Enhancing Matrix Factorization Through Initialization for Implicit Feedback Databases

Balázs Hidasi
Gravity R&D Ltd.
balazs.hidasi@gravityrd.com
Budapest University of Technology and Economics
hidasi@tmit.bme.hu

Domonkos Tikk
Gravity R&D Ltd.
domonkos.tikk@gravityrd.com

ABSTRACT

The implicit feedback based recommendation problem—when only the user history is available but there are no ratings—is a much harder task than the explicit feedback based recommendation problem, due to the inherent uncertainty of the interpretation of such user feedbacks. Still, this practically important recommendation task received less attention and therefore there are only a few common implicit feedback based algorithms and benchmark datasets. This paper focuses on a common matrix factorization method for the implicit problem and investigates if recommendation performance can be improved by appropriate initialization of the feature vectors before training. We present a general initialization framework that preserves the similarity between entities (users/items) when creating the initial feature vectors, where similarity is defined using e.g. context or metadata information. We demonstrate how the proposed initialization framework can be coupled with MF algorithms. The efficiency of the initialization is evaluated using various context and metadata based similarity concepts on two implicit variants of the MovieLens 10M dataset and one real life implicit database. It is shown that performance gain can attain 10% improvement in recall@50 and in AUC@50.

Author Keywords

Recommender systems, Implicit feedback, Initialization, Similarity, Context information

ACM Classification Keywords

I.2.6 [Artificial Intelligence]: Learning - Parameter Learning

General Terms

Algorithms, Experimentation

CaRR 2012, February 14, 2012, Lisbon, Portugal.

INTRODUCTION

Recommender systems identify specific content that matches users' personal interests within huge content collections. The relevance of an item (the unit of content) with respect to a user is predicted by recommender algorithms; items with the highest prediction scores are displayed to the user.

A typical classification [5] divides recommender algorithms into two main approaches: the content based filtering (CBF) and the collaborative filtering (CF). Content based filtering algorithms use user metadata (e.g. demographic data) and item metadata (e.g. author, genre, etc.) and try to predict the preference of the user based on these attributes. In contrast, collaborative filtering methods do not use metadata, but only data of user–item interactions. Depending on the nature of the interactions, algorithms can also be classified into explicit and implicit feedback based methods. In the former case, users provide explicit information on their item preferences, typically in form of user ratings. In the latter case, however, users express their item preferences only implicitly, as they regularly use an online system; most typical implicit feedback types are viewing and purchasing. Obviously, implicit feedback data is less reliable as we will detail later. CF algorithms proved to be more accurate than CBF methods, if sufficient preference data is available; for a quantification of sufficiency, see e.g. [11]. If this does not hold, the so-called cold-start problem occurs.

In the last few years, latent factor based CF methods gained enhanced popularity, because they were found to be much more accurate in the Netflix Prize, a community contest launched in late 2006 that provided for a long term the largest explicit benchmark dataset (100M ratings) [2]. Latent factor methods build generalized models that intend to capture user preference. These algorithms represent each user and item as a feature vector and the rating of user u for item i is predicted as the scalar product of these vectors. Different matrix factorization (MF) methods are often used to compute these vectors, which approximate the partially known rating matrix using alternating least squares (ALS) [1], gradient descent method [19], coordinate descent method [12], conjugate gradient method [21], singular value decomposition [8],

or a probabilistic framework [16].

CF methods are able to provide accurate recommendations if enough feedback is available. In a few application areas, such as movie rental, travel applications, video streaming, users have motivation to provide ratings to obtain better service. In general, however, users of online e-commerce shops or services do not tend to provide ratings on items even if such an option is available, because (1) when purchasing they have no information on their satisfaction rate (2) they are not motivated to return later to the system to do so. In such cases, user preferences can only be inferred by interpreting user actions (also called *events*). For instance, a recommender system may consider the navigation to a particular product page as an implicit sign of preference for the item shown on that page [15]. The user history specific to items are thus considered as implicit feedback on user taste. Note that the interpretation of implicit feedback data may not necessarily reflect user satisfaction which makes the implicit feedback based preference modeling a much harder task. For instance, a purchased item could be disappointing for the user, so it might not mean a positive feedback. We can neither interpret missing navigational or purchase information as negative feedback, that is such, information is not available.

