Speeding up ALS learning via approximate methods for context-aware recommendations

Balázs Hidasi[1,2], Domonkos Tikk[1]

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

Abstract. Implicit feedback based recommendation problems, typically set in real-world applications, recently have been receiving more attention in the research community. From the practical point of view, scalability of such methods is crucial. However, factorization based algorithms efficient in explicit rating data applied directly to implicit data are computationally inefficient, therefore different techniques are needed to adapt to implicit feedback. For alternating least squares (ALS) learning, several research contributions have proposed efficient adaptation techniques for implicit feedback. These algorithms scale linearly with the number of non-zero data points, but cubically in the number of features, which is a computational bottleneck that prevents the efficient usage of accurate high factor models. Also, map-reduce type big data techniques are not viable with ALS-learning, because there is no known technique that solves the high communication overhead required for random access of the feature matrices. To overcome this drawback here we present two generic approximate variants for fast ALS-learning, using conjugate gradient (CG) and coordinate descent (CD). Both CG and CD can be coupled with all methods using ALS-learning. We demonstrate the advantages of fast ALS variants on iTALS, a generic context-aware algorithm, which applies ALS-learning for tensor factorization on implicit data. In the experiments, we compare the approximate techniques with the base ALS-learning in terms of training time, scalability, recommendation accuracy and convergence. We show that the proposed solutions offer a trade-off between recommendation accuracy and speed of training time; this makes it possible to apply ALS-based methods efficiently even for billions of data points.

Keywords: recommender systems, tensor factorization, context awareness, implicit feedback, scalability, comparison

1. Introduction

In this paper we investigate a class of practically important latent factor based collaborative filtering recommendation algorithms that use alternating least squares (ALS) based learning, and propose speed-up techniques that enable trade-off between recommendation accuracy and training time. Now, we first characterize this practically important recommendation problem class in terms of input data, scalability, and context-awareness.

Recommender systems (Ricci et al. 2011) are information filtering tools that help users suffering information overload to find interesting items (products, content, etc.). Here we focus on the class of latent factor based collaborative filtering (CF) methods that gained popularity due to their good *accuracy* and *scalability* (Koren & Bell 2011). They capture the users' preferences by uncovering latent features that explain the observed user–item ratings using factor models.

In most practical scenarios, however, users do not rate content/items explicitly: one can only observe the users' interactions[1] with items—retrieved from web logs, for instance—as they use the system. This type of feedback is termed implicit feedback, also called one-class CF in the literature, and contains unary data, i.e. logged user–item interactions. Implicit feedback data contains less information on user preferences than explicit feedback, and it exhibits the problem of no negative feedback and inherent noisiness (Hu et al. 2008).

Context-aware recommendation systems (CARS) refine recommendations by considering additional information, available to the system. They extend the dualistic user–item modeling concept and consider additional information that may influence the user preferences at recommendation. Such data are together termed *contextual information*, or briefly *context* (Adomavicius & Tuzhilin 2008). One class of CARS uses latent factor methods (see e.g. Karatzoglou et al. (2010), Rendle & Schmidt-Thieme (2010), Hidasi & Tikk (2012), Shi et al. (2012), Rendle (2012)). For demonstrating our proposed methods, in this paper we will use iTALS (Hidasi & Tikk 2012) that is a generic context-aware algorithm, which applies ALS-learning for tensor factorization on implicit data.

For practical applicability, beyond the implicit feedback and context-awareness, the training time of the algorithms is key aspect. Faster training allows to (1) capture a more recent state of the system modeled (advantageous for any system, but required for ones where the lifetime of the items is short or new items appear constantly); (2) retrain the models more frequently; (3) apply trade-off between running times and accuracy by using more features or running more epochs.

A straightforward way of speeding up training – without any modification on the base algorithm – is to distribute computations between multiple processing units (e.g. processor cores, machines). A considerable advantage of most ALS based methods is that the majority of computations are independent and therefore can be done simultaneously. With iTALS, the feature vectors of a dimension are computed independently (see Section 3), therefore the degree of parallelization can be as high as the number of entities (users, items, context-states). Since the computation for one entity is fast, the method scales well. However, ALS-based methods (including iTALS) require that at least the model (feature matrices) are stored in memory and each processing unit has access to

[1] User purchased an item or viewed an product page, etc. Interactions also called events or transactions.

this shared memory.[2] Otherwise a huge communication overhead arises, since the computations require random access to the feature matrices. This also implies that ALS does not work well with standard map-reduce based big data technologies (Balassi et al. 2014), but requires a different solution (e.g.: multi-core/multiprocessor machine, GPGPU, multi-GPU systems, cluster with shared memory, etc.).

We argue that models and data fit in memory in most cases. With the indexing overhead required by iTALS, $\sim 45\,\mathrm{M}$ 3-tuple records can be stored in 1 GB. The models take even less space: low and high factor models ($K = 40$) would require 21.36 MB and 106.81 MB,[3] respectively. Today's normal PCs therefore can handle around 1 billion record, their high end counterparts can deal with several billions and shared memory clusters are capable of working with tens or even hundreds of billion events. Thus this approach is feasible for most of the recommendation tasks.

On the other hand, there is room for improvement beyond distributing the computations. Distributed computation does not decrease the total load on the infrastructure and the training might still take long due to some computationally expensive steps in the algorithm. Usually the CPU is the bottleneck for such algorithms, in contrast to the classic big data problem. ALS slows down significantly if the number of features (K) is high (see Section 3.1). This prevents the efficient usage of high factor models. High factor models are generally more accurate than low factor ones therefore it would be beneficial to use them.

In this paper, we propose two approximate methods that significantly speed up ALS-learning, especially if the number of features is high, that is, the gain in speed increases as the number of features increases.

The proposed methods allow for a trade-off between speed and recommendation accuracy. One can either train a model with the same accuracy in significantly less time, train a model with more features (and thus be more accurate) with the same training time, or anything in-between. Our solution does not address the incompatibility between ALS and map-reduce based big data technology, we instead offer feasible trade-off solutions using approximate methods to reduce the computational complexity of ALS-learning.

Our main contributions are: (1) the two approximate variants for ALS-learning using coordinate descent and conjugate gradient methods; (2) we show that these enable faster training and improved scalability for iTALS and every other ALS-based factorization methods; (3) in our experiments we point out that the proposed approximate methods offer trade-off between recommendation accuracy and training time.

The paper is organized as follows. Section 2 briefly reviews related work in the field of recommendation systems on context-awareness, tensor factorization, and implicit feedback algorithms. In Section 3 we review the iTALS algorithm, including its application (3.2) of the implicit feedback based context-aware recommendation problem and analyze its complexity (3.1). We introduce the two approximate methods for ALS and extend them to be usable with iTALS and other ALS-methods using complex models in Section 4. Section 5 explains our

[2] It is beneficial if the data is stored in the shared memory as well, but it can be stored on disk as well, if properly indexed.

[3] Here we assumed a relatively high density of $\sim 1\%$, $100K$ for users and $45K$ for items that is realistic for $\sim 45\,\mathrm{M}$ record.

experimental setup and describes the context dimensions we used in our experiments. The analysis and comparison of our methods is presented in Section 6. The work is concluded in Section 7.

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

$A \circ B \circ \ldots$	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 index $(i, j, k, \ldots)$ is the product of the element of A, B, ... at index $(i, j, k, \ldots)$.
A_i	The i^{th} column of matrix A.
$A_{i_1,i_2,\ldots}$	The $(i_1, i_2, \ldots)$ element of tensor/matrix A.
K	The number of features, the main parameter of the factorization.
D	The number of dimensions of the tensor.
T	A D dimensional tensor that contains only zeroes and ones (preference tensor).
W	A tensor with the same size as T (weight tensor).
S_i	The size of T in the i^{th} dimension $(i = 1, \ldots, D)$.
N^+	The number of ratings (explicit case); non-zero elements in tensor T (implicit case).
$M^{(i)}$	A $K \times S_i$ sized matrix. Its columns are the feature vectors for the entities in the i^{th} dimension.
$A_{j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_D}$	denotes an element of tensor A where the index in the i^{th} dimension is fixed to j, and other indices are arbitrary.
N_I	Number of inner iterations.

