Journal of Universal Computer Science, vol. 19, no. 12 (2013), 1834-1853
submitted: 20/9/12, accepted: 10/5/13, appeared: 28/6/13 © *J.UCS*

Initializing Matrix Factorization Methods on Implicit Feedback Databases[1]

Balázs Hidasi
(Gravity R&D, Budapest, Hungary
Budapest University of Technology and Economics, Budapest, Hungary
balazs.hidasi@gravityrd.com)

Domonkos Tikk
(Gravity R&D, Budapest, Hungary
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. Recently, implicit feedback problem is being received more attention, as application oriented research gets more attractive within the field. 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. We experiment with various similarity functions, different context and metadata based similarity concepts. The evaluation is performed on two implicit variants of the MovieLens 10M dataset and four real life implicit databases. We show that the initialization significantly improves the performance of the MF algorithms by most ranking measures.

Key Words: Recommender systems, Implicit feedback, Initialization, Similarity, Contextual information
Category: I.2.6

1 Introduction

Recommender systems identify specific contents that match 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 [Herlocker et al., 2004] divides recommender algorithms into two main approaches: the content based filtering (CBF) and the

[1] This is an extended version of the paper published at the CARR 2012 Workshop in Lisbon, Portugal. Here we propose Sim$^{\mathbf{2}}$Factor method, a sophisticated variant of the original SimFactor method, perform experiments with several similarity functions on more benchmark datasets.

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, user preferences is captured seamlessly via user activity. Implicit feedback algorithms use user interactions like viewing and purchasing retrieved e.g. from website usage logs. CF algorithms proved to be more accurate than CBF methods, if sufficient preference data is available; for a quantification of sufficiency, see e.g. [Pilászy and Tikk, 2009]. If this does not hold, the so-called cold-start problem occurs.

In the last few years, latent factor based CF methods became popular, 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) [Bennett and Lanning, 2007]. 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 used to compute these vectors by approximating the partially known rating matrix using alternating least squares (ALS) [Bell and Koren, 2007], gradient descent method [Takács et al., 2007], coordinate descent method [Pilászy et al., 2010], conjugate gradient method [Takács et al., 2011], singular value decomposition [Koren, 2008], or a probabilistic framework [Salakhutdinov and Mnih, 2008].

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 are motivated to provide ratings to get 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) at the purchase 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 [Ricci et al., 2011]. The user history specific to items are thus considered as implicit feedback on user taste. Note that the interpretation of implicit feedback data can be somewhat speculative as such data may not necessarily reflect user satisfaction. 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. Uncertainty of user feedback interpretation makes the implicit feedback based preference modeling a challenging problem.

The implicit alternating least squares (iALS) method [Hu et al., 2008] is considered the seminal work for implicit feedback methods, which cast the problem to a latent factor model and keeps its computational efficiency given implicit user feedback using a proper rewriting of the objective function (see Section Related work). Recently, implicit feedback problems attracted more attention, see e.g. [Jahrer and Töscher, 2012, Shi et al., 2012, Takács and Tikk, 2012].

In this paper we examine the importance of the initialization of the iALS algorithm. We show that if the usual random or zero initialization is replaced by a similarity based one, the performance of the model 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 MF method (see e.g. [Pilászy et al., 2010, Takács et al., 2011]).

The main contributions of this papers are: (1) along a simple idea we propose a general concept for the initialization of 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, TV1, TV2, LastFM and on the 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 describe the SimFactor algorithm that can approximate the similarities between entities (users or items) efficiently using feature vectors. In *Results* we present the results of our experiments with different initialization methods. Finally *Conclusion* sums up this work.

2 Related work

We first present the iALS algorithm [Hu et al., 2008] that is the baseline algorithm in our experiments. 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 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 all elements of $R^{(p)}$ are given, while the

R matrix of the explicit problem is only partially observed. 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 weighting reflects the uncertainty of the interpretation of implicit feedback data: the presence of an event (e.g. *buy*) provides more reliable information on the user preference than its absence. 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 feedbacks than to negative ones.

Explicit feedback algorithms scale linearly with the number of observed ratings in the matrix, and the density of the matrix is usually below 1 %. However, in the implicit case all elements of $R^{(p)}$ are given, therefore the computational complexity of such algorithms is $O(N \times M)$. Given the above density, it means that the naive computation is several orders of magnitude slower compared to the explicit case.

In [Hu et al., 2008], a proper rewriting of the objective function for ALS learning 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 elements 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 the number of all entries in the rating matrix.

The importance of proper initialization was recognized for some matrix factorization algorithms like the Nonnegative Matrix Factorization (NMF). It was shown in [Smilde et al., 2004] 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 [Albright et al., 2006], spherical k-means clustering methods [Wild et al., 2004, Dhillon and Modha, 2000] that provides low rank representation, SVD [Boutsidis and Gallopoulos, 2008] and sum of randomly selected feature vectors [Albright et al., 2006]. Commonly, these methods use the same data for initialization and for training the NMF.