Despite its practical importance, this harder but more realistic task have been studied less. The implicit alternating least squares (iALS) method [6] is considered the seminal work in this area, which also cast the problem to a latent factor model and keeps its computational efficiency given implicit user feedback using "the implicit trick" (see Section Related work).

In this paper we examine the importance of the initialization of this iALS algorithm. We show that if the usual random or zero initialization is replaced by a similarity based version, the model performance improves significantly. We propose a matrix factorization based initialization method which integrates additional, possibly external, information sources—we performed experiments with context and metadata—to calculate the initial weights in the model. The proposed initialization methodology can be combined with arbitrary implicit feedback matrix factorization method (see e.g. [12], [21]).

The main contributions of this papers are: (1) along a simple idea we propose a general concept of initializing matrix factorization methods; (2) we propose a novel method (SimFactor) that enables to improve the quality of the initial vectors; (3) we run experiments with a large variety of initialization settings using different types of additional information sources on MovieLens 10M and on a real life implicit feedback grocery datasets.

The rest of the paper is organized as follows. *Related work* describes the iALS algorithm and presents currently used initialization approaches. The concept of our initialization methods is described in *Method*. Here we also describes the SimFactor algorithm that can approximate the similarities between entities efficiently using feature vectors. In *Results* we present the results of our experiments with different initialization methods. Finally *Conclusion* sums up this work.

RELATED WORK

We first present the iALS algorithm [6] as our experiments revolve around this matrix factorization algorithm.

We will use the following notation in this work: N is number of users, M is number of items, K denotes the number of features, R is rating matrix, P and Q are user and item feature matrices.

The implicit task is solved in iALS by a weighted matrix factorization. Instead of the R matrix, an $R^{(p)}$ (preference) matrix is constructed in a way that the (u, i) element of the matrix is 1 only if user u has at least one event on item i, otherwise 0. It is important to note that this $R^{(p)}$ matrix is dense unlike the R matrix of the explicit problem (but $R^{(p)}$ contains a lot of zero elements). A W weight matrix is also created: if the (u, i) element of $R^{(p)}$ is 0 then the (u, i) element of W is 1, otherwise it is greater than 1. The specific value can be computed based on the number and type of events between user u and item i. The weights can be computed in several ways. This decomposition of the R matrix can be interpreted as that the presence of an event (e.g. *buy*) provide more reliable information on the user preference than the absence of an event. In other words, we can be more confident in our assumption (buy = like) in case of positive implicit feedbacks. We model this by assigning (much) greater weight to positive implicit feedback than to negative one.

Since the $R^{(p)}$ matrix is dense, any algorithm that scales with the number of ratings can not solve this problem efficiently because the number of "implicit ratings" is $N \times M$. Given that the density of the rating matrix is usually below 1 %, the naive implementation would require several orders of magnitude more computation time compared to the explicit case, which scales linearly with the number of ratings.

In [6], an "implicit trick" for ALS is proposed to brake down the computational time. ALS approximates the matrix R as the product of two lower rank matrices, $R \approx PQ$, and performs a series of weighted linear regressions. First, matrices P and Q are initialized with random values. Then we fix matrix Q and compute each column of matrix P using weighted linear regression (minimizing $(R^{(p)}_{u,\bullet} - (P_{\bullet,u})^T Q) W^{(u)} (R^{(p)}_{u,\bullet} - (P_{\bullet,u})^T Q)^T$, where $W^{(u)}$ is a $M \times M$ diagonal matrix and $W^{(u)}_{i,i} = W_{u,i}$). Then, matrix P is fixed and the columns of Q are computed analogously.

The bottleneck in computing a column of P comes from the computation of the $QW^{(u)}Q^T$ that is naively done in $O(K^2 M)$. However, $QW^{(u)}Q^T$ can be rewritten as $QQ^T + Q(W^{(u)} - I)Q^T$ (I is the identity matrix), from

which QQ^T can be precalculated. Because $(W^{(u)} - I)$ has only a few non-zero elements, the cost of computing $Q(W^{(u)} - I)Q^T$ is only $O(K^2 n_u)$ where n_u is the number of non-zero element in the u^{th} row of $R^{(p)}$. Hence, the total cost (all N column) of the computation of P is proportional with the number of positive implicit feedback instead of number of all entries in the rating matrix.

