Fast ALS-Based Tensor Factorization for Context-Aware Recommendation from Implicit Feedback

Balázs Hidasi[1,2,⋆] and Domonkos Tikk[1]

[1] Gravity R&D Ltd.
[2] Budapest University of Technology and Economics
{balazs.hidasi,domonkos.tikk}@gravityrd.com

Abstract. Albeit the implicit feedback based recommendation problem—when only the user history is available but there are no ratings—is the most typical setting in real-world applications, it is much less researched than the explicit feedback case. State-of-the-art algorithms that are efficient on the explicit case cannot be straightforwardly transformed to the implicit case if scalability should be maintained. There are few implicit feedback benchmark datasets, therefore new ideas are usually experimented on explicit benchmarks. In this paper, we propose a generic context-aware implicit feedback recommender algorithm, coined iTALS. iTALS applies a fast, ALS-based tensor factorization learning method that scales linearly with the number of non-zero elements in the tensor. The method also allows us to incorporate various contextual information into the model while maintaining its computational efficiency. We present two context-aware implementation variants of iTALS. The first incorporates seasonality and enables to distinguish user behavior in different time intervals. The other views the user history as sequential information and has the ability to recognize usage pattern typical to certain group of items, e.g. to automatically tell apart product types that are typically purchased repetitively or once. Experiments performed on five implicit datasets (LastFM 1K, Grocery, VoD, and "implicitized" Netflix and MovieLens 10M) show that by integrating context-aware information with our factorization framework into the state-of-the-art implicit recommender algorithm the recommendation quality improves significantly.

Keywords: recommender systems, tensor factorization, context awareness, implicit feedback.

1 Introduction

Recommender systems are information filtering algorithms that help users in information overload to find interesting items (products, content, etc). Users get personalized recommendations that contain typically a few items deemed to be of user's interest. The relevance of an item with respect to a user is predicted by

⋆ B. Hidasi was supported by TÁMOP-4.2.2.B-10/1–2010-0009.

P. Flach et al. (Eds.): ECML PKDD 2012, Part II, LNCS 7524, pp. 67–82, 2012.

recommender algorithms; items with the highest prediction scores are displayed to the user.

Recommender algorithms are usually sorted into two main approaches: the content based filtering (CBF) and the collaborative filtering (CF). CBF algorithms use user metadata (e.g. demographic data) and item metadata (e.g. author, genre, etc.) and predict user preference using these attributes. In contrast, CF methods do not use metadata, but only data of user–item interactions. Depending on the nature of the interactions, CF algorithms can be further 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; 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 [1].

CF algorithms can be classified into memory-based and model-based ones. Until recently, memory-based solutions were concerned as the state-of-the-art. These are neighbor methods that make use of item or user rating vectors to define similarity, and they calculate recommendations as a weighted average of similar item or user rating vectors. In the last few years, model-based 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 the largest explicit benchmark dataset (100M ratings) [2] for a long time.

Model-based methods build generalized models that intend to capture user preference. The most successful approaches are the latent factor algorithms. These 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 applied to compute these vectors, which approximate the partially known rating matrix using alternating least squares (ALS) [3], gradient [4] and coordinate descent method [5], conjugate gradient method [6], singular value decomposition [7], or a probabilistic framework [8].

Explicit feedback based methods are able to provide accurate recommendations if enough ratings are available. In certain application areas, such as movie rental, travel applications, video streaming, users have motivation to provide ratings to get better service, better recommendations, or award or punish a certain vendor. However, in general, users of an arbitrary online service 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 (2) they are not motivated to return later to the system to rate. 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 [9]. 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

difficult 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 has been less studied. The proposed solutions for the implicit task are often the algorithms for the explicit problems that had been modified in a way that they can handle the implicit task.

The classical MF methods only consider user-item interaction (ratings or events) when building the model. However, we may have additional information related to items, users or events, which are together termed *contextual information*, or briefly *context*. Context can be, for instance, the time or location of recommendation, social networks of users, or user/item metadata [10]. Integrating context can help to improve recommender models. Tensor factorization have been suggested as a generalization of MF for considering contextual information [11]. However, the existing methods only work for the explicit problem. In this work, we developed a tensor factorization algorithm that can efficiently handle the implicit recommendation task.

The novelty of our work is threefold: (1) we developed a fast tensor factorization method—coined iTALS—that can efficiently factorize huge tensors; (2) we adapted this general tensor factorization to the implicit recommendation task; (3) we present two specific implementations of this general implicit tensor factorization that consider different contextual information. The first variant uses seasonality which was also used in [11] for the explicit problem. The second algorithm applies sequentiality of user actions and is able to learn association rule like usage patterns. By using these patterns we can tell apart items or item categories having been purchased with different repetitiveness, which improves the accuracy of recommendations. To our best knowledge, iTALS is the first factorization algorithm that uses this type of information.

This paper is organized as follows. Section 2 briefly reviews related work on context-aware recommendation algorithms and tensor factorization. In Section 3 we introduce our tensor factorization method and its application to the implicit recommendation task. Section 4 shows two application examples of our factorization method: (1) we show how seasonality can be included in recommendations and (2) we discuss how a recommendation algorithm can learn repetitiveness patterns from the dataset. Section 5 presents the results of our experiments, and Section 6 sums up our work and derive the conclusions.

1.1 Notation

We will use the following notation in the rest of this paper:

- $A \circ B \circ \ldots \rightarrow$ The Hadamard (elementwise) product of A, B, ... The operands are of equal size, and the result's size is also the same. The element of the result at $(i, j, k, \ldots)$ is the product of the element of A, B, ... at $(i, j, k, \ldots)$. This operator has higher precedence than matrix multiplication in our discussion.