2. Related work

Context-aware recommender systems (Adomavicius et al. 2005) have emerged as an important research area in the last five years and entire workshops are devoted to this topic on major conferences: CARS series started in 2009, (Adomavicius & Ricci 2009); CAMRA in 2010, (Said et al. 2010). The application fields of context-aware recommenders include among others: point-of-interest (Bader et al. 2011), video (Zarka et al. 2012), music (Dias & Fonseca 2013) and news recommendation (Lommatzsch 2014). Context-aware recommender approaches can be classified into three main groups: pre-filtering, post-filtering and contextual modeling (Adomavicius & Tuzhilin 2008). Pre-filtering methods partition data by the value of the context variables and use traditional models on these segments. Post-filtering ignores the contextual data at recommendation generation, but filters out irrelevant items (in a given context) or adjust recommendation score (according to the context) when the recommendation list is prepared. Contextual modeling approaches directly incorporate the context into the model and learn the relation between users, items and context-states. Tensor factorization based solutions, including our approach, falls into the contextual modeling category.

Tensor factorization (TF) 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 $D = C+2$ dimensional tensor, T. The first dimension corresponds to users, the second to items and the subsequent C dimensions $[3, \ldots, D]$ are devoted to contexts. TF methods approximate this tensor via lower rank approximation w.r.t. a loss function (usually some kind of RMSE – root mean squared error – variant). In (Karat-

zoglou et al. 2010), a sparse HOSVD (Lathauwer et al. 2000) method is presented for the explicit context aware case, which decomposes a D dimensional sparse tensor into D (low rank) matrices and a D dimensional (low rank) 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 $S_1 \times K$, $S_2 \times K$, ..., $S_D \times K$ and the size of the tensor is $K \times \cdots \times K$. The authors use gradient descent to learn the model. The complexity of one training iteration scales *linearly* with the number of ratings (N^+) and *cubically* with the number of features (K), which is a large improvement compared to the dense HOSVD's $O(K \cdot (S_1 + \cdots + S_D)^D)$. A further improvement was proposed by (Rendle et al. 2011): their factorization machine (FM) scales linearly *both* N^+ and K. However, if the original tensor is large and dense like for the implicit recommendation task then neither method scales well, because $N^+ = S_1 \cdots S_D$.

The two main approaches for ranking in the implicit w.r.t. loss functions are pointwise and pairwise ranking. However, the naive minimization of the objective function in the implicit case is typically expensive, as it scales with the size of the user–item matrix. There are two dominant approaches to overcome this difficulty: (1) the trade-off solution that sacrifices the accuracy to some extent for computational efficiency by sampling the data (usually the missing "negative" feedback is sampled); (2) the direct minimization of the objective function without sampling by decomposing the calculation to independent parts.

For pointwise ranking and direct minimization, the seminal work was proposed by (Hu et al. 2008); their implicit ALS (iALS) applies an alternating least squares optimization and decomposes the derivatives of the objective function to user-dependent and item-independent parts, hence the complexity of a training iteration is reduced to scale *linearly* with the number of events (positive feedbacks, N^+). (Pan et al. 2008) proposed two prediction based frameworks for handling implicit feedback. The first one is similar to iALS, but it contains a naive ALS optimizer instead of a tuned one. The second one is based on negative example sampling.

For pairwise ranking, (Jahrer & Töscher 2011) applied a stochastic gradient descent (SGD) optimizer on a sampled approximation of the objective function, while (Takács & Tikk 2012) proposed a direct minimization for the same objective function with an appropriate decomposition of the derivatives. Another ranking based approach for implicit feedback is the Bayesian personalized ranking (BPR; Rendle et al. (2009)), where objective function of BPR is derived from the Bayesian analysis of the problem. The optimization technique used for training is SGD.

Only a few factorization algorithms can handle both implicit feedback and context. Both iTALS (Hidasi & Tikk 2012) and its variant iTALSx (Hidasi 2014) use pointwise ranking and direct minimization with ALS-learning. TFMAP (Shi et al. 2012) aims to maximize mean average precision (MAP) through pairwise ranking and sampling. It is suggested by (Rendle 2012) that Factorization Machines (Rendle et al. 2011) can be used for the implicit case with BPR as the objective function, however it is not elaborated and sampling for BPR is not trivial when $D > 2$. A very recent method (Nguyen et al. 2014) uses Gaussian processes and can be applied for both explicit and implicit cases.

Our proposed speed-up techniques are applicable to all methods that use ALS-learning, and they also provide trade-off between accuracy and training time.

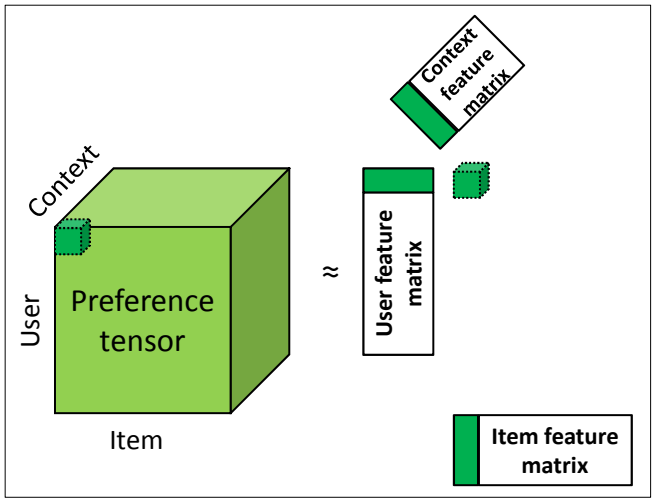

Fig. 1. Concept of the tensor decomposition in 3 dimension with the classical user–item–context setting.

3. Review of iTALS

In this section we review 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 the algorithm suitable to handle the context-aware implicit recommendation problem.[4] iTALS uses pointwise ranking through weighted RMSE (wRMSE) based loss function and directly minimizes said loss function. Learning efficiency is guaranteed by the careful decomposition of the gradient into independent computations.

Let T be a tensor of zeroes and ones and let W contain weights for each element of T. Let $T_{u,i,c_3,\cdots,c_D} = 1$ if user u has (at least one) event on item i while the context-state of j^{th} context dimension was c_j. Due to its construction, most elements of T are zeros. The weight matrix W takes element w_0 if the corresponding element in T is 0, and is set to be (much) greater than w_0 otherwise. Instead of using the form of the common HOSVD decomposition (D matrices and a D dimensional tensor) our proposed model (see eq. (1)) approximates T by a decomposition into D matrices. The size of the matrices are $K \times S_1, K \times S_2, \ldots, K \times S_D$. (See figure 1.) The approximation of a given element of T is the elementwise product of columns from $M^{(i)}$ low rank matrices:

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

We want to minimize the following loss function (wRMSE):

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

[4] With proper weighting scheme, the iTALS could be used with explicit feedback as well.