In CF algorithms, feature weights are typically initialized with small random weights [Rendle and Schmidt-Thieme, 2008, Takács et al., 2009]. Certain works report on some parameterized randomization, drawing the random numbers from a normal distribution [Thai-Nghe et al., 2012], or defining adjustable lower and upper bounds separately for the item and user weights [Takács et al., 2009]. To the best of our knowledge, more sophisticated initialization approaches, using external data sources have not been proposed so far.

3 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 an 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 that 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 obviously be aligned with the learning algorithm applied. To do this, first we have to define the similarity between entities (users or items), which depends on the similarity function and on the available item, user or transactional data. In this paper, we experiment with four similarity functions: cosine, scalar product, correlation and weighted correlation. The following data will be used in our experiments to compute similarity:

- *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 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 the frequency, or 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 [Jolliffe, 1986], matrix factorization, auto-associative MLP [Kramer, 1991], 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 is 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 better preserves the relation between the original similarities than ALS.

4 Method

4.1 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 for 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 [Snedecor and Cochran, 1980] 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 of 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 $\textsc{SimFactor}(D)$

1: $D' \leftarrow \textsc{Transform}(D)$
2: $< X, Y > \leftarrow \textsc{FactorizeMatrix}(D')$
3: $Z \leftarrow Y^T Y$
4: $< U, \Lambda > \leftarrow \textsc{EigenDecomposition}(Z)$
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

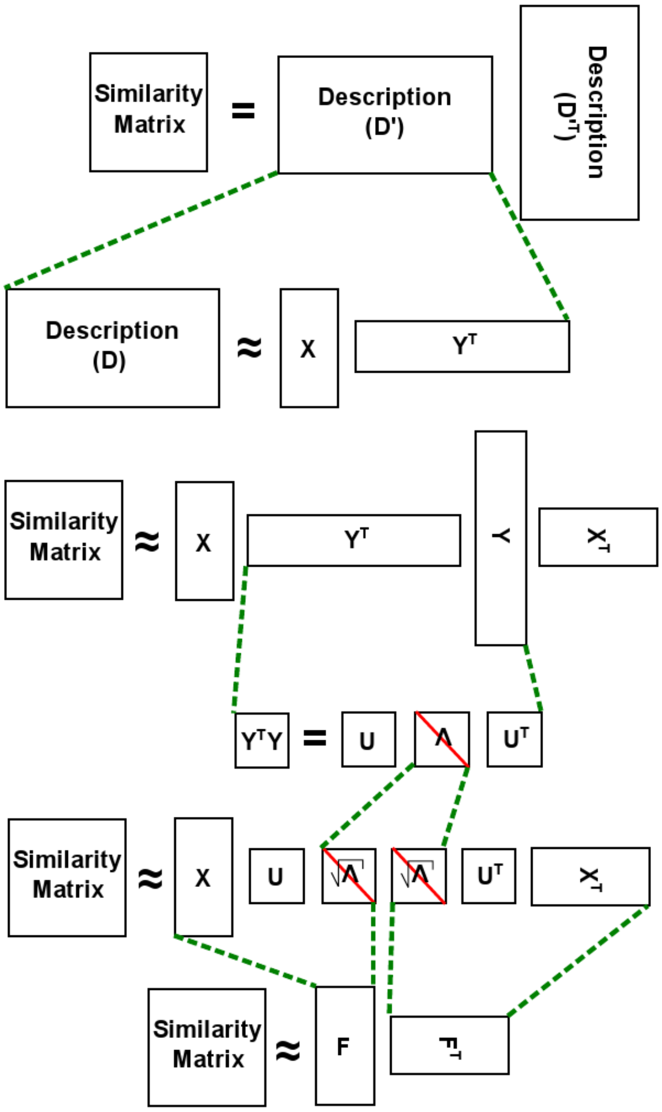

Figure 1: Concept of the matrix transformations in SimFactor

matrix factorization is applied on the description, but in contrast to ALS, both low rank matrices are kept. For the matrix factorization, arbitrary MF method can be used. Here, we applied weighted ALS.

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 $S \approx XY^TYX^T$, where 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. U contains the eigenvectors, Λ is singular and has the eigenvalues in its diagonal. $\Lambda = \sqrt{\Lambda} \cdot \sqrt{\Lambda}$, where $\sqrt{\Lambda}$ is also diagonal and contains the square roots of the eigenvalues. Now we can approximate the similarity matrix as: $S \approx XU\sqrt{\Lambda}\sqrt{\Lambda}U^TX^T$. Introducing the $N_{entities} \times K$ matrix $F = XU\sqrt{\Lambda}$, this can be rewritten as $S \approx FF^T$.

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.

4.2 Similarity-based similarities – Sim^{2}Factor

One can also define the similarity between items based on how similar they are to other items. Given the similarity matrix of the items, similarity measures can be calculated on its rows and these values can be used as item similarity values. This approach can sometimes seize actual similarities between items more precisely, because even if the original similarity values are inaccurate, the system of those values is often more consistent.