The importance of proper initialization was recognized for some matrix factorization algorithms like the Non-negative Matrix Factorization (NMF). It was shown in [17] that a good initialization can improve the speed and accuracy of the algorithms, as it can produce faster convergence to an improved local minimum. The rich literature of NMF initialization includes centroid methods [10], spherical k-means clustering methods [23, 24] that provides low rank representation [4], SVD [3] and sum of randomly selected feature vectors [10]. It is common in all of these methods that they use the same data for initialization and for training the NMF.

In collaborative filtering algorithms, feature weights are typically initialized with small random weights [13, 20]. Certain works report on some parameterized randomization, drawing the random numbers from a normal distribution [22], or defining adjustable lower and upper bounds separately for the item and user weights [20]. To the best of our knowledge, more sophisticated initialization approaches, using external data sources have not been proposed so far.

METHOD

Most of the MF methods are iterative algorithms that are started from a random point: the item and user feature matrices are initialized randomly. After some iterations these methods converge to a local optimum that depends on the starting point. Our hypothesis is that appropriate initialization of feature vectors yields that MF methods will produce more accurate feature vectors and therefore give more accurate predictions.

When investigating the feature vectors of accurate MF models, one can observe similar items (e.g. items belonging to the same product category, or episodes of a movie series) have similar item feature vectors. This suggest that similarity-based initialization of feature vectors may result in more appropriate models. The calculation of the initial item and user feature vectors should be obviously aligned with the learning algorithm applied. To do this, first we have to define the similarity between entities (items, users), which depends on the similarity function and on the available item, user and transactional data. In this paper we use cosine similarity as similarity function for simplicity, but this can be substituted by any other similarity metric. As for the available data, we can make use of any of the following data in our experiments:

- *Item metadata vectors:* let us consider an indexed set of metadata tags, which contains all the possible tags that occur in item metadata (can be textual or categorical). The item metadata vector contains a non-zero value in the i^{th} position if the i^{th} tag occurs in item's metadata. One can apply various weighting schemes (e.g.: tfidf) to determine the elements of the vectors.
- *User/Item event vectors:* a user event vector of M length indicates with a non-zero values for which item the user has at least one event (analog for items).
- *User/Item context state vectors:* let us define the set of context states (C) as the possible combination of values of context variable. Here we consider only categorical context variables with finite range. For instance if we take *seasonality* as context, and a season is a week and time bands are days, then we have 7 context states. When more than one context variable is used then the context states are the Descartes-product of individual context values. I.e. if additionally we store in another context variable if the purchase was made online or offline, then we have 14 context states. Then the i^{th} element in the user context state vector is non-zero if the user has at least one event in the i^{th} context state (analog for items).
- *User/Item context-event vectors:* the user context-event vectors have length $C \cdot M$; each coordinate represents whether user has events on the given item in the given context state (analog for items).

Remark that most of these vectors are typically very sparse, except context state vectors with few context variables. Note that in each of the above cases, one has several choices in creating the item/user description vectors from the raw data: vectors may be binary, may contain event counters, furthermore one may apply normalization or a weighting scheme.

We assemble a matrix, D, from the appropriate input vectors (row-wise), which is referred to as the description of the items (D_I) or users (D_U). For this we select an arbitrary but single data source from the above options; e.g., we use the item context state data vectors to form D. In order to make use of the description as initial weights in a matrix factorization method, one should compress them to be compliant with the feature size of the MF model. This can be performed by any dimension reduction techniques like PCA [7], matrix factorization, auto-associative MLP [9], etc. These methods minimize the information loss at the compression and simultaneously perform noise reduction.

In this paper we use two methods for compression. The first is a simple matrix factorization, the weighted ALS, that minimizes the weighted squared error of the predictions by fixing one of the feature matrices and computing the rows of the other by using weighted linear regression. When factorizing item description, we only keep the item feature matrix after the factorization process (analog for user description), which is then readily used as initial feature vectors in the iALS algorithm.

Our starting hypothesis was that description vectors characterize well the similarities between entities. Therefore the relation of similarities (e.g. ratios, order, etc.) between original description vectors should be carried over to the compressed description vectors. Next we introduce the SimFactor compression method that is able to preserve the relations between the original similarities in a much larger extent than ALS.