The loss function L is minimized by alternating least squares (ALS), that is, all but one of the $M^{(i)}$ matrices are fixed (without the loss of generality, next step are shown for $M^{(1)}$). L is convex in the non-fixed variables. L reaches its minimum in $M^{(1)}$, where its derivative with respect to $M^{(1)}$ is zero. Since the derivative of L is linear in $M^{(1)}$, the columns of the matrix can be computed separately. That is for the $(i_1)^{\text{th}}$ column of $M^{(1)}$:

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

While $\mathcal{O}$ can be computed efficiently, calculation of $\mathcal{I}$ is expensive, because it requires summing over all possible configurations in which the current entity—whose feature vector is being computed; the first entity of the first dimension in this case—takes part. For all columns of a feature matrix, this would result in a complexity that is the product of the number of columns of all feature matrices. Therefore we transform $\mathcal{I}$ by using $W_{i_1,i_2,\ldots,i_D} = W'_{i_1,i_2,\ldots,i_D} + w_0$ and get:

$$\mathcal{I} = \sum_{i_2=1,\ldots,i_D=1}^{S_2,\ldots,S_D} W'_{i_1,i_2,\ldots,i_D} \left(M^{(2)}_{i_2} \circ \cdots \circ M^{(D)}_{i_D} \right) \left(M^{(2)}_{i_2} \circ \cdots \circ M^{(D)}_{i_D} \right)^T + \\ + \underbrace{w_0 \sum_{i_2=1,\ldots,i_D=1}^{S_2,\ldots S_D} \left(M^{(2)}_{i_2} \circ \cdots \circ M^{(D)}_{i_D} \right) \left(M^{(2)}_{i_2} \circ \cdots \circ M^{(D)}_{i_D} \right)^T}_{\mathcal{J}} \tag{4}$$

The transformation enables more efficient computations. First, because the first sum in equation (4) is tractable, as $W'_{i_1,i_2,\ldots,i_D}$ is zero for those indices where $T_{i_1,i_2,\ldots,i_D}$ is zero, thus it scales with the number of transactions the actual entity is involved with. A similar decomposition step is applied in iALS (Hu et al. 2008). Second, $\mathcal{J}$ is the same for all columns of $M^{(1)}$, thus can be precomputed efficiently as follows. (See the formal complexity analysis in Section 3.1.)

$$\mathcal{J} = w_0 \sum_{i_2=1,\ldots,i_D=1}^{S_2,\ldots,S_D} \left(M^{(2)}_{i_2} \circ \cdots \circ M^{(D)}_{i_D} \right) \left(M^{(2)}_{i_2} \circ \cdots \circ M^{(D)}_{i_D} \right)^T = \\ = w_0 \underbrace{\left(\sum_{i_2=1}^{S_2} M^{(2)}_{i_2} \left(M^{(2)}_{i_2} \right)^T \right)}_{\mathcal{M}^{(2)}} \circ \cdots \circ \underbrace{\left(\sum_{i_D=1}^{S_D} M^{(D)}_{i_D} \left(M^{(D)}_{i_D} \right)^T \right)}_{\mathcal{M}^{(D)}} \tag{5}$$

where we used that each element of $\mathcal{J}$ is computed as:

$$\begin{aligned}\mathcal{J}_{j,k} =& w_0 \left(\sum_{i_2=1,\ldots,i_D=1}^{S_2,\ldots,S_D} \left(M_{i_2}^{(2)} \circ \cdots \circ M_{i_D}^{(D)}\right) \left(M_{i_2}^{(2)} \circ \cdots \circ M_{i_D}^{(D)}\right)^T \right)_{j,k} = \\ =& w_0 \sum_{i_2=1,\ldots,i_D=1}^{S_2,\ldots,S_D} \left(M_{j,i_2}^{(2)} \cdot \ldots \cdot M_{j,i_D}^{(D)}\right) \left(M_{k,i_2}^{(2)} \cdot \ldots \cdot M_{k,i_D}^{(D)}\right) = \\ =& w_0 \left(\sum_{i_2=1}^{S_2} M_{j,i_2}^{(2)} M_{k,i_2}^{(2)} \right) \cdot \ldots \cdot \left(\sum_{i_D=1}^{S_D} M_{j,i_D}^{(D)} M_{k,i_D}^{(D)} \right)\end{aligned} \tag{6}$$

Algorithm 3.1 Fast ALS-based tensor factorization for implicit feedback

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; λ: regularization coefficient
Output: $\{M^{(i)}\}_{i=1,\ldots,D}$ $K \times S_i$ sized low rank matrices
procedure iTALS(T, W, K, E, λ)

```
1:  for i = 1, ..., D do
2:      M^(i) ← Random K × S_i sized matrix
3:      M^(i) ← M^(i)(M^(i))^T
4:  end for
5:  for e = 1, ..., E do
6:      for i = 1, ..., D do
7:          C^(i) ← w_0 M^(1) ∘ ··· ∘ M^(i−1) ∘ M^(i+1) ··· ∘ M^(D)
8:          O^(i) ← 0
9:          for j = 1, ..., S_i do
10:             C^(i,j) ← C^(i)
11:             O^(i,j) ← O^(i)
12:             for all {t | t = T_{j_1,...,j_{i−1},j,j_{i+1},...,j_D}, t ≠ 0} do
13:                 W_t ← W_{j_1,...,j_{i−1},j,j_{i+1},...,j_D} − w_0
14:                 v ← M^(1)_{j_1} ∘ ··· ∘ M^(i−1)_{j_{i−1}} ∘ M^(i+1)_{j_{i+1}} ∘ ··· ∘ M^(D)_{j_D}
15:                 C^(i,j) ← C^(i,j) + W_t v v^T
16:                 O^(i,j) ← O^(i,j) + W_t v
17:             end for
18:             M^(i)_j ← (C^(i,j) + λI)^{−1} O^(i,j)
19:         end for
20:         M^(i) ← M^(i)(M^(i))^T
21:     end for
22: end for
23: return {M^(i)}_{i=1,...,D}
```

end procedure

The pseudocode of the proposed iTALS (Tensor factorization using ALS for implicit recommendation problem) is given in Algorithm 3.1. The pseudocode follows the deduction above. In line 3 we precompute $\mathcal{M}^{(i)}$ introduced in eq. (5). We create the column independent part of eq. (4) in line 7. We add the column dependent parts of eq. (3) in lines 12–17 and compute the desired column (with

regularization) 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.

Please note that a few features such as biases and regularization were omitted for clearer presentation. These features can be easily incorporated into the algorithm and their inclusion does not change the its scalability or structure. The aforementioned features are in fact available in the implementation we use.

3.1. Complexity

The computational cost of one epoch (lines 6–21) is $O(DN^+K^2 + K^3\sum_{i=1}^{D} S_i)$. As shown above, $\mathcal{J}$ from eq. (5) is the same for each column, therefore its calculation is needed only once for each $M^{(i)}$ matrix, which takes $O(DK^2)$ time (see eq. (6)). To calculate $\mathcal{I}$ from eq. (4) for the j^{th} column, we need the precomputed $\mathcal{J}$, as well as $O(N_j^+K^2)$ steps, where N_j^+ is the non-zero elements in T with fixed j dimension (the cardinality of the set in line 12). Although, the first sum of eq. (4) runs over all entities of all but one dimension of T, $W'_{j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_D}$ was constructed to be zero unless the respective element of T is non-zero. To compute the partial derivative of the loss function with respect to a column of the non-fixed matrix $M^{(i)}$, we also need $\mathcal{O}$ (see eq. (3)). It is calculated in $O(N_j^+K)$ time because most of $T_{j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_D}$ are zero. Finally, we have to compute the actual feature vector that requires the inversion of a $K \times K$ matrix ($O(K^3)$). After all columns of the matrix are recomputed $\mathcal{M}^{(i)}$ also needs to be recomputed that takes $O(S_iK^2)$ time.

Summing these complexities for one $M^{(i)}$ matrix we need $O(S_iK^3 + DK^2 + \sum_{j=1}^{S_i} N_j^+ \left(K + K^2\right) + S_iK^2)$ time that can be simplified to $O(N^+K^2 + S_iK^3)$ (assuming that $D \ll S_i \ll N^+$, that is the case when the data is not too sparse). Summing this cost for all matrices we get $O(DN^+K^2 + K^3\sum_{i=1}^{D} S_i)$, which is cubical in K, the number of features, and linear in N^+, the number of non-zero elements of the tensor. In practical cases $DN^+ \gg \sum_{i=1}^{D} S_i$ (i.e. the data is not too sparse[5]), therefore the first term is dominant, thus the method scales *quadratically* in K for smaller K values (also common in practice; see section 6.2 for actual running time measurements).