As we noted before, the computation of the similarity matrix is often not practical or impossible due to time and memory limitations. Fortunately the SimFactor algorithm can be modified in a way that enables us to approximate the similarity based similarity values without the computation of the similarity matrix. This modification is coined the Sim^{2}Factor. The computational complexity of SimFactor and Sim^{2}Factor are the same. Algorithm 2 describes the method.

Sim^{2}Factor takes description matrix D as an input like SimFactor. The initial steps are the same: first D is transformed in a way that the scalar product of its rows will result in the desired similarity value. In the second step (line 2) an arbitrary matrix factorization is performed. The resulting factor matrices are transformed into the initial feature vectors (F) between lines 3–10. The key of the algorithm is that the similarity based similarity matrix can be calculated as $S' = SS^T = SS = DD^TDD^T$ (S is symmetric). By approximating $D \approx XY^T$ we have $S' \approx XY^TYX^TXY^TYX^T$. Introducing $Z = Y^TY$ ($K \times K$) that can be computed efficiently we get $S' \approx XZX^TXZX^T$. Using $Z' = ZX^T$ we can write $S' \approx XZ'(Z')^TX^T = XZ''X^T$ (Z'' is symmetric). Then the symmetric $K \times K$ sized Z'' is decomposed into product of two $K \times K$ sized matrices in the

Algorithm 2 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)

$D' \leftarrow$ TRANSFORM(D)
$< X, Y > \leftarrow$ FACTORIZEMATRIX(D')
$Z \leftarrow Y^T Y$
$Z' \leftarrow Z X^T$
$Z'' \leftarrow Z'(Z')^T = (ZX^T)(XZ)$
$< U, \Lambda > \leftarrow$ EIGENDECOMPOSITION(Z'')
$F \leftarrow$ matrix of $N_{entities} \times N_{entities}$ size
for $i = 1, \ldots, N_{entities}$ **do**
    $F_i = \sqrt{\Lambda_{i,i}} X_i U$
end for
return F

end procedure

same way as in SimFactor ($Z'' = (U\sqrt{\Lambda})(U\sqrt{\Lambda})^T$). The initial feature matrix is $F = X(U\sqrt{\Lambda})$. The steps of the transformation is also shown on Figure 2.

We note that unlike SimFactor, Sim^{2}Factor does not allow us to use an arbitrary similarity metric between the rows of S. The values of S' will always be the scalar products of the rows of S. This is because the description matrix can not be modified efficiently to force the usage of other metrics in S'. However, with the modification of the description matrix, different similarity metrics can be used to approximate S.

4.3 Similarity metrics

As we mentioned earlier we can use different similarity metrics by transforming the description matrix D. Here we introduce four different metrics and the required transformations on D.

1. **Scalar product:** Scalar product of the rows of the description matrix. No transformation is required on D.

2. **Cosine similarity:** Cosine similarity between vectors. The rows of D must be ℓ_2-normalized.

3. **Correlation:** We found that the correlation values between description vectors are often better at describing similarities between entities. In order to

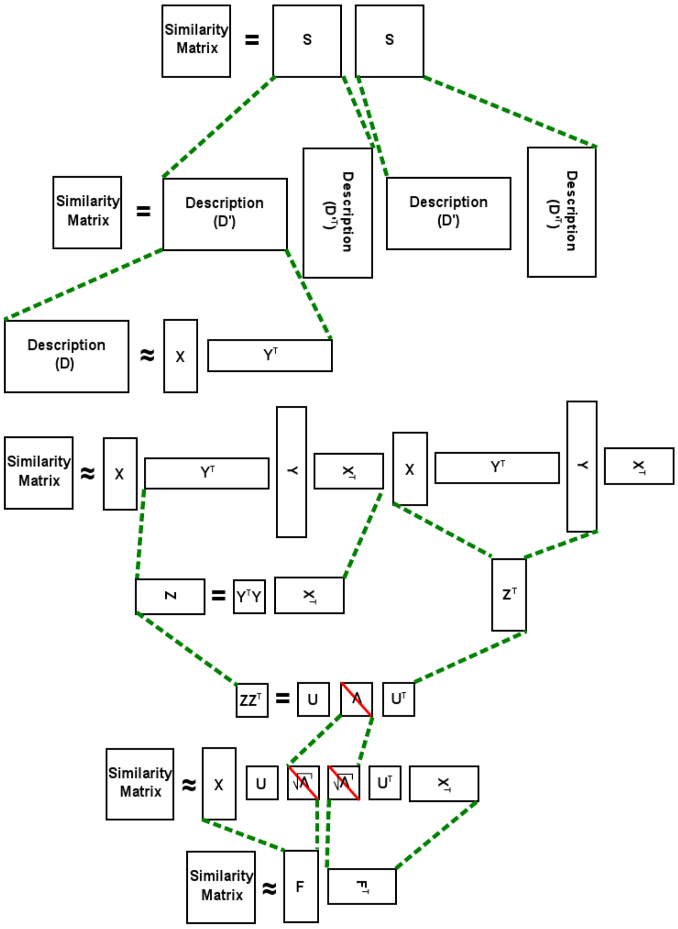

Figure 2: Concept of the matrix transformations in Sim^{2}Factor

use this metric we must use the following transformation:

$$D'_{i,j} = \frac{\left(D_{i,j} - \frac{\sum_{k=1}^{M} D_{i,k}}{M}\right)}{\sqrt{\sum_{k=1}^{M} D_{i,k}^2 - \frac{\left(\sum_{k=1}^{M} D_{i,k}\right)^2}{M}}}.$$