SIMFACTOR ALGORITHM

As noted above, standard dimension reduction techniques may distort the system of similarities between the entities. One could design a method that keeps this property by starting from the similarity matrix of the users/items. The problem with such an approach is that it requires the precomputation of the entire similarity matrix, which is computationally very inefficient. Further, this solution does not scale well, because the matrix has to be stored in memory the sake of efficient computation. According to our test, even when sparse data structures are used for storing similarities, the calculation of the similarity matrix takes a considerable amount of time, when N or M is large.

SimFactor is a simple algorithm that compresses the description of the items while preserves the relations between the original similarities as much as possible. This method only works for similarity metrics that can be computed via the scalar product of two (transformed) vectors. The most commonly used metrics in recommendation systems like cosine similarity, adjusted cosine similarity or Pearson correlation [18] can be computed in this way. As for cosine similarity, one needs to ℓ_2-normalize the input vectors then their scalar product will be the same as the cosine similarity between the original vectors. The pseudocode for SimFactor is described in Algorithm 1 (see also Figure 1).

Algorithm 1 SimFactor

Input: D matrix that contains the item or the user description
Output: F matrix that contains the feature vectors of the items or users
procedure SimFactor(D)
1: $D' \leftarrow$ Transform(D)
2: $< X, Y > \leftarrow$ FactorizeMatrix(D')
3: $Z \leftarrow Y^T Y$
4: $< U, \Lambda > \leftarrow$ EigenDecomposition(Z)
5: $F \leftarrow$ matrix of $N_{entities} \times N_{entities}$ size
6: **for** $i = 1, \ldots, N_{entities}$ **do**
7: $\quad F_i = \sqrt{\Lambda_{i,i}} X_i U$
8: **end for**
9: **return** F
end procedure

SimFactor starts with the appropriate transformation of the description matrix (line 1; e.g. ℓ_2-normalization when using cosine similarity). Next in line 2, a matrix factorization is applied on the description, but in contrast to the method described above, both low rank matrices are kept. For the matrix factorization, arbitrary MF method can be used. Here, we applied weighted ALS.

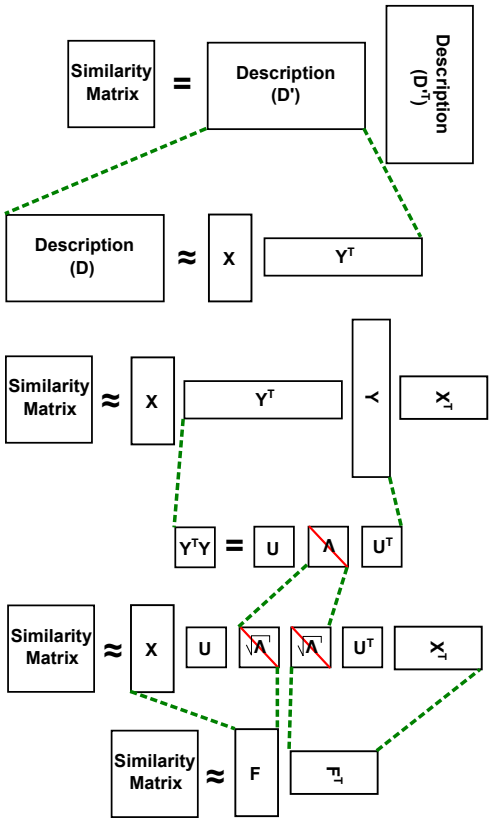

Figure 1. Concept of the matrix transformations in SimFactor

The steps performed between lines 3 and 8 are also depicted on Figure 1. The matrix of similarities (S) is the product of the transformed description matrix and its transpose ($S = D'D'^T$), while the factor matrices (output of the MF method in line 2) approximate the transformed description matrix ($D' \approx XY^T$). Therefore the similarity matrix can be approximated by $S \approx XY^TYX^T$. Y^TY is a $K \times K$ symmetric (non-singular) matrix, thus its eigen-decomposition always exists in the following form: $Y^TY = U\Lambda U^T$. U and Λ are $K \times K$ matrices, the earlier contains the eigenvectors the latter is singular and has the eigenvalues in its diagonal. Λ can be written as the product of two identical matrices denoted with $\sqrt{\Lambda}$. $\sqrt{\Lambda}$ is also diagonal and contains the square roots of the eigenvalues. At this point our approximation of the similarity matrix looks like this:

$S \approx XU\sqrt{\Lambda}\sqrt{\Lambda}U^T X^T$. Introducing the $N_{entities} \times K$ matrix $F = XU\sqrt{\Lambda}$, this can be rewritten $S \approx FF^T$.