- $A_{\bullet,i}/A_{i,\bullet} \rightarrow$ The i^{th} column/row of matrix A.
- $A_{i_1,i_2,\ldots} \rightarrow$ The $(i_1, i_2, \ldots)$ element of tensor/matrix A.
- $K \rightarrow$ The number of features, the main parameter of factorization.
- $D \rightarrow$ The number of dimensions of the tensor.
- $T \rightarrow$ A D dimensional tensor that contains only zeroes and ones (preference tensor).
- $W \rightarrow$ A tensor with the exact same size as T (weight tensor).
- $S_i \rightarrow$ The size of T in the i^{th} dimension $(i = 1, \ldots, D)$.
- $N^+ \rightarrow$ The number of non-zero elements in tensor T.
- $M^{(i)} \rightarrow$ A $K \times S_i$ sized matrix. Its columns are the feature vectors for the entities in the i^{th} dimension.

2 Related Work

Context-aware recommender systems [12] emerged as an important research topic in the last years and entire workshops are devoted to this topic on major conferences (CARS series started in 2009 [13], CAMRA in 2010 [14]). The application fields of context-aware recommenders include among other movie [15] and music recommendation [16], point-of-interest recommendation (POI) [17], citation recommendation [18]. Context-aware recommender approaches can be classified into three main groups: pre-filtering, post-filtering and contextual modeling [10]. Baltrunas and Amatriain [16] proposed a pre-filtering approach by partitioned user profiles into *micro-profiles* based on the time split of user event falls, and experimented with different time partitioning. Post-filtering ignores the contextual data at recommendation generation, but disregards irrelevant items (in a given context) or adjust recommendation score (according to the context) when the recommendation list is prepared; see a comparison in [19]. The tensor factorization based solutions, including our proposed approach, falls into the contextual modeling category.

Tensor factorization incorporates contextual information into the recommendation model. Let us have a set of items, users and ratings (or events) and assume that additional context of the ratings is available (e.g. time of the rating). Having C different contexts, the rating data can be cast into a $C + 2$ dimensional tensor. The first dimension corresponds to users, the second to items and the subsequent C dimensions $[3, \ldots, C + 2]$ are devoted to contexts. We want to decompose this tensor into lower rank matrices and/or tensors in a way that the reconstruction the original tensor from its decomposition approximates well the original tensor. Approximation accuracy is calculated at the known positions of the tensor using RMSE as error measure. In [11], a sparse HOSVD [20] method is presented that decomposes a D dimensional sparse tensor into D matrices and a D dimensional tensor. If the size of the original tensor is $S_1 \times S_2 \times \cdots \times S_D$ and the number of features is K then the size of the matrices are $S_1 \times K$, $S_2 \times K$, $\ldots$, $S_D \times K$ and the size of the tensor is $K \times K \times \cdots \times K$. The authors use gradient descent on the known ratings to find the decomposition, and by doing so, the complexity of one iteration of their algorithm scales *linearly* with the number of

non-missing values in the original tensor (number of rating) and *cubically* with the number of features (K). This is much less than the cost of the dense HOSVD, which is $O(K \cdot (S_1 + \cdots + S_D)^D)$. A further improvement was proposed by Rendle *et al* [21], where the computational complexity was reduced so that their method scales linearly *both* with the number of explicit ratings and with the number of features. However, if the original tensor is large and dense like for the implicit recommendation task then neither method scales well.

3 ALS Based Fast Tensor Factorization

In this section we present iTALS, a general ALS-based tensor factorization algorithm that scales linearly with the non-zero element of a dense tensor (when appropriate weighting is used) and cubically with the number of features. This property makes our algorithm suitable to handle the context-aware implicit recommendation problem.

Let T be a tensor of zeroes and ones and let W contain weights to each element of T. $T_{u,i,c_1,\cdots,c_C}$ is 1 if user u has (at least one) event on item i while the context-state of j^{th} context dimension was c_j, thus the proportion of ones in the tensor is very low. An element of W is 1 if the corresponding element in T is 0 and greater than 1 otherwise. Instead of using the form of the common HOSVD decomposition (D matrices and a D dimensional tensor) we decompose the original T tensor into D matrices. The size of the matrices are $K \times S_1, K \times S_2, \ldots, K \times S_D$. The prediction for a given cell in T is the elementwise product of columns from $M^{(i)}$ low rank matrices. Equation 1 describes the model.

$$\hat{T}_{i_1,i_2,\ldots,i_D} = 1^T M^{(1)}_{\bullet,i_1} \circ M^{(2)}_{\bullet,i_2} \circ \cdots \circ M^{(D)}_{\bullet,i_D} \tag{1}$$

We want to minimize the loss function of equation 2:

$$L(M^{(1)}, \ldots, M^{(D)}) = \sum_{i_1=1,\ldots,i_D=1}^{S_1,\ldots,S_D} W_{i_1,\ldots,i_D} \left(T_{i_1,\ldots,i_D} - \hat{T}_{i_1,\ldots,i_D} \right)^2 \tag{2}$$