This transformation yields a fully specified D' matrix, where the majority of the elements in each row is the same, but their value differ in different rows. Therefore D' can be efficiently stored. The matrix factorization method used in the initialization must also handle the above property of the similarity matrix efficiently. The weighted ALS fulfills this condition.

4. **Binarized correlation:** Correlation matrices can be ill-conditioned e.g. when the values of different rows of the description matrix are from different scale. In such case one may opt for the correlation of the binarized description vectors. For this, D should be first binarized and then the above transformation applied.

4.4 Composite initialization

Several possible description matrices were mentioned in Section 3 using different information sources. It is possible that by combining these information, higher accuracy can be achieved. There are several ways to combine them. Our first method puts the description matrices one after another, so the description vector of an item is the concatenation of the description vectors ($D = (D_1|D_2|\ldots|D_n)$, here $|$ denotes the concatenation). The method requires matrices to have the same row size that is achieved by imputing rows with zeroes into smaller matrices. Weighting can also be used: $D = (w_1D_1|w_2D_2|\ldots|w_nD_n)$ The description matrix created in this manner has many columns and might be hard to factorize precisely. Our second approach defines the initial feature matrix as a weighted combination of the feature matrices created using different descriptions: $F = w_1F_1 + w_2F_2 + \ldots + w_nF_n$. The size of all F_i should be the same.

4.5 Computational cost

Each of the presented initialization methods start with the factorization of the item–descriptor matrix. Since the zeroes in this matrix should be taken into account to get a good approximation of the similarities, using implicit ALS is a natural choice for the factorization algorithm. The cost of this initial factorization is $O(K^3(N_D+M)+K^2D^+)$ where N_D is the number of possible descriptors (e.g.: number of metadata, number of context-states, etc) and D^+ is the number of non-zero values in the descriptor matrix D.

The complexity of SimFactor—in addition to the initial matrix factorization—is $O(K^2N+K^3+K^2M)$, where the terms correspond to the calculation of Y^TY, finding the eigen-decomposition, and calculating $F = XU\sqrt{\Lambda}$, respectively. The cost of Sim^{2}Factor scales similarly to SimFactors (the only difference is in the constant multiplier). This $O(K^2N + K^3 + K^2M)$ cost of the transformation steps is negligible compared to the $O(K^3(N_D + M) + K^2D^+)$ cost of the initial matrix factorization. Therefore the bottleneck of the method is the factorization of the description matrix. If N_D is the same order of magnitude as N then the cost of the initialization is roughly the same as that of the factorization of the user–item matrix therefore the total running time doubles. This is comparable when training the standard model without initialization but with more factors, but without the advantages initialization. We also note that in practice the running time of the implicit ALS is quite reasonable for low factor models and it can be further enhanced by approximating the ALS algorithm [Pilászy et al., 2010, Takács et al., 2011]. Note that these enhancements can also be applied to the factorization of the description matrix.

5 Results

We used five data sets for the experimentation. In all cases, time-based train–test splits were used. *MovieLens 10M* [Resnick et al., 1994] was transformed into implicit feedback data sets (1) keeping only the 5 star ratings and (2) keeping ratings with values 4.5 and 5 as positive feedbacks. We used the 7 days for testing (from 01/12/2008) and the earlier events for training. The *TV1* and *TV2* data sets are VoD consumption data [Cremonesi and Turrin, 2009]. Here we use the last week and the last day, respectively, for testing. The *LastFM 1K* [Celma, 2010] data set contains music listening data of ∼1 000 users on songs of ∼170 000 artists (artists are considered items). The training set contains all events until 28/04/2009. *Grocery* data set contains purchase events of an online store. The number of events is around 6.24 million targeted on 17,000 items (of them 14,000 has at least one event). The last month of 5 years' data was used for testing.

We used various data sources when creating the description matrix (see details in *Method*). For context, we chose the time stamp of the content consumption, which is available in almost every data set. Based on this, we determine the notion of seasonality, the periodicity of user behavior patterns observed in the data; this may vary for different domains. Within a season we do not expect repetitions in the aggregated behavior of users, but we expect that at the same time offset in different seasons, the aggregated behavior of the users are similar. For example it is reasonable to set the season length to be 1 day for video-on-demand (VoD) consumption, however, for shopping data 1 week or 1 month is a more reasonable choice.

