to j and b_{uj} is the bias related to u and j.

Biases are taken into consideration because they mask the fundamental relations between items. Item biases include the fact that certain items tend to receive better feedback than others. Similarly, user biases include the tendency of certain users to give better feedback than others. Finally, the similarity among items, used to compute both $D^k(u;i)$ and d_{ij}, is measured with the adjusted cosine similarity [6].

3.2 Pairwise Scores $\boldsymbol{\theta}$

The pairwise scores $\theta(i,j)$ capture the mutual influence items i and j have on their own utility. In other words, $\theta(i,j)$ quantifies to what extent the selection of i is correlated with the selection of j and vice-versa. Given that it is not possible to track at what times i and j were selected together in the studied datasets, we compute $\theta(i,j)$ considering all their co-occurences in the historical data, regardless of when they were presented to users. A straightforward way of computing $\theta(i,j)$ is via Maximum Likelihood Estimator (MLE), as follows:

$$\theta(i,j) = \frac{l_{ij}}{f_{ij}} \tag{4}$$

where l_{ij} is the number of consumers that liked i and j and f_{ij} is the number of consumers that gave feedback to i and j. It turns out that this MLE computation yields good results, as detailed in Section 5.

The notion of *"liked"* can be understood as *"clicked"*, *"received a high rating"*, *"purchased"*, etc. The problem with computing θ via Equation 4 is that most items do not receive much feedback – i.e., recommendation datasets. As a consequence, using MLE to approximate the value of θ can lead to arbitrarily bad approximations. To make more realistic approximations, one can penalize pairs of items with a poor support, shrinking the computation with a factor λ [6]:

$$\theta(i,j) = \frac{f_{ij}}{\lambda + f_{ij}} \times \frac{l_{ij}}{f_{ij}} \tag{5}$$

Note that Equation 5 converges to Equation 4 when $\lambda \to 0$. The main challenge in using Equation 5 is to find an adequate value for λ.

3.3 Combining Scores

In this work, we combine individual and pairwise scores to select N items out of K for recommendation. The problem is therefore posed as selecting a set of items $I = \{i_1, ..., i_N\}$ that maximizes the following utility function:

$$\sum_{i_a \in I} \frac{\phi(i_a)}{|I|} + \sum_{(i_l, i_m) \in I^2} \frac{\theta(i_l, i_m)}{|I|^2} \tag{6}$$

where the normalization in both summations is important to keep their contributions fair – i.e., both values will remain in the interval $[0, 1]$.

There are some different ways of obtaining an exact solution to this optimization problem. For instance, one can trivially enumerate all N-combinations of a set with K items and choose the one that sums up to the highest value. It is also possible to use integer programming to solve it (NP-Hard) [13]. To the best of our knowledge, all these techniques are costly and there is no polynomial algorithm that maximizes this function in an exact way.

4 The *GSMI* Algorithm

Combining scores to select items for recommendation leads to an intractable optimization, as discussed previously. To tackle with this problem under a practical viewpoint, we propose *GSMI*, a greedy algorithm that selects N items, one at a time, taking into account items that were selected previously.

The algorithm receives a set of items $I = \{i_1, \ldots, i_k\}$ and their individual scores $\{\phi(i_1), \ldots, \phi(i_k)\}$, and returns a set R with N selected items, where $N \leq K$. It is described as follows.

Algorithm 1. *GSMI* Algorithm

```
1: $i \leftarrow \operatorname*{argmax}_{i \in I} \phi(i)$
2: $R \leftarrow \{i\}$
3: $I \leftarrow I \setminus \{i\}$
4: while $|R| < N$ do
5:     $j \leftarrow \operatorname*{argmax}_{j \in I} \sum_{a \in R_j} \frac{\phi(a)}{|R_j|} + \sum_{(b,c) \in R_j^2} \frac{\theta(b,c)}{|R_j|^2}$, where $R_j = R \cup \{j\}$
6:     $R \leftarrow R_j$
7:     $I \leftarrow I \setminus \{j\}$
8: end while
9: return $R$
```

GSMI starts selecting the item that has the best individual score, i. All other $N - 1$ selected items are chosen in a way that maximizes the equation in line 5, where the maximized set is comprised by all items that were already chosen, R, and the new item itself. The crucial greedy choice of *GSMI* is selecting the item with best individual score first.