If all but one $M^{(i)}$ is fixed, L is convex in the non-fixed variables. We use this method to minimize the loss function. L reaches its minimum (in $M^{(i)}$) where its derivate with respect to $M^{(i)}$ is zero. Since the derivate of L is linear in $M^{(i)}$ the columns of the matrix can be computed separately. For the $(i_1)^{\text{th}}$ column of $M^{(1)}$:

$$\begin{aligned} 0 = \frac{\partial L}{\partial M^{(1)}_{\bullet,i_1}} = -2 \underbrace{\sum_{i_2=1,\ldots,i_D=1}^{S_2,\ldots,S_D} W_{i_2,\ldots,i_D} T_{i_1,\ldots,i_D} \left(M^{(2)}_{\bullet,i_2} \circ \cdots \circ M^{(D)}_{\bullet,i_D} \right)}_{\mathcal{O}} + \\ \underbrace{2 \sum_{i_2=1,\ldots,i_D=1}^{S_2,\ldots,S_D} W_{i_2,\ldots,i_D} \left(M^{(2)}_{\bullet,i_2} \circ \cdots \circ M^{(D)}_{\bullet,i_D} \right) \left(M^{(2)}_{\bullet,i_2} \circ \cdots \circ M^{(D)}_{\bullet,i_D} \right)^T M^{(1)}_{\bullet,i_1}}_{\mathcal{I}} \end{aligned} \tag{3}$$

It takes $O(DKN_{i_1}^+)$ time to compute $\mathcal{O}$ in equation 3, because only $N_{i_1}^+$ cells of T for i_1 in the first dimension contain ones, the others are zeroes. For every column it yields a complexity of $O(DKN^+)$. The naive computation of $\mathcal{I}$ however is very expensive computationally: $O(K\prod_{i=2}^{D} S_i)$. Therefore we transform $\mathcal{I}$ by using $W_{i_2,\ldots,i_D} = W'_{i_2,\ldots,i_D} + 1$ and get:

$$\mathcal{I} = \sum_{i_2=1,\ldots,i_D=1}^{S_2,\ldots,S_D} W'_{i_2,\ldots,i_D} \left(M_{\bullet,i_2}^{(2)} \circ \cdots \circ M_{\bullet,i_D}^{(D)}\right)\left(M_{\bullet,i_2}^{(2)} \circ \cdots \circ M_{\bullet,i_D}^{(D)}\right)^T M_{\bullet,i_1}^{(1)} + \underbrace{\sum_{i_2=1,\ldots,i_D=1}^{S_2,\ldots S_D} \left(M_{\bullet,i_2}^{(2)} \circ \cdots \circ M_{\bullet,i_D}^{(D)}\right)\left(M_{\bullet,i_2}^{(2)} \circ \cdots \circ M_{\bullet,i_D}^{(D)}\right)^T M_{\bullet,i_1}^{(1)}}_{\mathcal{J}} \tag{4}$$

The first part in equation 4 can be calculated in $O(K^2N_{i_1}^+)$ as $W'_{i_2,\ldots,i_D} = (W_{i_2,\ldots,i_D} - 1)$ and the weights for the zero elements of T are ones. This step is the generalization of the Hu *et. al*'s adaptation of ALS to the implicit problem [22]. The total complexity of calculating all columns of the matrix is $O(K^2N^+)$. $\mathcal{J}$ is the same for all columns of $M^{(1)}$ (independent of i_1) and thus can be pre-computed. However the cost of directly computing $\mathcal{J}$ remains $O(K\prod_{i=2}^{D} S_i)$. Observe the following:

$$\begin{aligned}\mathcal{J}_{j,k} &= \left(\sum_{i_2=1,\ldots,i_D=1}^{S_2,\ldots,S_D} \left(M_{\bullet,i_2}^{(2)} \circ \cdots \circ M_{\bullet,i_D}^{(D)}\right)\left(M_{\bullet,i_2}^{(2)} \circ \cdots \circ M_{\bullet,i_D}^{(D)}\right)^T\right)_{j,k} = \\ &= \sum_{i_2=1,\ldots,i_D=1}^{S_2,\ldots,S_D} \left(M_{j,i_2}^{(2)} \cdot \ldots \cdot M_{j,i_D}^{(D)}\right)\left(M_{k,i_2}^{(2)} \cdot \ldots \cdot M_{k,i_D}^{(D)}\right) = \\ &= \left(\sum_{i_2=1}^{S_2} M_{j,i_2}^{(2)} M_{k,i_2}^{(2)}\right) \cdot \ldots \cdot \left(\sum_{i_D=1}^{S_D} M_{j,i_D}^{(D)} M_{k,i_D}^{(D)}\right)\end{aligned} \tag{5}$$

Using equation 5 we can transform the second part from equation 4 into the following form:

$$\begin{aligned}\mathcal{J} &= \sum_{i_2=1,\ldots,i_D=1}^{S_2,\ldots,S_D} \left(M_{\bullet,i_2}^{(2)} \circ \cdots \circ M_{\bullet,i_D}^{(D)}\right)\left(M_{\bullet,i_2}^{(2)} \circ \cdots \circ M_{\bullet,i_D}^{(D)}\right)^T = \\ &= \underbrace{\left(\sum_{i_2=1}^{S_2} M_{\bullet,i_2}^{(2)}\left(M_{\bullet,i_2}^{(2)}\right)^T\right)}_{\mathcal{M}^{(2)}} \circ \cdots \circ \underbrace{\left(\sum_{i_D=1}^{S_D} M_{\bullet,i_D}^{(D)}\left(M_{\bullet,i_D}^{(D)}\right)^T\right)}_{\mathcal{M}^{(D)}}\end{aligned} \tag{6}$$

The members of equation 6 can be computed in $O(S_iK^2)$ time. From the $\mathcal{M}^{(i)}$ matrices the expression can be calculated in $O(K^2D)$ time. Note that the $\mathcal{M}^{(i)}$ is needed for computing all but the i^{th} matrix but only changes if $M^{(i)}$ changed.