3.2. Application to implicit feedback based context-aware problems

In order to apply the algorithm to the implicit feedback based context-aware recommendation problem, all we need to do is to create T and W tensors, based on the training data. The training data contains the interactions of the users and items, and the context of these interactions. First, we should select the context-dimensions to be used. We may also transform or combine the original contextual attributes to derive new attributes (e.g. seasonality from timestamps,

[5] $DN^+ = \sum_{i=1}^{D} S_i$ means that we only have one event/example for each user, for each item and each context-state. In this case, CF method are not applicable due to sparseness.

see Section 5.1.1). Then, the transformed training data contains tuples of D length. $T_{u,i,c_3,\ldots,c_D}$ is 1 if the training data contains the tuple $(u, i, c_3, \ldots, c_D)$. Multiple occurrences of the same tuple does not modify the value. Every other elements of T are set to 0. The elements of W corresponding to the zero elements of T are set to w_0. An empirically good choice for w_0 is 1. The rest of the elements of W should be set to a greater value than w_0. In practice, it usually works well if these cells are set $\alpha \cdot \#(u, i, c_3, \ldots, c_D)$, that is the number occurrences of the corresponding tuple in the training data multiplied by a constant (we use $\alpha = 100$ in the experimentation).

This representation of the problem intuitively means that users prefer items they already interacted against unseen items. However, we are more certain in the former, because an interaction is (a somewhat noisy) sign of preference, while the absence of an interaction does not necessarily mean dislike. The uniform weighting of missing feedback used here is conform to the (implicit version) of missing data completely at random (Little & Rubin 1987) assumption. However, different weighting schemes (e.g. activity based, category based, etc.) can be easily applied without degrading scalability and thus the algorithm can be made conform with the missing at random or the missing at not random assumptions. The application of such weighting schemes is outside of the scope of this paper.

The scalability of the algorithm is good for this problem, because the number of non-zero elements in T equals to the number of (unique) events in the training data. This is very beneficial, because training data is usually very sparse (density is usually $< 1\%$). Although, the training time also depends on the number of entities in each dimension (e.g. users, items, etc.), the dominant member of the complexity (for smaller K values) is $K^2 N^+$ and thus the training time is linear in the size of the training data.

3.3. iTALSx – a variant with pairwise preference model

A variant of iTALS using pairwise preference model is called iTALSx (Hidasi 2014). Here the elements of T are approximated by a pairwise interaction model instead of the elementwise (n-way) product. The model is similar to the one used in (Rendle & Schmidt-Thieme 2010) and (Rendle 2012), however, its extension to D dimensions does not contain the pairwise interactions between context dimensions:

$$\begin{aligned}\hat{T}_{i_1,i_2,\ldots,i_D} = 1^T (M^{(1)}_{i_1} \circ M^{(2)}_{i_2} + M^{(1)}_{i_1} \circ M^{(3)}_{i_3} + \ldots + M^{(1)}_{i_1} \circ M^{(D)}_{i_D} + \\ + M^{(2)}_{i_2} \circ M^{(3)}_{i_3} + \ldots + M^{(2)}_{i_2} \circ M^{(D)}_{i_D}).\end{aligned} \tag{7}$$

The altered preference model requires different decomposition steps to be computationally efficient. iTALSx performs better if the number of features is lower or if the dataset is very sparse (i.e. few events, lots of items and users), but it falls behind iTALS if K or the number of events is sufficiently high (Hidasi 2014).

4. Approximate solutions for ALS

Recall that except for the matrix inversion, ALS based algorithms scales quadratically with K. For iTALS, the DN^+K^2 term dominates the $K^3 \sum_{i=1}^{D} S_i$ term

in the complexity when we use low-factor models, when $DN^+ \gg \sum_{i=1}^{D} S_i$. The computation of the cubical term can still take a long time, especially with higher K values or more context dimensions. Therefore, we introduce two approximate solutions and adapt them to iTALS to further reduce its time complexity: iTALS-CD applies the coordinate descent learning for iTALS, while iTALS-CG adapts the conjugate gradient descent method. The adaptations are generalizations of the techniques proposed for matrix factorization in (Pilászy et al. 2010) and (Takács et al. 2011). This generalization is not exclusive to iTALS, other ALS based factorization algorithms can benefit from the direct application of these methods. The approximate methods are presented with iTALS, using it as an example. They can be easily applied to other methods using similar steps.

We will show that the approximate variants provide a trade-off: they can achieve lower running times (see Section 6.2) in exchange for somewhat higher loss function values. Note that higher loss function values are not necessarily translated to lower accuracy in recommendations when one applies other, non-error based metrics (classification or ranking metrics) for the evaluation (see Section 6.1).

4.1. Coordinate descent

The coordinate descent (CD) approach approximates the feature vector by computing its coordinates separately. CD approximates the least squares solution of a $b = Ax$ linear system (seeking x). By fixing all but one feature and computing the remaining one, the matrix inversion can be avoided (it is reduced to a division) thus the computation time can be greatly reduced (see Algorithm 4.1). Note that while the solution provided by CD can be good enough, it does not converge to the least squares solution.

Algorithm 4.1 Weighted coordinate descent method

Input: A: $N_E \times K$ matrix of input examples; b: output for the examples; $x^{(0)}$: initial solution; w: vector of weights; λ: regularization coefficient; N_I: number of iterations
Output: x: approximate solution of $Ax = b$
procedure SOLVE-WEIGHTED-CD(A, b, x_0, w, λ, N_I)

1: $x \leftarrow x^{(0)}$
2: **for** $i = 1, \ldots, N_I$ **do**
3: **for** $j = 1, \ldots, N_E$ **do**
4: $\Delta x_j \leftarrow \frac{A_j^T(w \circ b) - \lambda x_j}{A_j^T(w \circ A_j) + \lambda}$
5: $x_j \leftarrow x_j + \delta x_j$
6: **end for**
7: **end for**
8: **return** x

end procedure

The biggest difficulty to adapt CD to the iTALS framework (or to any other complex models) is posed by the extremely high number of examples that corresponds to the number of rows of A. This quantity is the product of the sizes of all but one dimension, that is $N_E = \prod_{j=1, j \neq i}^{D} S_j$, when the feature vector of the

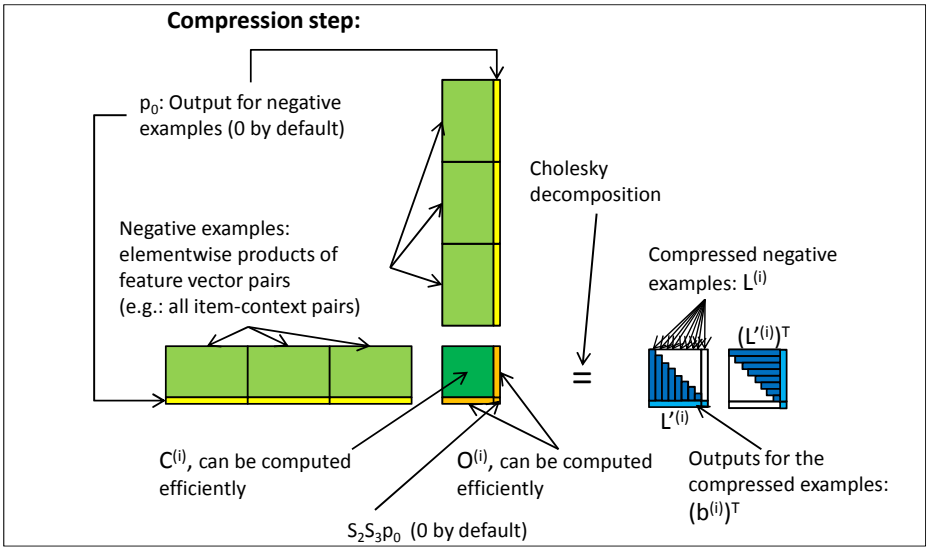

Fig. 2. Concept of the compression of negative examples in the 3 dimensional user–item–context setting.

i^{th} dimension is sought. This quantity can be greatly reduced by compressing the information of the negative feedbacks as shown next (also see figure 2):

- Compute the shared part of eq. (4): $C^{(i)}$; and $O^{(i)}$. Note that $C^{(i)}$ is in fact the covariance of all possible negative examples (i.e. the Hadamard product of all combinations of feature vectors from all but the i^{th} dimension). Similarly $O^{(i)}$ is the covariance with the output, but since missing events are represented by zeroes in the tensor, $O^{(i)}$ is a vector of zeroes.
- $O^{(i)}$ is appended to $C^{(i)}$ from the right and $\left(O^{(i)}\right)^T$ from the bottom. Thus we get a $(K+1)\times(K+1)$ sized matrix: $C'^{(i)}$. The $(K+1, K+1)$ element of $C'^{(i)}$ is set to $p_0 \prod_{j=1, j\neq i}^{D} S_j$, where p_0 is the value associated with the negative preference ($p_0 = 0$ by default). $C'^{(i)}$ is symmetric and positive definite. This step is needed because the input and the output must be compressed simultaneously.
- $C'^{(i)}$ is decomposed into $L'^{(i)}\left(L'^{(i)}\right)^T$ using Cholesky decomposition. $L'^{(i)}$ is a lower triangular matrix. The decomposition requires $O(K^3)$, but has to be computed only once per recomputing a feature matrix.
- The columns of $L'^{(i)}$ are the compressed negative examples. The first K coordinates of a column form the example and the last coordinate is the output for that input.