In F, every row is a feature vector for an entity and the scalar product of the i^{th} and j^{th} rows approximates the similarity between the corresponding entities. This way SimFactor produces low-rank feature vectors that try to preserve the original similarity values. We can use these feature vectors as the initial features in the iALS algorithm. The complexity of SimFactor—in addition to the initial transformation and matrix factorization—is $O(K^2N + K^3 + K^2M)$, where the subsequent terms correspond to the calculation Y^TY, finding the eigendecomposition and calculating $F = XU\sqrt{\Lambda}$, respectively. We found that, in practice, this cost is negligible compared to the cost of the initial matrix factorization in line 2.

RESULTS

We used two datasets for experimentation. The first dataset is the MovieLens 10M [14] that was transformed into an implicit feedback dataset. We used two different transformations: (1) keeping only the 5 star ratings and (2) keeping ratings with values 4 and above as positive feedbacks. We used the last 20 days for testing (from 08/12/2008) and the rest for training. As the available metadata is not sufficient we did not use the metadata for initialization with the MovieLens database. The second dataset contains purchase events of an online grocery store. The number of events is slightly above 6.24 million targeted on 17,000 items (of them 14,000 has at least one event). We used all but the last month's data for training and the last month for testing.

We used various data sources when creating the description matrix (see details in *Method*). For context information, we chose seasonality because the time stamp is available in almost every dataset. On seasonality we mean that we define periodicity and divide it into smaller time intervals called *time bands*. For example, hours of a day can be time bands of a day, or the weekdays and the weekend can be time bands of a week. Note that the length of the time bands does not have to be identical. The context of an event is the time band in which it happened. We used different periods and time bands and kept only the best results.

Our first experiment compared weighted ALS and SimFactor to characterize their similarity preserving capability. We draw randomly 2 times 100 000 entity pairs, calculated the original similarity values and measured the RMSE (root mean square error) of the similarity value prediction as well as the improvement in the relation of the pairs. Next, we characterized difference between similarity preserving using *ratio improvement* calculated as:

$$\frac{\sum_{i_1,i_2=1}^{100\,000}\left(s_{i_1}/s_{i_2} - s''_{i_1}/s''_{i_2}\right)^2}{\sum_{i_1,i_2=1}^{100\,000}\left(s_{i_1}/s_{i_2} - s'_{i_1}/s'_{i_2}\right)^2} - 1$$

where prime denotes the prediction of SimFactor and double prime denotes the prediction of ALS.

The results in Table 1 show that SimFactor was more accurate in every experiment. The improvement varies between 10–50%. In addition to better accuracy, SimFactor also preserves the original relations of the similarities better than the weighted ALS. The performance metrics depends greatly on the description matrix.

We used recall@50 as primary evaluation metric for the main experiments, which is the fraction of the proportion of relevant items among the top50 (ranked) recommendations for the user and the user's events in the test set. Items considered relevant to a user if the user has at least one event on that item in the test set. Recall@50 is an important measure in practical applications as the user usually sees maximum the top few items (50 items could be seen on multiple pages during a visit). We also present the area under the precision-recall curve (from $1 \ldots 50$) as a secondary metric.

The experiments started by optimizing the hyperparameters of an iALS algorithm. We used low-factor models as they can be used in practice as well. Then we run multiple experiments with different random initializations and chose the best result as the baseline. We used weighted ALS and SimFactor (that also uses a weighted ALS as its first step) to create the initial feature. Note that since iALS is an alternating method that discards the results of previous computations when calculating the feature vectors. we can not initialize both item and user features at once as one of them will be discarded in the first step. We ran multiple experiments for each input data type for the initialization and kept only the best one per input data type.