Therefore we count the cost of computing $\mathcal{M}^{(i)}$ to the cost of recomputing $M^{(i)}$. To get the desired column of the matrix we need to invert a $K \times K$ sized matrix per column (see equation 3). That requires $O(K^3 S_1)$ time for all columns of $M^{(1)}$. The columns of the other matrices can be calculated similarly.

Algorithm 1. Fast ALS-based tensor factorization for implicit feedback recommendations

Input: T: a D dimensional $S_1 \times \cdots \times S_D$ sized tensor of zeroes and ones; W: a D dimensional $S_1 \times \cdots \times S_D$ sized tensor containing the weights; K: number of features; E: number of epochs
Output: $\{M^{(i)}\}_{i=1,\ldots,D}$ $K \times S_i$ sized low rank matrices
procedure iTALS(T, W, K, E)

for $i = 1, \ldots, D$ **do**
  $M^{(i)} \leftarrow$ Random $K \times S_i$ sized matrix
  $\mathcal{M}^{(i)} \leftarrow M^{(i)}(M^{(i)})^T$
end for
for $e = 1, \ldots, E$ **do**
  for $i = 1, \ldots, D$ **do**
    $C^{(i)} \leftarrow \mathcal{M}^{(\ell_1)} \circ \cdots \circ \mathcal{M}^{(\ell_{D-1})}, (i \notin \{\ell_1, \ldots, \ell_{D-1}\})$
    $T^{(i)} \leftarrow$ UnfoldTensor(T,i)
    for $j_i = 1, \ldots, S_i$ **do**
      $C^{(i)}_{j_i} \leftarrow C^{(i)}$
      $O^{(i)}_{j_i} \leftarrow 0$
      for all $t : \{t \in T^{(i)}_{j_i}, t \neq 0\}$ **do**
        $\{j_\ell | \ell \neq i\} \leftarrow$ Indices of t in T
        $W_t \leftarrow$ GetWeight(W,t)
        $v \leftarrow M^{(\ell_1)} \circ \cdots \circ M^{(\ell_{D-1})}, (i \notin \{\ell_1, \ldots, \ell_{D-1}\})$
        $C^{(i)}_{j_i} \leftarrow C^{(i)}_{j_i} + vW_tv^T$ and $O^{(i)}_{j_i} \leftarrow O^{(i)}_{j_i} + W_tv$
      end for
      $M^{(i)}_{\bullet,j_i} \leftarrow (C^{(i)}_{j_i} + \lambda I)^{-1} O^{(i)}_{j_i}$
    end for
    $\mathcal{M}^{(i)} \leftarrow M^{(i)}(M^{(i)})^T$
  end for
end for
return $\{M^{(i)}\}_{i=1\ldots D}$
end procedure

The total cost of computing $M^{(i)}$ is $O(K^3 S_i + K^2 N^+ + KDN^+)$ that can be simplified to $O(K^3 S_i + K^2 N^+)$ using that usually $D \ll K$. Therefore the cost of computing each matrix once is $O\left(K^3 \sum_{i=1}^{D} S_i + K^2 N^+\right)$. Thus the cost of an epoch is linear in the number of the non-zero examples and cubical in the number of features. The cost is also linear in the number of dimensions of the tensor and the sum of the length of the tensors in each dimension. We will also show in Section 5.1 that the $O(K^2)$ part is dominant when dealing with practical problems. The complexity of the gradient descent method for implicit feedback

is $O(K \prod_{i=1}^{D} S_i)$ that is linear in the number of features but the $\prod_{i=1}^{D} S_i$ part makes impossible to run it on real life datasets. Sampling can be applied to reduce that cost but it is not trivial how to sample in the implicit feedback case.

The pseudocode of the suggested iTALS (Tensor factorization using ALS for implicit recommendation problem) is given in Algorithm 1. There we use two simple functions. UnfoldTensor(T, i) unfolds tensor T by its i^{th} dimension. This step is used for the sake of clarity, but with proper indexing we would not need to actually unfold the tensor. GetWeight(W, t) gets the weight from the weight tensor W for the t element of tensor T and creates a diagonal matrix from it. The size of W_t is $K \times K$ and it contains the weight for t in its main diagonal and 0 elsewhere. The pseudocode follows the deduction above. In line 3 we precompute $\mathcal{M}^{(i)}$. We create the column independent part from equation 4 in line 7. We add the column dependent parts to each side of equation 3 in lines 12–17 and compute the desired column in line 18. In this step we use regularization to avoid numerical instability and overfitting of the model. After each column of $M^{(i)}$ is computed $\mathcal{M}^{(i)}$ is recomputed in line 20.

4 Context-Aware iTALS Algorithm

In this section we derive two specific algorithms from the generic iTALS method presented in Section 3. The first method uses seasonality as context, the second considers the user history as sequential data, and learns meta-rules about sequentiality and repetitiveness.

4.1 Seasonality

Many application areas of recommender systems exhibit the seasonality effect, therefore seasonal data is an obvious choice as context [23]. Strong periodicity can be observed in most of the human activities: as people have regular daily routines, they also follow similar patterns in TV watching at different time of a day, they do their summer/winter vacation around the same time in each year. Taking the TV watching example, it is probable that horror movies are typically watched at night and animation is watched in the afternoon or weekend mornings. Seasonality can be also observed in grocery shopping or in hotel reservation data.