The number of examples for the negative feedback was compressed into $K+1$ examples, that is shared for every feature vector of the i^{th} matrix. For the j^{th} feature vector we also need the positive feedback for the j^{th} entity, but their number is low (N_j^+), therefore the coordinate descent method can be computed efficiently.

Algorithm 4.2 iTALS using coordinate descent for speedup

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; λ: regularization coefficient; N_I: number of inner iterations of the CD method
Output: $\{M^{(i)}\}_{i=1,\ldots,D}$ $K \times S_i$ sized low rank matrices
procedure iTALS(T, W, K, E, λ, N_I)

1: **for** $i = 1, \ldots, D$ **do**
2: $M^{(i)} \leftarrow$ Random $K \times S_i$ sized matrix
3: $\mathcal{M}^{(i)} \leftarrow M^{(i)}(M^{(i)})^T$
4: **end for**
5: **for** $e = 1, \ldots, E$ **do**
6: **for** $i = 1, \ldots, D$ **do**
7: $C^{(i)} \leftarrow w_0 \mathcal{M}^{(1)} \circ \cdots \circ \mathcal{M}^{(i-1)} \circ \mathcal{M}^{(i+1)} \cdots \circ \mathcal{M}^{(D)}$
8: $O^{(i)} \leftarrow 0$
9: $C'^{(i)} \leftarrow$ append $O^{(i)}$ to $C^{(i)}$ from right and $\left(O^{(i)}\right)^T$ from bottom
10: $C'^{(i)}_{K+1,K+1} \leftarrow 0$
11: $L'^{(i)} \left(L'^{(i)}\right)^T \leftarrow$ Cholesky-decomposition$(C'^{(i)})$
12: $L^{(i)} \leftarrow$ strip the last row of $L'^{(i)}$
13: $b^{(i)} \leftarrow$ the last row of $L'^{(i)}$ transposed
14: **for** $j = 1, \ldots, S_i$ **do**
15: $L^{(i,j)} \leftarrow L^{(i)}$
16: $b^{(i,j)} \leftarrow b^{(i)}$
17: $w^{(i,j)} \leftarrow$ vector of w_0 values; same length as $b^{(i,j)}$
18: **for all** $\{t \mid t = T_{j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_D}, t \neq 0\}$ **do**
19: $w'_t \leftarrow W_{j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_D} - w_0$
20: $v \leftarrow M^{(1)}_{j_1} \circ \cdots \circ M^{(i-1)}_{j_{1-1}} \circ M^{(i+1)}_{j_{i+1}} \circ \cdots \circ M^{(D)}_{j_D}$
21: $L^{(i,j)} \leftarrow$ append v to $L^{(i,j)}$ from right
22: $b^{(i,j)} \leftarrow$ append 1 to $b^{(i,j)}$
23: $w^{(i,j)} \leftarrow$ append w'_t to $w^{(i,j)}$
24: **end for**
25: $M^{(i)}_j \leftarrow$ Solve-weighted-CD$(L^{(i,j)}, b^{(i,j)}, M^{(i)}_j, w^{(i,j)}, \lambda, N_I)$
26: **end for**
27: $\mathcal{M}^{(i)} \leftarrow M^{(i)}(M^{(i)})^T$
28: **end for**
29: **end for**
30: **return** $\{M^{(i)}\}_{i=1,\ldots,D}$
end procedure

Algorithm 4.2 shows the pseudocode for iTALS-CD. It is identical with algorithm 3.1 until line 8. In lines 9–13 the negative examples are compressed. We also need a weight vector because we optimize for wRMSE (line 17). Updating steps of this matrix and vectors with positive examples are executed in lines 18–24. The solution for $M^{(i)}_j$ is computed in line 25 using a weighted coordinate descent method (see Algorithm 4.1). The signature of the solver is Solve-weighted-CD(A, b, x_0, w, λ, N_I), where the linear system is $A^T x = b$, x_0 is an initial

solution, the error is weighted by weight vector w as $(\left|\left|w \circ \left(b - A^T x\right)\right|\right)$, λ is the regularization parameter and N_I is the number of iterations.

The complexity of computing the i^{th} matrix consists of the following parts: computing $C^{(i)}$ in $O(K^2 D)$, $L^{(i)}$ in $O(K^3)$, the computation of all positive examples in $O(N^+ K)$, optimizing using CD[6] in $O(N_I(S_i K^2 + N^+ K))$ and finally recomputing $\mathcal{M}^{(i)}$ in $O(S_i K^2)$ time. This sums up to $O(K^3 + N_I S_i K^2 + N_I N^+ K)$ for the i^{th} matrix (assuming that $D \ll N_I S_i$). Therefore the cost of one epoch is $O(DK^3 + N_I K^2 \sum_{i=1}^{D} S_i + DN^+ N_I K)$. Comparing this to the complexity of iTALS ($O(DN^+ K^2 + K^3 \sum_{i=1}^{D} S_i)$) we can observe that iTALS-CD also scales cubically in K, however, the coefficient is reduced from $\sum_{i=1}^{D} S_i$ to D. The other two terms are similar, however there is $N_I K^2$ and $N_I K$ in the place of K^3 and K^2. In practice, when $N_I \ll K$ and D is low, for smaller K values it scales *linearly* in K because $DN^+ \gg \sum_{i=1}^{D} S_i$.

4.2. Conjugate gradient

The conjugate gradient (CG; (Hestenes & Stiefel 1952)) method is the state-of-the-art iterative method for solving $Ax = b$ type systems of linear equations, where A is symmetric positive definite (see Algorithm 4.3). The geometric interpretation of CG is that first a direction is selected in which the error can be reduced the most. In the following iterations the algorithm selects the best direction that is pairwise conjugate to every previous direction.

Algorithm 4.3 Conjugate gradient method

Input: A: $K \times K$ symmetric positive definite matrix; b: output vector; $x^{(0)}$: initial solution; M: preconditioning matrix (e.g.: diag(A)); N_I: number of iterations
Output: x: approximate solution of $Ax = b$
procedure SOLVECG(A, b, x_0, M, N_I)

$r^{(0)} \leftarrow b - Ax^{(0)}$
$z^{(0)} \leftarrow M^{-1} r^{(0)}$
$p^{(0)} \leftarrow z^{(0)}$
for $i = 0, \ldots, N_I - 1$ **do**
  $\alpha^{(i)} \leftarrow \frac{(r^{(i)})^T z^{(i)}}{(p^{(i)})^T A p^{(i)}}$
  $x^{(i+1)} \leftarrow x^{(i)} + \alpha^{(i)} p^{(i)}$
  $r^{(i+1)} \leftarrow r^{(i)} - \alpha^{(i)} A p^{(i)}$
  $z^{(i+1)} \leftarrow M^{-1} r^{(i+1)}$
  $\beta^{(i)} \leftarrow \frac{(z^{(i+1)})^T r^{(i+1)}}{(z^{(i)})^T r^{(i)}}$
  $p^{(i+1)} \leftarrow z^{(i+1)} + \beta_i p_i$
end for
return $x^{(N_I)}$
end procedure

iTALS-CG approximates the feature vectors by replacing $M_j^{(i)} = (C^{(i,j)} +$

[6] The complexity of algorithm 4.1 is $O(N_E N_I K)$ that is $O\left((K^2 + N_j^+ K) N_I\right)$ in our case for one feature vector.