Having the length of the season determined, we need to create *time bands* (bins) in the seasons. These time bands are the possible context-states and correspond to time interval of equal or varying length. The time context of an event is the time band in which it happened. We used different seasonality and time band lengths and kept only the best results.

Not all description matrices were used for every dataset. For example sufficient metadata is only available for the Grocery. We also found that context-state and context-event descriptors often work similarly therefore we used the latter only for TV1 and Grocery.

In the first experiment we compared weighted ALS and SimFactor to characterize their similarity preserving capability. We drew randomly 2 times 10 000

Table 1: Accuracy of the similarity prediction

Database	Input data	Non-zero ratio in D	Method	RMSE	RMSE improvement	Order match
Grocery	Item metadata	0.09%	ALS SimFactor	0.2683 0.0389	85.51%	67.60% 85.38%
	Item context-state	22.81%	ALS SimFactor	0.4923 0.0192	96.09%	92.97% 98.29%
	Item context-event	0.01%	ALS SimFactor	0.0332 0.0254	23.47%	63.24% 61.30%
	User context-state	21.97%	ALS SimFactor	0.3363 0.0033	99.03%	89.53% 99.92%
	User context-event	0.04%	ALS SimFactor	0.0425 0.0215	49.34%	66.70% 74.17%
TV1	Item context-state	66.10%	ALS SimFactor	0.5056 0.0144	97.16%	94.32% 97.49%
	Item context-event	0.01%	ALS SimFactor	0.0602 0.0602	0.00%	61.57% 61.59%
	User context-state	16.29%	ALS SimFactor	0.3541 0.0114	96.78%	87.30% 99.54%
	User context-event	0.04%	ALS SimFactor	0.1879 0.1488	20.82%	57.94% 57.51%
TV2	Item context-state	42.02%	ALS SimFactor	0.4426 0.0231	94.78%	94.97% 97.84%
	User context-state	5.08%	ALS SimFactor	0.2669 0.1384	48.15%	80.00% 80.78%
MovieLens (5)	Item context-state	39.08%	ALS SimFactor	0.3166 0.0425	86.59%	90.40% 94.26%
	User context-state	13.53%	ALS SimFactor	0.3830 0.0009	99.78%	85.06% 100.00%
MovieLens (4.5)	Item context-state	47.05%	ALS SimFactor	0.3380 0.0405	88.02%	91.62% 95.04%
	User context-state	11.74%	ALS SimFactor	0.3316 0.0804	75.76%	83.61% 95.44%
LastFM	Item context-state	25.87%	ALS SimFactor	0.2892 0.0380	86.87%	84.66% 96.52%
	User context-state	79.69%	ALS SimFactor	0.5214 0.0125	97.61%	83.07% 98.75%

entity pairs, calculated the original and the approximated similarity values and counted when the later similarity pairs matched their original order. We also measured the RMSE (root mean square error) of the similarity value prediction.

The results in Table 1 show that SimFactor was more accurate than weighted ALS in every experiment. The improvement in RMSE is usually over 80% except when the description matrix is very sparse. In addition to better accuracy, SimFactor also preserves the original relations of the similarities better than the weighted ALS. One can observe that the results depend greatly on the description matrix.

We used the following ranking measures to evaluate recommendation lists with 10, 20 and 50 cutoffs: recall, MAP (mean average precision), NDCG (normalized discounted cumulative gain) and MRR (mean reciprocal rank). We used low-factor models as they are of practical use. As baseline, we ran several experiments with different random initializations and chose the best result. In other words the baseline is the vanilla implicit ALS. We used weighted ALS, SimFactor and Sim^{2}Factor for initialization (the latter two apply weighted ALS as their 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 for each input data type.

Table 2 and Table 3 summarize the results of our experiments: the top5 initialization methods for each dataset by recall@50 and MAP@50 are shown. Other metrics produce similar results (not shown here). One can observe that SimFactor and Sim^{2}Factor methods clearly outperform the standard factorization based initialization, as most of the top performing initializations are achieved with those methods. For a given dataset, the top5 is typically dominated by similar initialization methods. As for the similarity metric, correlation and binarized correlation is often more efficient than cosine similarity.

Figure 3 shows the improvement achieved in recall by the top5 methods on each dataset with different cutoffs (10, 20 and 50). The improvement over the baseline is greater at shorter list, therefore a proper initialization can be better exploited with at the practically more important case.

We point out that on Grocery top the performing methods with all evaluation metrics use context for initialization (opposed to metadata). This suggest that context, 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 bands; for example adult programs mostly viewed late night. The distribution of the events for an entity in the time bands appears to be an efficient descriptor.