GSMI runs in polynomial time. The loop in line 4 will be executed exactly $N - 1$ times. In line 5, an item is chosen out of $K - 1$ in the worst case; in the best case, out of $K - N + 1$ ones. It means that $O(K)$ items need to be analysed at each time. In line 5, the first summation is performed in $O(|R|)$ time. The second summation is performed in $O(|R|^2 \times \beta)$ time, where $O(\beta)$ is the time complexity of θ. An upper bound for the time complexity of *GSMI* is therefore $O(K + |R| \times (K \times |R|^2 \times \beta)) = O(KN^3\beta)$, given that $|R| \leq N$.

The time complexity of computing θ, $O(\beta)$, depends on the size of the dataset, on the maximum number of feedback given by a certain user to the items in question, and on the used data structures. For each loop iteration, it is possible to reuse partial summations from the previous iteration, in a way that the total time complexity is reduced to $O(KN^2\beta)$. Besides that, it is possible to optimize function calls to compute θ by taking advantage of its symmetry and by using memoization. Therefore, as we discuss in Section 5, it is simple to speed up *GSMI* and make it scalable to big datasets.

It is worth pointing out that *GSMI* is compatible with any recommender system where it is possible to estimate I, its corresponding scores $\{\phi(i_1), \ldots, \phi(i_k)\}$, and approximations for pairwise scores θ. Therefore, the proposed algorithm is *a priori* compatible with systems that employ both matrix factorization techniques and sketching/fingerprinting methods for dealing with big data.

To validate *GSMI*, we use the explicit feedback users give over items as a utility measurement: the better it is, the more useful the recommendations are [14]. The hypothesis we started investigating can therefore be simply posed as *"Does GSMI select items that receive better feedback when compared to those selected by $Top - N$?"* For all studied datasets, feedback consists of ratings. To perform the comparison between *GSMI* and $Top - N$ we thus measure the average rating users gave to recommended items, applying 5-fold cross-validation [4]. We generate individual scores for all $(user, item)$ pairs in the test set and then perform items selection using both *GSMI* and $Top - N$. The recommendation list containing items that receive higher ratings is the one that is considered more useful.

In this work we also compare *GSMI* and $Top - N$ under a diversity perspective. It has recently become a consensus that a desirable feature for successful recommender systems is the ability of generating diverse, non-monotonous recommendations to users [15]. Diversity is usually defined as the opposite of similarity, and the most explored approach for measuring it uses content-based similarity between items [8]. The diversity metric we apply, intra-list distance (ILD), was proposed by Zhang and Hurley [16] and works as follows:

$$ILD = \frac{2}{|R|(|R| - 1)} \sum_{i_k, i_l \in R, l < k} 1 - sim(i_k, i_l) \qquad (7)$$

where R is comprised by all selected items and $sim(i_k, i_l)$. More details of how we performed experiments with ILD are given in Section 5.

5 Experiments

In this work, we investigated mutual influence in three different datasets: MovieLens 100K[2], MovieLens 1M[3], and Jester 1[4]. Table 1 summarizes some of their characteristics.

[2] `http://www.grouplens.org/system/files/ml-100k.zip`

[3] `http://www.grouplens.org/system/files/ml-1m.zip`

[4] `http://goldberg.berkeley.edu/jester-data/jester-data-1.zip`

Table 1. Succint characterization of the studied datasets

Characteristic	MovieLens 100K	MovieLens 1M	Jester 1
Domain	Movies	Movies	Jokes
Feedback	Ratings (1 - 5)	Ratings (1 - 5)	Ratings (-10.00 - 10.00)
Number of users	943	6,040	24,983
Number of items	1,682	3,900	100
Number of feedback	100,000	1,000,209	1,810,455
Minimum ratings/user	20	20	36
Sparsity rate	0.937	0.958	0.275

Table 2. Average ratings given by users to items recommended by two different methods: $Top-N$ and $GSMI$. $GSMI-5$, $GSMI-10$, and $GSMI-20$ correspond to $GSMI$ selecting $N = 5, 10, 20$ items respectively. The average ratings were computed for different values of N using PureSVD and NNCosNgbr as predictors. For each $GSMI/Top-N$ pair, we performed a t-test over each dataset, and with a 95% confidence level only the underlined results are not statistically different.