In order to consider seasonality, first we have to define the length of season. During 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 will be similar. The length of the season depends on the data. For example it is reasonable to set the season length to be 1 day for VoD consumption, however, this is not an appropriate choice for shopping data, where 1 week or 1 month is more justifiable. 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. Time bands specify the time resolution of a season, which is also data dependent. We can create time bands with equal or different length.

For example, every day of a week are time bands of equal length, but 'morning', 'around noon', 'afternoon', 'evening', 'late evening', 'night' could be time bands of a day with different length. Obviously, these two steps require some a-priori knowledge about the data or the recommendation problem, but iTALS is not too sensitive to minor deviations related to the length and the resolution of the season.

In the next step, events are assigned to time bands according to their time stamp. Thus, we can create the (user, item, time band) tensor. We factorize this tensor using the iTALS algorithm and we get feature vectors for each user, for each item and for each time band. When a recommendation is requested for user u at time t, first the time band of t is determined and then the preference value for each item using the feature vector of user u and the feature vector of time band tb_t is calculated.

4.2 Sequentiality

Recommendation algorithms often recommend items from categories that the user likes. For example if the user often watches horror movies then the algorithm will recommend her horror movies. This phenomenon is even stronger if time decay is applied and so recent events have greater weights. Pushing newer events can increase accuracy, because similar items will be recommended. This functioning can be beneficial in some application fields, like VoD recommendation, but will fail in such cases where repetitiveness in user behavior with respect to items can not be observed. A typical example for that is related to household appliance products: if a user buys a TV set and then she gets further TV sets recommended, she will not probably purchase another one. In such a case, complementary or related goods are more appropriate to recommend, DVD players or external TV-tuners for example. On the other hand, the purchase of a DVD movie does not exclude at all the purchase of another one. Whether recommendation of similar items is reasonable, depends on the nature of the item and behavior of the user. Next, we propose an approach to integrate the repetitiveness of purchase patterns into the latent factor model.

Using association rules is a possible approach to specify item purchase patterns. Association rules [24] are often used to determine which products are bought frequently together and it was reported that in certain cases association rule based recommendations yield the best performance [25]. In our setting, we can extract purchase patterns from the data using association rule mining on the subsequent user events within a given time window. There are two possibilities: we can generate category–category rules, or category–item rule, thus having usage patterns:

- if a user bought an item from category A then she will buy an item from category B next time, or
- if a user bought an item from category A then she will buy an item X next time.

We face, however, with the following problems, when attempting to use such patterns in recommendations: (1) the parameter selection (minimum support, minimum confidence and minimum lift) influences largely the performance, their optimization may be slow; (2) rules with negated consequents (e.g. bought from A will not buy from B) are not found at all; (3) with category–category rules one should devise further weighting/filtering to promote/demote the items in the pushed category; (4) the category–item rules are too specific therefore either one gets too many rules or the rules will overfit.

We show how repetitiveness related usage patterns can be efficiently integrated into recommendation model using the the iTALS algorithm. Let us now consider the *category of last purchased item* as the context for the next recommendation. The tensor has again three dimensions: users, items and item categories. The (i, u, c) element of the tensor means that user u bought item i and the user's latest purchase (before buying i) was an item from category c. Using the examples above: the user bought a given DVD player after the purchase of a TV set. After factorizing this tensor we get feature vectors for the item categories as well. These vectors act as weights in the feature space that reweight the user–item relations. For example, assuming that the first item feature means "having large screen" then the first feature of the TV category would be low as such items are demoted. If the second item feature means "item can play discs" then the second feature of the TV category would be high as these items are promoted.

The advantage of this method is that it learns the usage patterns from the data globally by producing feature vectors that reweight the user–item relations. One gets simple but general usage patterns using the proposed solution that integrates seamlessly into the common factorization framework: no post-processing is required to define promotional/demotional weights/filters.

We can generalize the concept described above to take into account several recent purchases. We could create a $C + 2$ dimensional tensor, where the $[3, \ldots, C + 2]$ dimensions would represent the item categories of the last C purchases, but the resulting tensor would be very sparse as we increase C. Instead we remain at a three dimensional tensor but we set simultaneously C item categories to 1 for each user–item pair. We may also decrease the weights in W for those additional $C - 1$ cells as they belong to older purchases. Thus we may control the effect of previous purchases based on their recency. When recommending, we have to compute the (weighted) average of the feature vectors of the corresponding categories and use that vector as the context feature vector.

5 Experiments

We used five databases to validate our algorithms. Three of them contain genuine implicit feedback data (LastFM 1K and 2 proprietary), while the other two are implicit variants of explicit feedback data. The *LastFM 1K* [26] dataset contains listening habits of ∼1 000 users on songs of ∼170 000 artists (artists are considered items). The training set contains all events until 28/04/2009. The test set contains the events of the next day following the training period. In *VoD*

Table 1. Recall@20 for all datasets and algorithms using factorization with 20 and 40 features; in each row, the best and second best results are highlighted by bold and slanted typesetting, respectively