Table 2 sums the results of our experiments. The results on the grocery database are the most relevant as that database is originally implicit feedback based. The result clearly show the superiority of the SimFactor method over standard factorization as the top performing initializations used SimFactor. We observed the same when experimenting on MovieLens (5star), but on the MovieLens (4star+) SimFactor is less dominant. This can be argued by the additional noise being present in the MovieLens (4star+) datasets (note that "implicitization" was performed in a non-personal way and user may use different ratings scales).

We want to point out that the top performing methods on every dataset use contextual information for initialization. Both context state and context-event information is used amongst them, but on the grocery and MovieLens (5star) data, the context state based methods are the dominant. This suggest that context information, like seasonality, can efficiently discriminate between entities, and this can be exploited in weight initialization: Users have routines and people with similar routines are similar and might have similar taste. Similarly, different item types are typically consumed in different time

Table 1. Accuracy of the similarity prediction

Input data	Method	RMSE	RMSE Improvement	Ratio Improvement
MovieLens 10M (5 star ratings)				
Item context state	ALS	0.2055	45.79%	70.75%
	SimFactor	0.1114		
User context state	ALS	0.3345	11.66%	79.60%
	SimFactor	0.2955		
Item context-event	ALS	0.0568	22.68%	2.46%
	SimFactor	0.0439		
User context-event	ALS	0.1631	33.13%	64.43%
	SimFactor	0.1091		
Item event data	ALS	0.0593	23.98%	87.47%
	SimFactor	0.0451		
User event data	ALS	0.1285	32.03%	86.46%
	SimFactor	0.0874		
MovieLens 10M (4 star ratings and above)				
Item context state	ALS	0.2219	31.55%	74.58%
	SimFactor	0.1519		
User context state	ALS	0.2469	12.40%	61.29%
	SimFactor	0.2163		
Item context-event	ALS	0.0317	35.28%	31.82%
	SimFactor	0.0205		
User context-event	ALS	0.0835	34.64%	83.19%
	SimFactor	0.0546		
Item event data	ALS	0.0658	2.84%	73.53%
	SimFactor	0.0639		
User event data	ALS	0.1803	14.51%	37.39%
	SimFactor	0.1542		
grocery dataset				
Item context state	ALS	0.3254	52.36%	81.74%
	SimFactor	0.1550		
User context state	ALS	0.1178	10.81%	18.69%
	SimFactor	0.1050		
Item context-event	ALS	0.0314	16.86%	71.80%
	SimFactor	0.0261		
User context-event	ALS	0.1518	26.22%	62.55%
	SimFactor	0.1120		
Item event data	ALS	0.0492	13.39%	9.61%
	SimFactor	0.0427		
User event data	ALS	0.1930	48.70%	64.08%
	SimFactor	0.0990		
Item metadata	ALS	0.2709	12.38%	38.39%
	SimFactor	0.2374		

bands; for example adult programs mostly viewed late night. The distribution of the events for an entity in the time bands seems to be an efficient descriptor.

The largest improvement for the grocery dataset is 5.71% in the recall and the fifth best method improves it by 4.04%. Considering that the cost of the initialization is small and that this improvement can be translated into increased profit, we believe that the initialization of the algorithm is beneficial.

CONCLUSION

In this paper we proposed a general framework for initializing MF algorithms. Our hypothesis was that initializing item and user models with weights that reflect the similarity between entities will improve algorithm performance when compared to starting from a random state. This method also allows us to easily incorporate additional information like context or metadata information into the MF framework.

Our proposed SimFactor algorithm can efficiently implement the general idea. We found that SimFactor preserves the original similarities and their relations better than other MF methods which makes this method more suitable for our goals. The additional complexity of SimFactor is its negligible additional cost when compared to an arbitrary MF method.