$\lambda I)^{-1}O^{(i,j)}$ in line 18 of algorithm 3.1 with SOLVECG(A, b, x_0, M) with $A = C^{(i,j)} + \lambda I$, $b = O^{(i,j)}$, $x_0 = 0$ and $M = \text{diag}\,(C^{(i,j)} + \lambda I)$. The pseudocode of the conjugate gradient method is presented in Algorithm 4.3. The conjugate gradient method converges to the exact solution in at most K steps. If $N_I = K$ it provides the exact solution, however it is often sufficient to run fewer inner iterations for a good solution. The bottleneck of the CG method is the matrix-vector multiplication with A and the inversion of M in each iteration (see Algorithm 4.3).

Here $A = C^{(i,j)} + \lambda I = C^{(i)} + \sum_{N_j^+}^{k=1} W_t v_k v_k^T + \lambda I$ and we use the Jacobi preconditioner ($M = \text{diag}(A) = \text{diag}\,(C^{(i,j)} + \lambda I)$). With careful implementation the matrix-vector multiplication takes $O(K^2 + N_j^+ K)$ time and the inversion of the diagonal M matrix takes $O(K)$ time. Therefore it takes $O(N_I N_j^+ K + N_I K^2)$ time to compute a feature vector. This sums up to $O(N_I N^+ K + S_i N_I K^2)$ for recomputing one matrix instead of $O(S_i K^3)$, the complexity of the exact method. Therefore the total complexity of iTALS-CG is $O(DN^+ N_I K + N_I K^2 \sum_{i=1}^{D} S_i)$. Note that for iTALS-CG $\mathcal{I}$ is not needed only $\mathcal{J}$ (line 15 can be omitted from algorithm 3.1 when using the CG solver). Therefore the term DN^+K^2 can be omitted from the computation time as well. If $N_I \ll K$ iTALS-CG scales quadratically in the number of features (instead of cubically) in theory. In practice ($DN^+ \ll \sum_{i=1}^{D} S_i$) it scales *linearly* (instead of quadratically) with the number of features for small K values. However, if $N_I \approx K$ then its complexity is the same as of the exact iTALS. Since there are differences in the constant multipliers, iTALS-CG in fact can be slower than the exact iTALS in this case.

5. Experimental setup

We used five genuine implicit feedback data sets to evaluate our algorithm. Three of them are public (LastFM 1K, (Celma 2010); TV1, TV2, (Cremonesi & Turrin 2009)), the other two are proprietary (Grocery, VoD[7]). The properties of the data sets are summarized in Table 1. The column "Multi" shows the average multiplicity of user–item pairs in the training events.[8] The train–test splits are time-based: the first event in the test set falls chronologically after the last event of the training set. The length of the test period was selected to be at least one day, and depends on the domain and the frequency of events. We used the artists as items in LastFM.

Our primary evaluation metric is recall@20. The reason for using recall@N is twofold: (1) we found that in live recommender systems recall usually correlates well with click-through rate (CTR), that is, an important online metric for recommendation success. (2) As described in (Hidasi & Tikk 2012), recall@20 is a good proxy of estimating recommendation accuracy offline for real-world applications; similar finding is available in (Liu et al. 2012). Recall is defined as the ratio of relevant recommended items to relevant items. An item is considered

[7] Data was collected by the service provider of an online grocery store and a vod store respectively, by monitoring the purchases in the system. There were no recommender systems active during the data collection period.

[8] This value is 1.0 at TV1 and TV2. This is possibly due to preprocessing by the original authors that removed duplicate events.

Table 1. Main properties of the data sets

Dataset	Domain	Training set				Test set	
		#Users	#Items	#Events	Multi	#Events	Length
Grocery	E-grocery	24947	16883	6238269	3.0279	56449	1 month
TV1	IPTV	70771	773	544947	1.0000	12296	1 week
TV2	IPTV	449684	3398	2528215	1.0000	21866	1 day
LastFM	Music	992	174091	18908597	21.2715	17941	1 day
VoD	IPTV/VoD	480016	46745	22515406	1.2135	1084297	1 day

relevant for a user (and context-state) if there is an event in the test data with the given user and item (and context-state). An item is recommended if it is ranked in the top $N = 20$ based on its predicted preference value (i.e. $\hat{r}$). Recall does not take into account the position of an item on the recommendation list. We estimate that users are exposed to 20 recommendations in average during a visit (e.g. 4 pageviews, 5 items per recommendation), therefore we choose cutoff at 20. In a practice recommended items are usually randomly selected from the first N elements of the ranked item list, a $N = 20$ is a typical value. In our setting, a recommendation is therefore successful if it is among the top N items. Therefore recall@N suits the offline evaluation of recommender algorithms from the practical viewpoint.

The hyperparameters of the base algorithm, such as regularization coefficients were optimized on a part of the training data (validation set). Then the algorithm was trained on the whole training data (including the validation set) and recall was measured on the test set. The approximate versions used the same regularization coefficient as the corresponding base method. It is possible that there exists a better configuration of hyperparameters for the approximate versions, but keeping the parameters the same enables fair comparison, and we found that these configurations fit also quite well for the approximate learning methods. The number of epochs was set to 10 in all cases, because all methods converge in at most 10 epochs.

5.1. Context dimensions

We used the 3 dimensional ($D = 3$) version of the context-aware algorithms. Two kinds of derived context dimensions were used (separately). These context dimensions are seasonality and sequentiality. Each algorithm is used once with seasonality and once with sequentiality on every dataset. Both contexts can be derived for any dataset that has timestamp associated with its events. Additionally, the two contexts are rather different and capture different aspects of the data. Their applicability to most datasets makes them ideal subject for our experiments.

5.1.1. Seasonality

Many application areas of recommender systems exhibit the seasonality effect, because periodicity can be observed in many human activities. Therefore seasonal data is an obvious choice for context (Liu et al. 2010). First we have to define the length of the season. 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 will be similar. The length of the season depends on the data. Once we have this, within seasons we need to create *time bands* (bins) that 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. In the final step, events are assigned to time bands according to their time stamp.

For *Grocery* we defined a week as the season and the days of the week as the time bands. The argument here is that people usually do shopping on weekly or biweekly basis and that shopping habits differ on weekends and weekdays. One day was used as season for the other four data sets with 4 hour intervals. We note that one can optimize the lengths and distribution of time bands but this is beyond the scope of the current paper.

5.1.2. Sequentiality

In some domains, like movies or music, users consume similar items. In other domains, like electronic gadgets or e-commerce in general, they avoid items similar to what they already consumed and look for complementary products. Sequential patterns can be observed on both domain types. Sequentiality was introduced in (Hidasi & Tikk 2012) and uses the previously consumed item by the user as a context for the actual item. This information helps in the characterizations of repetitiveness related usage patterns and sequential consumption behavior.

During evaluation we fix the sequential context to the item that was targeted by the last transaction of the user in the training set. Thus we do not use information from the test data during the evaluation. The other way (i.e. constantly update the context value based on test events) would be valid as well and would result in better results. Because the test data spans over a short period of time that generally contains a few purchasing sessions for the users, preferences thus can be accurately predicted also from this information.

6. Comparison of learning methods

In this section we compare ALS, ALS-CG and ALS-CD w.r.t. recommendation accuracy, training times and convergence. We also determine the number of inner iterations based on trade-offs between running time and accuracy. We use three algorithms — iTALS, iTALSx and iALS (Hu et al. 2008) — with all three learning methods. As we mentioned before, the generalization of CG and CD learning for complex D dimensional models makes it possible to use these speed-up techniques for every ALS-based factorization.[9]

6.1. Recommendation accuracy

Table 2 shows recommendation accuracy in terms of recall@20 the for three algorithms (iTALS, iTALSx and iALS) with seasonality and sequentiality using various number of features. The number of inner iterations was set to 2 for both CG and CD.