Dataset	iALS	iCA baseline time bands	iTALS time bands	iTALS seq.	iTALS seq. (2)	iTALS seq. (5)
VOD (20)	0.0632	*0.0847*	**0.1125**	0.0689	0.0678	0.0666
VOD (40)	0.0753	*0.0910*	**0.1240**	0.0855	0.0930	0.0883
Grocery (20)	0.0656	0.0803	0.1032	**0.1261**	*0.1223*	0.1153
Grocery (40)	0.0707	0.0872	0.1081	*0.1340*	**0.1351**	0.1189
LastFM 1K (20)	0.0157	0.0249	0.0352	*0.0747*	**0.0793**	0.0733
LastFM 1K (40)	0.0333	0.0351	0.0418	0.0785	**0.0851**	*0.0800*
Netflix (20)	0.0540	*0.0593*	**0.0724**	0.0512	0.0534	0.0537
Netflix (40)	0.0552	*0.0561*	**0.0671**	0.0503	0.0527	0.0538
MovieLens (20)	0.0494	*0.0553*	**0.0896**	0.0406	0.0450	0.0457
MovieLens (40)	0.0535	0.0494	**0.0937**	0.0361	0.0480	*0.0498*

consumption dataset, with 8 weeks of training data we tested on the data of the next day. Thus, all test events occurred after the last train event. The training set contains 22.5 million events and 17 000 items. The online *grocery* dataset contains only purchase events. We used a few years' data for training and one month for testing. The training set contains 6.24 million events and 14 000 items. The two explicit feedback datasets are the Netflix [2] and the MovieLens 10M [27]. We kept the five star ratings for the former and ratings of 4.5 and above for the latter and used them as positive implicit feedback. For train-test splits we used the splitting dates 15/12/2005 and 01/12/2008, respectively.

We determined the seasonality for each dataset, that is, the periodicity patterns observed in the data. As for the VoD data, we defined a day as the season and defined custom time intervals as time bands ('morning', 'around noon', 'afternoon', 'evening', 'late evening', 'night' and 'dawn'), because people watch and channels broadcast different programs at different time of the day. For LastFM 1K and MovieLens we also used a day as the season and time bands of 30 minutes. For the Grocery data we defined a week as the season and the days of the week as the time bands. The argument here is that people tend to follow different shopping behavior on weekdays and weekends. For the Netflix data only the day of the rating is available, so we decided to define a week as the season and the days of the week as time bands.

In our next experiment, we used item category with Grocery and Netflix datasets, genre with VoD and MovieLens and artists for LastFM as the category of the item for the meta-rule learning algorithm. We experimented with using the last 1, 2, 5 events prior to the current event of the users.

We compared the two iTALS variants to the basic iALS as well as to a context-aware baseline for implicit feedback data. This method, referred as *implicit CA (iCA) baseline*, is the composite of several iALS models. For each context state

we train a model using only the events with the appropriate context, e.g., with the VoD we train 7 models for the 7 time bands. The context of the recommendation request (e.g. time of day) selects the model for the prediction. This baseline treats context-states independently. Due to its long running time we used iCA only with seasonality, as #(time bands) $\ll$ #(preceding item categories).

Every algorithm has three common parameters: the number of features, the number of epochs and the regularization parameter. We set the number of features to 20 and 40 commonly used in literature [1,7]. The number of epochs was set to 10 as the ranked list of items hardly changes after 10 epochs. The regularization was proportional to the support of the given item/user/context. We did not use any other heuristics like time decay to focus on the pure performance of the algorithms. The weights in W were proportional to the number of events belonging to the given cell of the tensor.

We measured recall and precision on the $N = 1, \ldots, 50$ interval. We consider items relevant to a user if the user has at least one event for that item in the test set. Recall@N is the ratio of relevant items on the ranked topN recommendations for the user relative to the number of the user's events in the test set. Precision@N is the ratio of the number of returned relevant items (for each user) and the number of total returned items. Greater values mean better performance.

Table 1 contains recall@20 values for every experiment. Recall@20 is important in practical application as the user usually sees maximum the top 20 items. Using context, the performance is increased overall. The selection of the appropriate context is crucial. In our experiments seasonality improved performance on all datasets. The sequentiality patterns caused large improvements on the Grocery and LastFM 1K datasets (significantly surpassed the results with the seasonality) but did not increased performance on the movie databases (VoD, Netflix, MovieLens). By including seasonality the performance is increased by an average of 30% for the VoD data. This agrees with our assumption that the VoD consumption has a very strong daily repetitiveness and the behavior in different time bands can be well segmented. The results increased by an additional 35% when we used iTALS instead of the context-aware baseline. The genre of the previously watched movies can also improve performance, however its extent is only around 10%. On the other two movie datasets iCA did not improve the performance significantly. We assume that this is due to the explicit–implicit transformation because the transformed implicit feedback is more reliable and also results a sparser tensor. The iTALS using seasonality however could achieve 30% and 80% improvement on Netflix and MovieLens respectively.

Inclusion of the sequentiality patterns increased the performance on Grocery and LastFM 1K datasets by more than 90% and 300% (compared to iALS, recall that no sequential iCA baseline is calculated). Interestingly, the model using the last category is the best with 20 features, but with 40 features the model using last two categories becomes better. We conjecture that this is connected to the greater expressive power of the model with more features. With seasonality the performance also improved by more than 50% and 75%, respectively, on these