Using different data sources to compute similarity can greatly affect the performance gain observed with initialization. We found that the greatest improvement can be achieved by using the context information (we experimented with seasonality). Context separates the entities more appropriately than any other information

Table 2. Results of the top10 performing initialization methods on both datasets

Data type	Method	Recall@50	Improvement to baseline	AUC@50	Improvement to baseline
MovieLens 10M (5 star ratings)					
Item context state	SimFactor	0.1413	10.00%	$1.4512 * 10^{-3}$	9.85%
User context state	SimFactor	0.1403	9.17%	$1.4000 * 10^{-3}$	5.98%
Item context-event	SimFactor	0.1403	9.17%	$1.3526 * 10^{-3}$	2.39%
Item context-event	MF	0.1403	9.17%	$1.3377 * 10^{-3}$	1.26%
Item context state	MF	0.1403	9.17%	$1.4596 * 10^{-3}$	10.49%
Item event data	SimFactor	0.1392	8.33%	$1.3754 * 10^{-3}$	4.11%
User context-event	MF	0.1392	8.33%	$1.3754 * 10^{-3}$	4.11%
User context state	MF	0.1392	8.33%	$1.4006 * 10^{-3}$	6.02%
User context state	SimFactor	0.1382	7.50%	$1.3899 * 10^{-3}$	5.21%
Item event data	MF	0.1381	7.50%	$1.3725 * 10^{-3}$	3.89%
Random initialization (baseline)		0.1285	N/A	$1.3211 * 10^{-3}$	N/A
MovieLens 10M (4 star ratings and above)					
User context-event	SimFactor	0.08636	5.67%	$1.1612 * 10^{-3}$	5.24%
Item context-event	MF	0.08574	4.91%	$1.1537 * 10^{-3}$	4.56%
User context-event	MF	0.08559	4.73%	$1.1672 * 10^{-3}$	5.79%
Item context state	MF	0.08528	4.35%	$1.1671 * 10^{-3}$	5.78%
Item context-event	SimFactor	0.08512	4.16%	$1.1486 * 10^{-3}$	4.09%
User context state	MF	0.08497	3.97%	$1.1390 * 10^{-3}$	3.23%
User context state	SimFactor	0.08466	3.59%	$1.1345 * 10^{-3}$	2.82%
User event data	MF	0.08451	3.40%	$1.1670 * 10^{-3}$	5.77%
Item event data	SimFactor	0.08435	3.21%	$1.1265 * 10^{-3}$	2.10%
User event data	SimFactor	0.08435	3.21%	$1.1294 * 10^{-3}$	2.35%
Random initialization (baseline)		0.08172	N/A	$1.1034 * 10^{-3}$	N/A
grocery dataset					
User context state	SimFactor	0.1508	5.71%	$1.0692 * 10^{-2}$	10.19%
User context state	MF	0.1496	4.88%	$1.0675 * 10^{-2}$	10.01%
User context-event	SimFactor	0.1488	4.30%	$1.0367 * 10^{-2}$	6.83%
User event data	SimFactor	0.1485	4.12%	$1.0626 * 10^{-2}$	9.50%
User context-event	MF	0.1484	4.04%	$1.0408 * 10^{-2}$	7.25%
Item event data	SimFactor	0.1474	3.29%	$1.0169 * 10^{-2}$	4.79%
Item metadata	MF	0.1472	3.15%	$1.0319 * 10^{-2}$	6.34%
Item context-event	MF	0.1469	2.97%	$1.0280 * 10^{-2}$	5.94%
Item context state	MF	0.1465	2.67%	$1.0222 * 10^{-2}$	5.35%
Item context state	SimFactor	0.1464	2.64%	$1.0199 * 10^{-2}$	5.10%
Random initialization (baseline)		0.1427	N/A	$0.9809 * 10^{-2}$	N/A

we examined and therefore it can not be neglected. An additional 4–6% improvement compared to the random initialization could be achieved through appropriate initialization on a real life dataset using SimFactor and only 3–3.5% when using MF. The similarity preserving property of the SimFactor can be a disadvantage when the description matrix is too noisy, that is the description matrix does not capture item or user similarities.

Future work includes experimentation with other similarity metrics (e.g. Pearson correlation instead of cosine similarity) and the combination of various context information into a single similarity measure.

The utility, that is, whether 4–6% improvement worths the time of the initialization depends on many factors. The main cost factor is the MF step in SimFactor whose complexity greatly depends on the description data and the parameters of the factorization. The number of context states is much lower than the number of items/users and therefore the description matrix is relatively small, thus the initial MF step is relatively fast. Furthermore, the context state based initialization turned out to be the most efficient one, which suggest that context based initialization has positive utility in real-world recommender applications.