We also experimented on Grocery with composite initialization methods (see Section 4.4) using the context-state and metadata description matrices. When using the concatenation of description matrices, none of the evaluation metric could be improved over the better initialization. On the other hand, using the

Table 2: Recall@50 values for top5 methods per dataset

Dataset	Description	Similarity function	Algorithm	Value	Improvement over baseline
Grocery	User context-state	Cos. Sim.	MF	0.1612	8.40%
	User context-state	Correlation	MF	0.1611	8.33%
	User context-state	Correlation	SimFactor	0.1604	7.85%
	User context-state	Cos. Sim.	SimFactor	0.1602	7.74%
	User context-state	Bin. Corr.	MF	0.1601	7.63%
	Random initialization (baseline)			0.1458	N/A
TV1	User context-event	Bin. Corr.	MF	0.2924	7.69%
	User context-event	Correlation	MF	0.2924	7.69%
	User context-event	Cos. Sim.	Sim^{2}Factor	0.2924	7.69%
	User context-event	Bin. Corr.	Sim^{2}Factor	0.2921	7.57%
	User context-event	Correlation	Sim^{2}Factor	0.2921	7.57%
	Random initialization (baseline)			0.2716	N/A
TV2	User context-state	Cos. Sim.	MF	0.4223	3.73%
	User context-state	Cos. Sim.	SimFactor	0.4210	3.42%
	User context-state	Scalar Prod.	SimFactor	0.4210	3.42%
	User context-state	Correlation	MF	0.4209	3.41%
	User context-state	Bin. Corr.	MF	0.4205	3.29%
	Random initialization (baseline)			0.4071	N/A
ML (5)	User context-state	Bin. Corr.	SimFactor	0.1656	28.57%
	User context-state	Cos. Sim.	SimFactor	0.1626	26.19%
	User context-state	Cos. Sim.	MF	0.1626	26.19%
	Item context-state	Bin. Corr.	Sim^{2}Factor	0.1626	26.19%
	User context-state	Correlation	SimFactor	0.1564	21.43%
	Random initialization (baseline)			0.1288	N/A
ML (4.5)	User context-state	Scalar Prod.	Sim^{2}Factor	0.1334	19.42%
	User context-state	Scalar Prod.	SimFactor	0.1312	17.48%
	User context-state	Cos. Sim.	SimFactor	0.1302	16.50%
	User context-state	Bin. Corr.	Sim^{2}Factor	0.1291	15.53%
	User context-state	Bin. Corr.	SimFactor	0.1291	15.53%
	Random initialization (baseline)			0.1117	N/A
LastFM	User context-state	Bin. Corr.	Sim^{2}Factor	0.0950	113.43%
	Item context-state	Correlation	Sim^{2}Factor	0.0947	112.66%
	Item context-state	Cos. Sim.	Sim^{2}Factor	0.0941	111.25%
	Item context-state	Scalar Prod.	Sim^{2}Factor	0.0941	111.25%
	Item context-state	Scalar Prod.	SimFactor	0.0932	109.34%
	Random initialization (baseline)			0.0445	N/A

weighted sum of initial vectors provided better results than both initialization methods. Figure 4 shows the improvement by method and metric.

Table 3: MAP@50 values for top5 methods per dataset

Dataset	Description	Similarity function	Algorithm	Value	Improvement over baseline
Grocery	User context-state	Cos. Sim.	SimFactor	0.1315	8.27%
	User context-state	Cos. Sim.	MF	0.1295	6.65%
	User context-state	Correlation	MF	0.1291	6.29%
	User context-state	Bin. Corr.	MF	0.1283	5.60%
	User context-state	Correlation	SimFactor	0.1275	4.94%
	Random initialization (baseline)			0.1194	N/A
TV1	Item context-event	Bin. Corr.	Sim^2Factor	0.0444	24.23%
	Item context-event	Correlation	Sim^2Factor	0.0444	24.23%
	Item context-event	Cos. Sim.	Sim^2Factor	0.0437	22.40%
	Item context-event	Scalar Prod.	Sim^2Factor	0.0437	22.40%
	Item context-state	Cos. Sim.	MF	0.0433	21.36%
	Random initialization (baseline)			0.0357	N/A
TV2	Item context-state	Correlation	SimFactor	0.0770	7.83%
	Item context-state	Cos. Sim.	SimFactor	0.0770	7.82%
	User context-state	Bin. Corr.	SimFactor	0.0764	6.88%
	User context-state	Scalar Prod.	MF	0.0763	6.77%
	User context-state	Cos. Sim.	SimFactor	0.0761	6.50%
	Random initialization (baseline)			0.0714	N/A
ML (5)	User context-state	Cos. Sim.	Sim^2Factor	0.0507	19.80%
	User context-state	Scalar Prod.	Sim^2Factor	0.0501	18.49%
	User context-state	Bin. Corr.	Sim^2Factor	0.0501	18.37%
	User context-state	Bin. Corr.	MF	0.0497	17.50%
	Item context-state	Bin. Corr.	SimFactor	0.0488	15.34%
	Random initialization (baseline)			0.0423	N/A
ML (4.5)	User context-state	Scalar Prod.	SimFactor	0.0333	50.62%
	Item context-state	Scalar Prod.	SimFactor	0.0329	48.81%
	User context-state	Bin. Corr.	Sim^2Factor	0.0320	44.70%
	Item context-state	Correlation	Sim^2Factor	0.0314	42.09%
	Item context-state	Scalar Prod.	MF	0.0313	41.45%
	Random initialization (baseline)			0.0221	N/A
LastFM	User context-state	Bin. Corr.	MF	0.1474	309.44%
	User context-state	Bin. Corr.	Sim^2Factor	0.1369	280.19%
	Item context-state	Correlation	MF	0.1305	262.42%
	Item context-state	Scalar Prod.	Sim^2Factor	0.1299	260.74%
	Item context-state	Cos. Sim.	MF	0.1278	255.06%
	Random initialization (baseline)			0.0360	N/A