[9] The actual speed-up and improvement in scalability depend on the efficiency of certain key steps (e.g. matrix-vector multiplication for CG). These may differ from algorithm to algorithm.

Dataset	#Features	Seasonality			Sequentiality		
		ALS	ALS-CG	ALS-CD	ALS	ALS-CG	ALS-CD
Grocery	**40**	0.1071	0.1065	0.1043	0.1339	0.1304	0.1317
	80	0.1146	0.1193	N/A	0.1439	0.1381	0.1426
	200	0.1312	0.1342	N/A	0.1570	0.1485	0.1540
TV1	**40**	0.1235	0.1194	N/A	0.1515	0.1521	0.1518
	80	0.1167	0.1147	N/A	0.1553	0.1511	0.1483
	200	0.1055	0.1063	N/A	0.1517	0.1520	0.1505
TV2	**40**	0.2001	0.2004	0.1972	0.3103	0.3066	0.3094
	80	0.2123	0.2102	N/A	0.2957	0.2974	0.2961
	200	0.2184	0.2111	N/A	0.2821	0.2848	0.2847
LastFM	**40**	0.0888	0.1040	0.0909	0.1657	0.1605	0.1579
	80	0.1290	0.1417	N/A	0.1864	0.1796	0.1780
	200	0.1382	0.1970	N/A	0.1784	0.2044	0.2045
VoD	**40**	0.0909	0.0913	0.0910	0.1380	0.1372	0.1347
	80	0.0996	0.1002	0.0990	0.1723	0.1720	0.1627
	200	0.1026	0.1036	0.1023	0.2116	0.2111	0.2092

(a) Results with iTALS

Dataset	#Features	Seasonality			Sequentiality		
		ALS	ALS-CG	ALS-CD	ALS	ALS-CG	ALS-CD
Grocery	**40**	0.1164	0.1208	0.1135	0.1299	0.1272	0.1283
	80	0.1406	0.1445	0.1340	0.1431	0.1385	0.1411
	200	0.1927	0.1915	0.1842	0.1655	0.1531	0.1610
TV1	**40**	0.1127	0.1077	0.1043	0.1417	0.1410	0.1414
	80	0.0942	0.0858	0.0905	0.1295	0.1309	0.1295
	200	0.0696	0.0650	0.0688	0.1106	0.1098	0.1104
TV2	**40**	0.2312	0.2274	0.2195	0.2866	0.2846	0.2856
	80	0.2223	0.2130	0.2117	0.3006	0.3017	0.2986
	200	0.1791	0.1741	0.1807	0.3023	0.3067	0.3079
LastFM	**40**	0.0599	0.0691	0.0507	0.1869	0.1830	0.1859
	80	0.0928	0.0773	0.0708	0.1984	0.1966	0.1929
	200	0.1264	0.0907	0.0922	0.2003	0.2007	0.2006
VoD	**40**	0.0916	0.0931	0.0927	0.1068	0.1073	0.1068
	80	0.0990	0.0999	0.0986	0.1342	0.1345	0.1347
	200	0.0977	0.0980	0.0970	0.1726	0.1732	0.1728

(b) Results with iTALSx

Dataset	#Features	ALS	ALS-CG	ALS-CD
Grocery	**40**	0.0714	0.0745	0.0814
	80	0.0861	0.0919	0.0966
	200	0.1281	0.1298	0.1237
TV1	**40**	0.1111	0.1072	0.1074
	80	0.0926	0.0899	0.0937
	200	0.0769	0.0712	0.0799
TV2	**40**	0.2161	0.2043	0.2162
	80	0.2145	0.1906	0.2140
	200	0.1958	0.1702	0.1894
LastFM	**40**	0.0623	0.0545	0.0467
	80	0.0922	0.0902	0.0574
	200	0.1199	0.1204	0.0453
VoD	**40**	0.0758	0.0779	0.0758
	80	0.0884	0.0889	0.0878
	200	0.0928	0.0921	0.0918

(c) Results with iALS

Table 2. Recommendation accuracy with various learning methods.

Method	Performs similarly	Underperforms	Outperforms	Fails	*Total*
CG	58 (77.33%)	11 (14.67%)	6 (8%)	0 (0%)	*75 (100%)*
CD	54 (72%)	9 (12%)	3 (4%)	9 (12%)	*75 (100%)*

Table 3. Overview of the relation of approximate methods to ALS. Similar performance means that the difference in recall@20 is lower than 5%.

Method	Insignificant difference	Worse than ALS	Better than ALS	Fails	*Total*
CG	29 (38.67%)	28 (37.33%)	18 (24%)	0 (0%)	*75 (100%)*
CD	32 (42.67%)	26 (34.67%)	8 (10.67%)	9 (12%)	*75 (100%)*

Table 4. Summary of statistical significance tests comparing CG and CD to ALS. ($p = 0.05$)

Some values are missing from the table, because the training with CD sometimes failed. One can observe that CD is somewhat unstable if there are n-way ($n > 2$) interactions in the preference model, the size of any of the interacting dimensions is low and the number of features is high. Additional experiments with different preference models confirmed this disadvantage.

The recommendation accuracy of ALS and the approximate methods are usually very similar. There are some exceptions with moderate differences. Although the value of the loss function (wRMSE) is correlated with the evaluation metric (recall), there is no direct connection between them. Thus the approximate methods can outperform the exact ALS. There are only a few examples where the difference in the accuracy is considerable, but there is no clear trend on the characteristics of these examples.

For overview on the relation of the approximate methods to ALS see Table 3. Since small differences in recall usually do not increase practical accuracy, a threshold of 5% was set. We consider a method considerably better than an other if its recall is by at least 5% larger. CG performs slightly better than CD w.r.t. recommendation accuracy similarity to the exact method (58 and 54). CG also has more (considerably) outperforming results compared to the exact ALS than CD (6 and 3). The number of underperforming cases is roughly the same for CG and CD (11 and 9). If there is any appreciable difference vs. ALS, CD usually exhibits lower performance (9 of 12), while CG outperforms ALS in more than one third of the cases (6 of 17).

Besides examining if differences between methods are considerable, we also checked if the differences are statistically significant. The test set was split into 10 parts randomly. The number of relevant and recommended items (i.e. the nominator of recall) was measured for all parts.[10] Then paired t-test was used to compare the methods with $p = 0.05$. Table 4 contains the aggregated results. The trends are similar as in Table 3: CD underperforms ALS in slightly more cases than CG does; and CG has more over performances than CD.

We also combined both comparison methodologies; Table 5 depicts the results. The table is very similar to Table 3, there are only a few considerably different cases that are not significantly different.

We can thus summarize that from the recommendation accuracy point of

[10] With fixed list length and test set these values are proportional to the recall@20 value.

Method	Performs similarly	Underperforms	Outperforms	Fails	*Total*
CG	62 (82.67%)	10 (13.33%)	3 (4%)	0 (0%)	*75 (100%)*
CD	57 (76%)	7 (9.33%)	2 (2.67%)	9 (12%)	*75 (100%)*

Table 5. Comparing CG and CD to ALS when differences should be both considerably and statistically significantly better.

view, both approximate learning methods are on par with the original ALS. However, due to its stability and somewhat better accuracy, CG is the more appropriate choice than CD in general. Let us investigate how much one can gain on the training time with the approximate methods, since this can justify the use of an approximate method.