REFERENCES

1. Bell, R. M., and Koren, Y. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proc of. ICDM-07, 7th IEEE Int. Conf. on Data Mining* (Omaha, Nebraska, USA, 2007), 43–52.
2. Bennett, J., and Lanning, S. The Netflix Prize. In *Proc. of KDD Cup Workshop at SIGKDD-07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining* (San Jose, California, USA, 2007), 3–6.
3. Boutsidis, C., and Gallopoulos, E. Svd based initialization: A head start for nonnegative matrix factorization, 2007.
4. Dhillon, I. S., and Modha, D. S. Concept decompositions for large sparse text data using clustering. In *Machine Learning* (2000), 143–175.
5. Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems 22*, 1 (2004), 5–53.
6. Hu, Y., Koren, Y., and Volinsky, C. Collaborative filtering for implicit feedback datasets. In *Proc. of ICDM-08, 8th IEEE Int. Conf. on Data Mining* (Pisa, Italy, December 15–19, 2008), 263–272.
7. Jolliffe, I. *Principal Component Analysis*. Springer Verlag, 1986.
8. Koren, Y. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proc. of the 14th ACM Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD'08)* (Las Vegas, Nevada, USA, 2008), 426–434.
9. Kramer, M. A. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal 37*, 2 (1991), 233–243.
10. Langville, A. N., Meyer, C. D., and Albright, R. Initializations for the nonnegative matrix factorization, 2006.
11. Pilászy, I., and Tikk, D. Recommending new movies: Even a few ratings are more valuable than metadata. In *Proc. of the 3rd ACM Conf. on Recommender Systems (Recsys'09)*, ACM (New York, NY, USA, 2009), 93–100.
12. Pilászy, I., Zibriczky, D., and Tikk, D. Fast ALS-based matrix factorization for explicit and implicit feedback datasets. In *Proc. of the 4th ACM Conf. on Recommender Systems (RecSys'10)*, ACM (Barcelona, Spain, 2010), 71–78.
13. Rendle, S., and Schmidt-Thieme, L. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Proc. of RecSys-08: ACM Conf. on Recommender Systems*, ACM (New York, NY, USA, 2008), 251–258.
14. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. Grouplens: an open architecture for collaborative filtering of netnews. In *Proc. of CSCW-94, 4th ACM Conf. on Computer Supported Cooperative Work* (Chapel Hill, North Carolina, USA, 1994), 175–186.
15. Ricci, F., Rokach, L., and Shapira, B. Introduction to recommender systems handbook. In *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds., Artificial Intelligence. Springer US, 2011, 1–35.
16. Salakhutdinov, R., and Mnih, A. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems 20*, J. C. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. MIT Press, Cambridge, Massachusetts, USA, 2008.
17. Smilde, A., Bro, R., and Geladi, P. *Multi-way Analysis*. Wiley, West Sussex, England, 2004.
18. Snedecor, G. W., and Cochran, W. G. *Statistical Methods*, 7th ed. Iowa State University Press, 1980.
19. Takács, G., Pilászy, I., Németh, B., and Tikk, D. Major components of the gravity recommendation system. *SIGKDD Explor. Newsl. 9* (December 2007), 80–83.
20. Takács, G., Pilászy, I., Németh, B., and Tikk, D. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research 10* (2009), 623–656.
21. Takács, G., Pilászy, I., and Tikk, D. Applications of the conjugate gradient method for implicit feedback collaborative filtering. In *RecSys'11: Proc. of the 4th ACM Conf. on Recommender Systems* (Chicago, IL, USA, October 23–27, 2011), 297–300.
22. Thai-Nghe, N., Drumond, L., Horváth, T., Krohn-Grimberghe, A., Nanopoulos, A., and Schmidt-Thieme, L. Factorization techniques for predicting student performance. In *Educational Recommender Systems and Technologies: Practices and Challenges (ERSAT 2011)*. IGI Global, 2012, 1–25.
23. Wild, S., Wild, W. S., Curry, J., Dougherty, A., and Betterton, M. Seeding non-negative matrix factorization with the spherical k-means clustering. Tech. rep., 2003.
24. Wild, S. M., Curry, J. H., and Dougherty, A. Improving non-negative matrix factorizations through structured initialization. *Pattern Recognition 37*, 11 (November 2004), 2217–2232.