6 Conclusion

In this paper we proposed a general framework for similarity based initialization of MF algorithms. Our hypothesis was that initializing item and user models with weights that reflect the similarity between entities improves algorithm performance when compared to a random initialization. This method also allows us

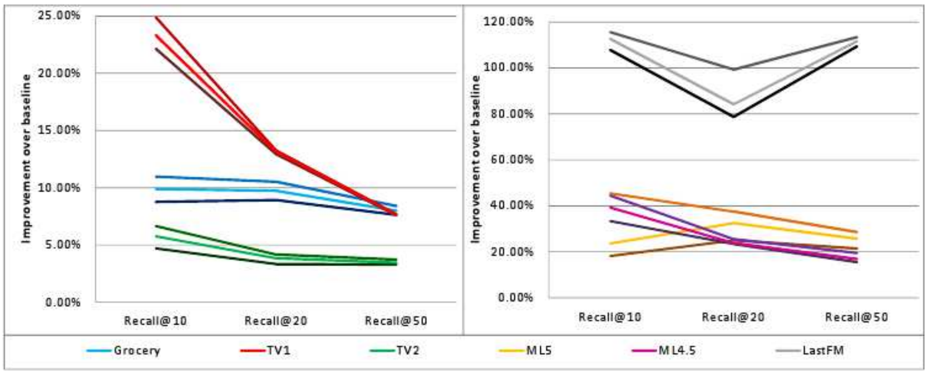

Figure 3: Recall improvement over the baseline at cutoffs 10, 20 and 50 for each dataset. Each base color denotes a dataset, the three lines of similar color show the maximal, the average and the minimal improvement by the top5 methods for that dataset.

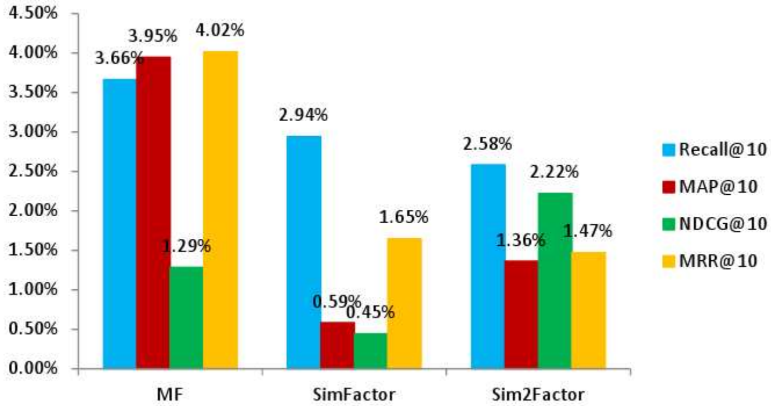

Figure 4: Improvement by the composite initialization (over the better of the members) on short recommendation lists

to easily incorporate additional information like context or metadata information into the MF framework.

We proposed the algorithm SimFactor that implements the general idea. We found that SimFactor indeed preserves the original similarities and their relation better than other MF methods. The SimFactor initialization has a negligible computational cost when used with an arbitrary MF method. We also extended SimFactor to use similarity based similarities.

The data source taken to compute similarity greatly affects the performance gain observed with initialization. We found that the the largest improvement can be achieved by using the contextual information (here we used seasonality). Context separates the entities better than any other information we experimented with. The similarity preserving property of the SimFactor can be a disadvantage when the description matrix is too noisy, that is, when the description matrix does not capture item or user similarities. We left for further research the investigation of the effect of alternative contexts, or how to handle continuous ranged context dimensions such as, e.g., product price.

The usefulness of SimFactor-based initialization can be regarded as a trade-off between training time and accuracy gain. Both quantities depend on the description data that is application domain specific. Here context based description is again more advantageous than metadata based ones, because the size of the context description vectors is smaller than with metadata. This promotes context based initialization to be used in real-world recommender applications.

References

[Albright et al., 2006] Albright, R., Cox, J., Duling, D., Langville, A., and Meyer, C. (2006). Algorithms, initializations, and convergence for the nonnegative matrix factorization. Technical Report Math 81706, NC State University.

[Bell and Koren, 2007] Bell, R. M. and Koren, Y. (2007). Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proc of. ICDM-07: IEEE Int. Conf. on Data Mining*, pages 43–52, Omaha, Nebraska, USA.