Predictor	Method	MovieLens 100K	MovieLens 1M	Jester 1
PureSVD	$Top-5$	3.924	4.127	1.292
	$GSMI-5$	3.974	4.175	1.518
	$Top-10$	3.837	4.004	1.031
	$GSMI-10$	3.878	4.057	1.188
	$Top-20$	3.738	3.908	$\underline{0.892}$
	$GSMI-20$	3.765	3.939	$\underline{0.911}$
NNCosNgbr	$Top-5$	3.821	4.027	2.312
	$GSMI-5$	3.875	4.096	2.363
	$Top-10$	3.775	3.928	1.529
	$GSMI-10$	3.824	3.993	1.589
	$Top-20$	3.691	3.836	$\underline{0.939}$
	$GSMI-20$	3.726	3.890	$\underline{0.940}$

The MovieLens datasets are significantly more sparse than Jester 1. While in the former users rated at least 20 movies, in the latter users gave feedback to at least 36% of the jokes. MovieLens 1M is comprised by many more users and items than MovieLens 100K, and its total amount of ratings is similar to Jester's. Finally, while ratings in the MovieLens dataset are discretized and vary from 1 to 5, users in Jester 1 can assign any real number from -10.00 to 10.00 to any rated joke.

We compared $GSMI$ with the $Top-N$ approach in order to evaluate how mutual influence alone can bring up gain to recommender systems. Table 2 presents results for experiments with $N = 5, 10, 20$. For the MovieLens datasets, we considered that movies were liked by users if their ratings were equal or higher than 4; in the case of Jester 1, if they were equal or higher than 5.00. For all experiments, PureSVD was executed with 50 latent factors, the number of neighbors in $D^k(u;i)$ in Equation 3 was fixed in 60, and the value of λ for pairwise scores θ in Equation 5 was fixed in 0.5.

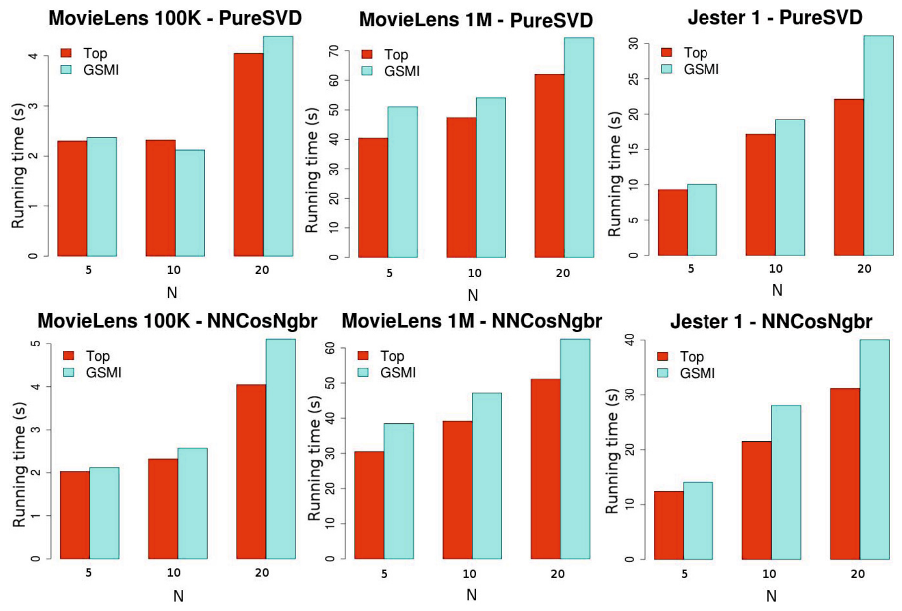

Fig. 1. Average running times per validation fold, in seconds, for different combinations of datasets and predictors, with $N = 5, 10, 20$

As shown in Table 2, predictor PureSVD generated better average ratings for both *Top* $- N$ and *GSMI* methods with respect to the MovieLens datasets. Concerning the Jester 1 dataset, NNCosNgbr performed better. In all cases, either *GSMI* produced superior average ratings or was statistically equivalent to the *Top* $- N$ results. The obtained gains were up to 17%. Although the difference between *Top* $- N$ and *GSMI* approaches may seem small, it is known that such differences have a huge impact on recommender systems [17].