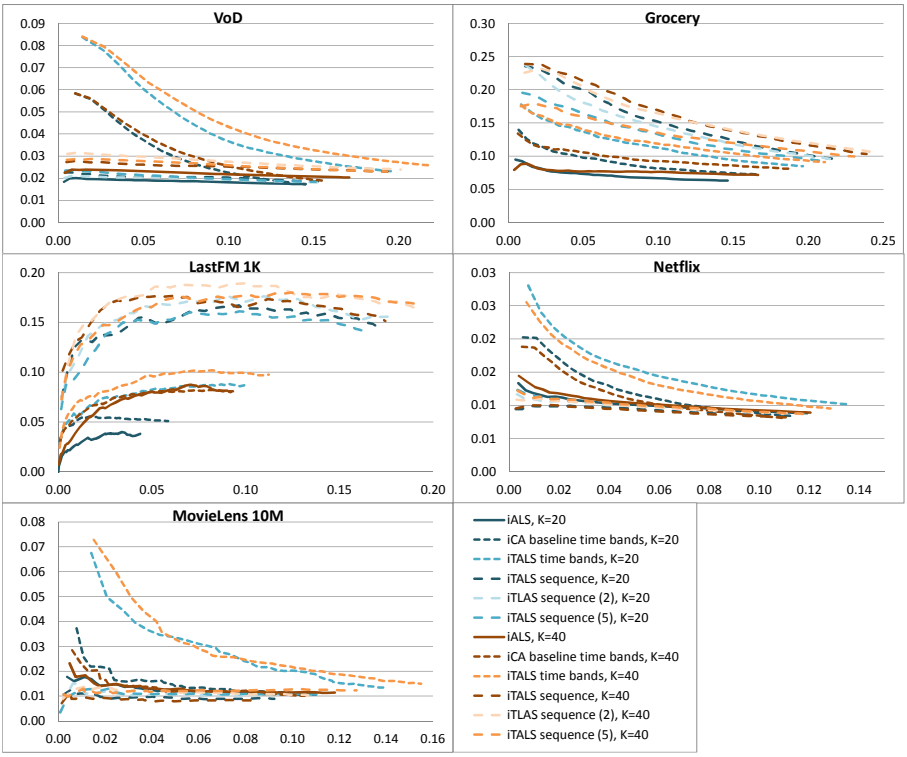

Fig. 1. Precision–recall curves for all datasets and algorithms using factorization with $K = 20$ (blue) and $K = 40$ (orange) factors. The y axis corresponds to precision and x to recall.

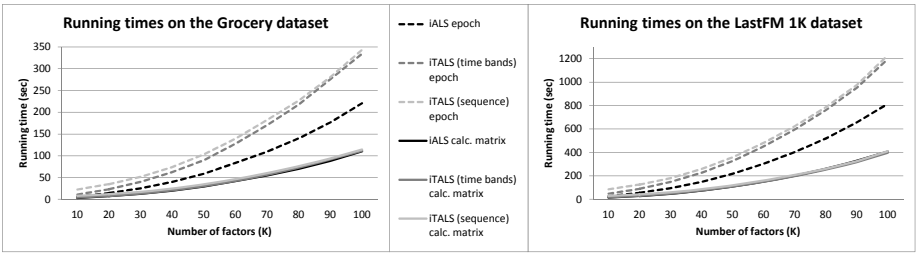

Fig. 2. Running times of iTALS compared to iALS on the Grocery and LastFM 1K datasets

datasets. We expected that the usage pattern learning will perform better on Grocery and LastFM 1K datasets than on the movie datasets as sequentiality is rather important in shopping and music listening than seasonality.

Figure 1 shows the precision–recall curves. The order of the performance of the algorithms is the same as with the recall@20. Observe that the distance

between the curves of the iTALS variants and the curve of the iALS is larger when we use 40 features. Recall that the feature vectors of the context works as a reweighting of the user–item relation. If the resolution of this relation is finer, the reweighting can be more efficient and each factor describes a more specific item property, so the behavior in different context can be described more specifically. Thus, increasing the number of features results in larger performance increase for the context-aware iTALS variants than for iALS.

5.1 Running Times

We compared the running times of the iTALS and iALS algorithms in terms of K (see Figure 2). The experiments were run on a laptop with an Intel Core i5 2410M 2.3GHz processor using only one core. We depict only curves for 2 datasets, since others are similar. We made several runs for each K; the median of the epoch running times are shown (dashed lines). The solid lines show the computation time for one feature matrix. Observe that iTALS scales quadratically with K as iALS; the (re)computation time of one feature matrix is basically the same. Since iTALS recomputes more feature matrices its running time per epoch is larger. Importantly, even if the number of context-states is large (as with the sequential iTALS on LastFM 1K) the $O(K^2)$ part of the complexity remains dominant. This is because the number of non-zero elements in T is much larger than the number of different items/users/contex-states in every case where the usage of context-aware approaches is justified.

6 Conclusion

In this paper we presented an efficient ALS-based tensor factorization method for the context-aware implicit feedback recommendation problem. Our method, coined iTALS, scales linearly with the number of *non-zeroes* in the tensor, thus it works well on implicit data. We presented two specific examples for context-aware implicit scenario with iTALS. When using the seasonality as context, we efficiently segmented periodical user behavior in different time bands. When exploiting sequentiality in the data, the model was able to tell apart items having different repetitiveness in usage pattern. These variants of iTALS allow us to analyze user behavior by integrating arbitrary contextual information within the well-known factorization framework. Experiments performed on five large datasets show that proposed algorithms can greatly improve the performance. Compared to iALS and iCA, our algorithm attained an increase in recall@20 up to 300% and 35%.

One should, however, avoid creating a high dimensional tensors because the number of non-zero elements remains the same no matter how many context types are integrated; so tensors with more dimensions become sparser and thus the results may be poorer than with only a few context dimensions used. Our work opens up a new path for context-aware recommendations in the most common implicit feedback task when only the user history but no rating is available.

Future work will include the characterization of the relation between reweighting, context features and the number of features (K) as well as the design of further context-aware iTALS-based recommendation algorithms.

Acknowledgment. Authors thank Gábor Takács for his very useful comments on the paper.