[Bennett and Lanning, 2007] Bennett, J. and Lanning, S. (2007). The Netflix Prize. In *Proc. of KDD Cup Workshop at SIGKDD-07: ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 3–6, San Jose, CA, USA.

[Boutsidis and Gallopoulos, 2008] Boutsidis, C. and Gallopoulos, E. (2008). Svd based initialization: A head start for nonnegative matrix factorization. *Pattern Recognition*, 41(4):1350–1362.

[Celma, 2010] Celma, O. (2010). *Music Recommendation and Discovery in the Long Tail.* Springer.

[Cremonesi and Turrin, 2009] Cremonesi, P. and Turrin, R. (2009). Analysis of cold-start recommendations in IPTV systems. In *Proc. of RecSys-09: ACM Conf. on Recommender Systems*, pages 233–236, New York, NY, USA.

[Dhillon and Modha, 2000] Dhillon, I. S. and Modha, D. S. (2000). Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1–2):143–175.

[Herlocker et al., 2004] Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53.

[Hu et al., 2008] Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *Proc. of ICDM-08: IEEE Int. Conf. on Data Mining*, pages 263–272, Pisa, Italy.

[Jahrer and Töscher, 2012] Jahrer, M. and Töscher, A. (2012). Collaborative filtering ensemble for ranking. *JMLR Workshop and Conference Proceedings*, 18:153–167. Proceedings of KDD-Cup 2011 competition.

[Jolliffe, 1986] Jolliffe, I. (1986). *Principal Component Analysis.* Springer Verlag.

[Koren, 2008] Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proc. of SIGKDD-08: ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 426–434, Las Vegas, NV, USA.

[Kramer, 1991] Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243.

[Pilászy and Tikk, 2009] Pilászy, I. and Tikk, D. (2009). Recommending new movies: Even a few ratings are more valuable than metadata. In *Proc. of RecSys-09: ACM Conf. on Recommender Systems*, pages 93–100, New York, NY, USA.

[Pilászy et al., 2010] Pilászy, I., Zibriczky, D., and Tikk, D. (2010). Fast ALS-based matrix factorization for explicit and implicit feedback datasets. In *Proc. of RecSys-10: ACM Conf. on Recommender Systems*, pages 71–78, Barcelona, Spain.

[Rendle and Schmidt-Thieme, 2008] Rendle, S. and Schmidt-Thieme, L. (2008). Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Proc. of RecSys-08: ACM Conf. on Recommender Systems*, pages 251–258, Lausanne, Switzerland.

[Resnick et al., 1994] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). Grouplens: an open architecture for collaborative filtering of netnews. In *Proc. of CSCW-94: ACM Conf. on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, NC, USA.

[Ricci et al., 2011] Ricci, F., Rokach, L., and Shapira, B. (2011). Introduction to recommender systems handbook. In Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, Artificial Intelligence, pages 1–35. Springer US.

[Salakhutdinov and Mnih, 2008] Salakhutdinov, R. and Mnih, A. (2008). Probabilistic matrix factorization. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20*, pages 1257–1264. MIT Press, Cambridge, MA, USA.

[Shi et al., 2012] Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Oliver, N., and Hanjalic, A. (2012). CLiMF: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proc. of RecSys-12: ACM Conf. on Recommender Systems*, pages 139–146, Dublin, Ireland.

[Smilde et al., 2004] Smilde, A., Bro, R., and Geladi, P. (2004). *Multi-way Analysis*. Wiley, West Sussex, England.

[Snedecor and Cochran, 1980] Snedecor, G. W. and Cochran, W. G. (1980). *Statistical Methods*. Iowa State University Press, 7th edition.

[Takács et al., 2007] Takács, G., Pilászy, I., Németh, B., and Tikk, D. (2007). Major components of the Gravity recommendation system. *SIGKDD Explor. Newsl.*, 9:80–83.

[Takács et al., 2009] Takács, G., Pilászy, I., Németh, B., and Tikk, D. (2009). Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656.

[Takács et al., 2011] Takács, G., Pilászy, I., and Tikk, D. (2011). Applications of the conjugate gradient method for implicit feedback collaborative filtering. In *Proc. of RecSys-11: ACM Conf. on Recommender Systems*, pages 297–300, Chicago, IL, USA.

[Takács and Tikk, 2012] Takács, G. and Tikk, D. (2012). Alternating least squares for personalized ranking. In *Proc. of RecSys-12: ACM Conf. on Recommender Systems*, pages 83–90, Dublin, Ireland.

[Thai-Nghe et al., 2012] Thai-Nghe, N., Drumond, L., Horváth, T., Krohn-Grimberghe, A., Nanopoulos, A., and Schmidt-Thieme, L. (2012). Factorization techniques for predicting student performance. In *Educational Recommender Systems and Technologies: Practices and Challenges (ERSAT 2011)*, pages 1–25. IGI Global.

[Wild et al., 2004] Wild, S. M., Curry, J. H., and Dougherty, A. (2004). Improving non-negative matrix factorizations through structured initialization. *Pattern Recognition*, 37(11):2217–2232.