5.1 Efficiency and Scalability of *GSMI*

GSMI is a greedy algorithm for the maximization problem posed by Equation 6. It is thus useful to compare it against exact solutions. We implemented such solutions for the MovieLens 100K dataset, with predictors PureSVD and NNCosNgbr and $N = 5$. The computations were carried out via the enumeration of all items combinations and posterior selection of the one that maximized Equation 6. Nonetheless, such computations took more than 3 hours to be completed, while *GSMI* generates results per validation fold in around 70 seconds in the worst case, as shown in Figure 1.

Regarding the utility of results, the average ratings obtained with the exact solutions were 3.984 and 3.895 for PureSVD and NNCosNgbr respectively. Both results were not statistically different from the corresponding average ratings obtained with *GSMI* for $N = 5$, 3.974 and 3.875, according to a t-test with a 95%

confidence level. The fact that *GSMI* for $N = 5$ generated statistically equivalent results for the MovieLens 100K dataset is an indicative that it is a good heuristic to approximate exact solutions. Also, results presented in Table 2 consistently indicate that *GSMI*, by embedding mutual influence in its selection strategy, can improve recommendations utility. As a consequence, it is important to devise competitive implementations for *GSMI* that scale in real-time situations.

Although *GSMI* is polynomial and rather fast, given that values for N are usually small in real-world scenarios [8], there are some easy and important optimizations that makes it scalable and competitive in practice. A first improvement is to precompute and store all pairwise scores θ in a hashtable as a preprocessing step. This offline computation speeds up the generation of different values for Equation 6 by avoiding redundant computations of Equation 5. Another improvement involves the use of memoization to reuse partial summations in the *GSMI* algorithm. Figure 1 illustrates the average computation time per validation fold for each dataset, varying N and the predictor algorithm. All experiments were performed in a Pentium Dual-Core 2.0GHz with 2GB RAM.

Results in Figure 1 correspond to the average aggregated time for the generation of all recommendation lists concerning a validation fold. For higher values of N, the time difference between *GSMI* and $Top - N$ could increase, but such analysis is not useful in real-world scenarios because N values are never big in practice. Therefore, for realistic values of N, *GSMI* scales well and its average running times per validation fold are only slightly bigger than those obtained with $Top - N$. In spite of that, the time difference for generating a single recommendation list with both methods is irrelevant. Given that in real-world systems recommendation lists are generated once at a time via the interaction with users, and *GSMI* yields better utility results, it is thus a feasible alternative.

5.2 Relation between *GSMI* and Recommendations Diversity

To investigate the relation between *GSMI* and recommendations diversity, we computed the ILD metric, as in Equation 7, for the MovieLens datasets. Movie similarities were computed via the Jaccard's similarity over their corresponding genres, as properly indicated in the datasets. We did not perform such experiments over the Jester 1 dataset because it does not provide any content-based information. Results for both $Top - N$ and *GSMI* with respect to predictors PureSVD and NNCosNgbr are summarized in Table 3.

According to our experiments, *GSMI* and $Top - N$ do not generate statistically significant diversity differences in recommendations. This is an evidence that *GSMI* is not likely to hurt recommendations diversity – at least when compared to $Top - N$. It also indicates that considering mutual influence via pairwise scores θ does not imply in either redundant or monotonous recommendations.

6 Conclusions and Future Work

In this work, we investigated how items can interfere with their own utility in recommendation scenarios. We stated that there is a mutual influence among

Table 3. *ILD* results for different values of N using PureSVD and NNCosNgbr as predictors. For each $GSMI/Top-N$ pair, we performed a t-test over each dataset. With a 95% confidence level, none of the results are statistically different.

Predictor	Method	MovieLens 100K	MovieLens 1M
PureSVD	$Top-5$	0.8557	0.7459
	$GSMI-5$	0.8544	0.7557
	$Top-10$	0.8619	0.7591
	$GSMI-10$	0.8615	0.7660
	$Top-20$	0.8645	0.7666
	$GSMI-20$	0.8643	0.7696
NNCosNgbr	$Top-5$	0.8649	0.7617
	$GSMI-5$	0.8652	0.7636
	$Top-10$	0.8649	0.7694
	$GSMI-10$	0.8652	0.7709
	$Top-20$	0.8649	0.7712
	$GSMI-20$	0.8652	0.7727

them that increases their utilities when simutaneously selected. It is thus possible to take advantage of these mutual influences to improve recommendation systems. The main intuition behind this project is that not only individual features matter in decision-making processes: the presented set of alternatives as a whole also plays an important role on it.