References

1. Pilászy, I., Tikk, D.: Recommending new movies: Even a few ratings are more valuable than metadata. In: Recsys 2009: ACM Conf. on Recommender Systems, New York, NY, USA, pp. 93–100 (2009)
2. Bennett, J., Lanning, S.: The Netflix Prize. In: KDD Cup Workshop at SIGKDD 2007, San Jose, California, USA, pp. 3–6 (2007)
3. Bell, R.M., Koren, Y.: Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In: ICDM 2007: IEEE Int. Conf. on Data Mining, Omaha, NE, USA, pp. 43–52 (2007)
4. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Major components of the Gravity recommendation system. SIGKDD Explor. Newsl. 9, 80–83 (2007)
5. Pilászy, I., Zibriczky, D., Tikk, D.: Fast ALS-based matrix factorization for explicit and implicit feedback datasets. In: Recsys 2010: ACM Conf. on Recommender Systems, Barcelona, Spain, pp. 71–78 (2010)
6. Takács, G., Pilászy, I., Tikk, D.: Applications of the conjugate gradient method for implicit feedback collaborative filtering. In: RecSys 2011: ACM Conf. on Recommender Systems, Chicago, IL, USA, pp. 297–300 (2011)
7. Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: SIGKDD 2008: ACM Int. Conf. on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, pp. 426–434 (2008)
8. Salakhutdinov, R., Mnih, A.: Probabilistic matrix factorization. In: Platt, J.C., Koller, D., Singer, Y., Roweis, S. (eds.) Advances in Neural Information Processing Systems 20. MIT Press, Cambridge (2008)
9. Ricci, F., Rokach, L., Shapira, B.: Introduction to recommender systems handbook. In: Recommender Systems Handbook, pp. 1–35. Springer, US (2011)
10. Adomavicius, G., Tuzhilin, A.: Context-aware recommender systems. In: Recsys 2008: ACM Conf. on Recommender Systems, Lausanne, Switzerland, pp. 335–336 (2008)
11. Karatzoglou, A., Amatriain, X., Baltrunas, L., Oliver, N.: Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering. In: Recsys 2010: ACM Conf. on Recommender Systems, Barcelona, Spain, pp. 79–86 (2010)
12. Adomavicius, G., Sankaranarayanan, R., Sen, S., Tuzhilin, A.: Incorporating contextual information in recommender systems using a multidimensional approach. ACM Trans. Inf. Syst. 23(1), 103–145 (2005)
13. Adomavicius, G., Ricci, F.: Workshop on context-aware recommender systems (CARS-2009). In: Recsys 2009: ACM Conf. on Recommender Systems, New York, NY, USA, pp. 423–424 (2009)
14. Said, A., Berkovsky, S., De Luca, E.W.: Putting things in context: Challenge on context-aware movie recommendation. In: CAMRa 2010: Workshop on Context-Aware Movie Recommendation, Barcelona, Spain, pp. 2–6 (2010)

15. Bogers, T.: Movie recommendation using random walks over the contextual graph. In: CARS 2010: 2nd Workshop on Context-Aware Recommender Systems, Barcelona, Spain, pp. 1–5 (2010)
16. Baltrunas, L., Amatriain, X.: Towards time-dependant recommendation based on implicit feedback. In: CARS 2009: Workshop on Context-aware Recommender Systems, New York, NY, USA, pp. 1–5 (2009)
17. Bader, R., Neufeld, E., Woerndl, W., Prinz, V.: Context-aware POI recommendations in an automotive scenario using multi-criteria decision making methods. In: CaRR 2011: Workshop on Context-awareness in Retrieval and Recommendation, Palo Alto, CA, USA, pp. 23–30 (2011)
18. He, Q., Pei, J., Kifer, D., Mitra, P., Giles, L.: Context-aware citation recommendation. In: WWW 2010: Int. Conf. on World Wide Web, Raleigh, NC, USA, pp. 421–430 (2010)
19. Panniello, U., Tuzhilin, A., Gorgoglione, M., Palmisano, C., Pedone, A.: Experimental comparison of pre- vs. post-filtering approaches in context-aware recommender systems. In: Recsys 2009: ACM Conf. on Recommender Systems, New York, NY, USA, pp. 265–268 (2009)
20. Lathauwer, L.D., Moor, B.D., Vandewalle, J.: A multilinear singular value decomposition. SIAM J. Matrix Anal. Appl. 21(4), 1253–1278 (2000)
21. Rendle, S., Gantner, Z., Freudenthaler, C., Schmidt-Thieme, L.: Fast context-aware recommendations with factorization machines. In: SIGIR 2011: ACM Int. Conf. on Research and Development in Information, Beijing, China, pp. 635–644 (2011)
22. Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: ICDM 2008: IEEE Int. Conf. on Data Mining, Pisa, Italy, pp. 263–272 (2008)
23. Liu, N.N., Cao, B., Zhao, M., Yang, Q.: Adapting neighborhood and matrix factorization models for context aware recommendation. In: CAMRa 2010: Workshop on Context-Aware Movie Recommendation, Barcelona, Spain, pp. 7–13 (2010)
24. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: SIGMOD 1993: ACM SIGMOD Int. Conf. on Management of Data, Washington DC, USA, pp. 207–216 (1993)
25. Davidson, J., Liebald, B., Liu, J., et al.: The YouTube video recommendation system. In: Recsys 2010: ACM Conf. on Recommender Systems, Barcelona, Spain, pp. 293–296 (2010)
26. Celma, O.: Music Recommendation and Discovery in the Long Tail. Springer (2010)
27. GroupLens Research: Movielens data sets (2006), `http://www.grouplens.org/node/73`