We proposed a means of computing such mutual influence, pairwise scores θ, and an algorithm that incorporates it to improve the recommendation of items, *GSMI*. To analyse mutual influence as an isolated evidence, we compared *GSMI* with $Top-N$, an item selection technique that does not rely upon any type of signal but sorted individual items scores. These individual scores, namely ϕ, were generated by two different state of the art predictors: PureSVD and NNCosNgbr [6].

We showed that for three distinct datasets – MovieLens 100K, MovieLens 1M, and Jester 1 – *GSMI* consistently generated recommendation lists with higher utility measures, i.e. higher average ratings [14], when compared to $Top-N$. We also present evidence that *GSMI* is easily scalable and therefore useful for real-world scenarios. Finally, we show that this algorithm does not seem to hurt recommendations diversity.

Given that we show that mutual influence is an important evidence for recommender systems, we intend to develop LTR algorithms that embed it in a near future. We also want to investigate exact solutions for our optimization problem (Equation 6) that can be feasible in practice, as well as ways of overcoming data sparsity as a means to compute scores θ in a more stable fashion. A thorough assessment of which of all studied algorithms leads to less performance variations is also planned as future work. Finally, we plan on implementing different baselines that also consider some type of influence or dependency among items, such as Latent Structured Ranking [10] or the mean-variance ranking model proposed by Wang [7].

Acknowledgements. This work was partially sponsored the Brazilian National Institute of Science and Technology for the Web (grant MCT/CNPq 573871/2008-6), and by the authors' individual grants and scholarships from CAPES and CNPq. The first author is also thankful for fruitful discussions with Google Software Engineer Davi M. J. Barbosa.

References

1. Toffler, A.: Future Shock. Random House (1970)
2. Adomavicius, G., Tuzhilin, A.: Towards the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering 17(6), 734–749 (2005)
3. Tversky, A.: Elimination by aspects: A theory of choice. Psychological Review 79(4), 281–299 (1972)
4. Passos, A., Gael, J.V., Herbrich, R., Paquet, U.: A penny for your thoughts? the value of information in recommendation systems. In: NIPS Workshop on Bayesian Optimization, Experimental Design, and Bandits, pp. 9–14 (2011)
5. Vargas, S., Castells, P.: Rank and relevance in novelty and diversity metrics for recommender systems. In: RecSys., pp. 109–116 (2011)
6. Cremonesi, P., Koren, Y., Turrin, R.: Performance of recommender algorithms on top-n recommendation tasks. In: RecSys., pp. 39–46 (2010)
7. Wang, J.: Mean-variance analysis: A new document ranking theory in information retrieval. In: Boughanem, M., Berrut, C., Mothe, J., Soule-Dupuy, C. (eds.) ECIR 2009. LNCS, vol. 5478, pp. 4–16. Springer, Heidelberg (2009)
8. Ricci, F., Rokach, L., Shapira, B., Kantor, P.B. (eds.): Recommender Systems Handbook. Springer (2011)
9. Xiong, C., Wang, T., Ding, W., Shen, Y., Liu, T.Y.: Relational click prediction for sponsored search. In: WSDM, pp. 493–502 (2012)
10. Weston, J., Blitzer, J.: Latent structured ranking. In: UAI, pp. 903–913 (2012)
11. Papagelis, M., Plexousakis, D.: Qualitative analysis of user-based and item-based prediction algorithms for recommendation agents. Engineering Applications of Artificial Intelligence 18(7), 781–789 (2005)
12. Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: SIGKDD, pp. 426–434 (2008)
13. Nemhauser, G., Wolsey, L.: Integer and combinatorial optimization. Wiley (1988)
14. Breese, J., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: UAI, pp. 43–52 (1998)
15. McNee, S.M., Riedl, J., Konstan, J.A.: Being accurate is not enough: how accuracy metrics have hurt recommender systems. In: SIGCHI, pp. 1097–1101 (2006)
16. Zhang, M., Hurley, N.: Avoiding monotony: improving the diversity of recommendation lists. In: RecSys., pp. 123–130 (2008)
17. Bell, R., Koren, Y.: Lessons from the netflix prize challenge. ACM SIGKDD Explorations Newsletter 9(2) (2007)