6.2. Running time

Figure 3 depicts the time required to run one epoch (i.e. computing each feature matrix once) of iTALS with ALS, CG and CD with different number of features using the VoD dataset. The number of inner iterations was set to 2 for CG and CD. The results provide underpinning for our earlier statements about the practical scaling of the methods in the number of features. It is clear that both approximate methods scale better than the exact ALS. The speed-up factor is ~ 10.6 for CG and ~ 2.9 for CD if $K = 200$ (and it becomes even greater for larger K values). For the more commonly used $K = 80$, the speed-up is ~ 3.5 and ~ 1.3 for CG and CD, respectively. As expected, with few features (e.g. $K = 20$), due to the computational overhead the approximate methods can be somewhat slower than the exact ALS. Summarizing: for low factor models, ALS can be an appropriate learner, while for higher factors, ALS-CG and ALS-CD offer considerable speed-up.

For high factor models, CG is significantly faster than CD. CD starts scaling super-linear much earlier (around $K = 100$ in this example) than CG (still linear for $K = 200$) and also starts off with a steeper scaling graph. The speed-up from CD to CG for $K = 80$ and $K = 200$ is ~ 2.6 and ~ 3.8, respectively.

6.3. Number of inner iterations

The number of inner iterations is an important parameter of the approximate methods. Generally, the larger this value is, the more accurate the algorithms are at the cost of the increased training times. In this section we determine a good choice of this value by analyzing the trade-off between training time and accuracy.

Figure 4 compares the accuracy of CG and CD to ALS with different numbers of inner iterations. We selected two examples: one where CG and CD approximates ALS well in Table 2 (iTALSx, LastFM, sequentiality, $K = 80$); and one where they don't (iTALSx, LastFM, seasonality, $K = 80$). With other experiments, we observed very similar results (not shown here). We also note that the former case is more common than the latter.

In the first example, both approximate methods start from a lower value at $N_I = 1$. From there, their accuracy is gradually increased and CG improves

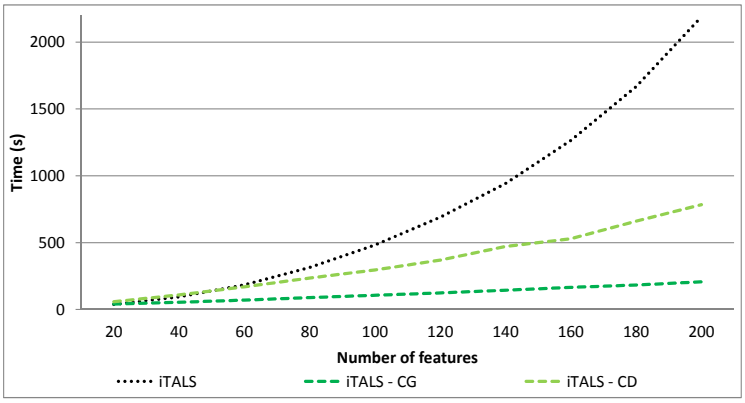

Fig. 3. Running times of one epoch of different learning methods (ALS, CD, CG) with iTALS w.r.t. different number of feature (K) values, using one CPU core

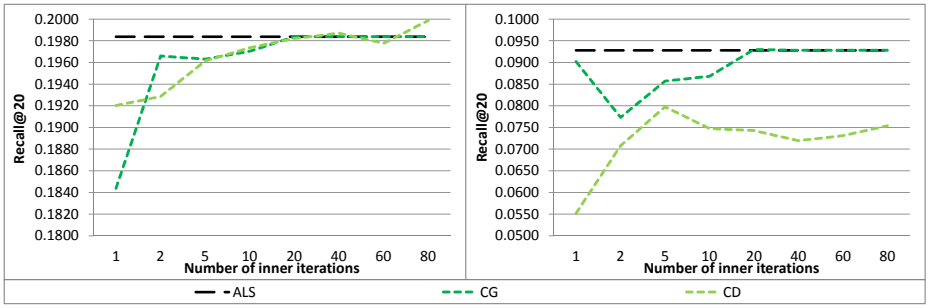

Fig. 4. Recommendation accuracy with different number of inner iterations for CG and CD learning; value for LS solver is shown for comparison

slightly faster. CG and ALS compute exactly the same features if $N_I = K$. Thus CG converges to ALS as the number of inner iterations increases. They also yield the same accuracy if K is sufficiently high (20 in this case). On the other hand, CD does not converge to ALS, but gives quite similar accuracy values. It reaches the accuracy of ALS around the same $N_I = 20$ as well, and one can observe very slight variance of accuracy for $N_I > 20$. At $N_I = 80$ the accuracy becomes even slightly better than that of ALS. This is not a general behavior of CD, but as it does not converge to the exact ALS, sometimes it can give slightly better results. Note that CG can also outperform ALS (as shown in Table 2), but only by low N_I values. Even is CG starts off with a higher accuracy than ALS, by the increasing N_I it converges to ALS. CD can, however, theoretically outperform ALS at any N_I values.

In the second example, there is a larger difference between accuracy values of the approximate and the exact learning. CG has a relatively high accuracy at $N_I = 1$, but this is not a general behavior by any means. From $N_I = 2$, the accuracy of CG starts increasing monotonously and reaches that of the ALS around $N_I = 20$. On the other hand, the accuracy of CD varies throughout and it never even approaches that of the ALS.

Our experiments show that the fact that CG converges to ALS can equally

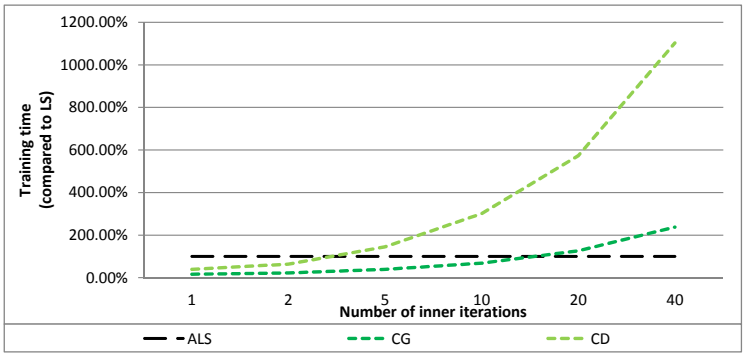

Fig. 5. Running times with different number of inner iterations compared to that of the LS solver

be favorable or can be a limit. On one hand it attests to the stability of the method, on the other hand, in some cases CD can outperform both ALS and CG. In practice, however, such a case is not typical (see Table 2), thus we can conclude that the convergence of CG is useful.

Figure 5 compares the training times of CG and CD to ALS by different N_I values. The experiment used iTALSx, LastFM, sequentiality and $K = 80$. We note that the results are very similar with other settings.[11] CG scales significantly better with N_I than CD. CD reaches the training time of ALS around $N_I = 3 \dots 4$, that is only $\sim 4 - 5\%$ of K. CG reaches the training time of ALS much later, around $N_I = 15$, that is $\sim 19\%$ of K.

Approximate methods are used to speed up the training. Therefore such N_I should be used when the training time is significantly less. Our experiments suggest that this value is $1 \dots 2$ for CD and $1 \dots 10$ for CG if $K = 80$. For different K values these intervals change relative to K. Generally, $N_I = 1$ is bad a choice due to low accuracy (see Figure 4 for example), therefore N_I should be at least 2. Table 2 shows that $N_I = 2$ is usually a good choice as the accuracy of ALS is usually well approximated. Larger N_I values are not advised for CD due to its poor scaling with N_I. For CG, N_I values up to 10–15% of K still result in considerable speed up, but usually small values $(2 \dots 5)$ are sufficient.

6.4. Convergence of accuracy

Figure 6 compares the accuracy of ALS, CG and CD (with 2 and 5 inner iterations each) after each recomputation of any feature matrix (i.e. their convergence w.r.t. accuracy). We investigate two cases: (1) when ALS converges faster (iTALSx, LastFM, sequentiality, $K = 80$), (2) when ALS converges more slowly (iTALS, VoD, sequentiality, $K = 40$). If ALS converges slowly then approximate methods can keep up and converge with basically the same speed. If the convergence of ALS is faster, approximate methods with $N_I = 2$ are initially less accurate, but achieve the same results after a few epochs. When the number of

[11] In the following sense: N_I values relative to the number of features. That is, if K is lower/higher then approximate methods reach the training time of ALS at lower/higher N_I values.

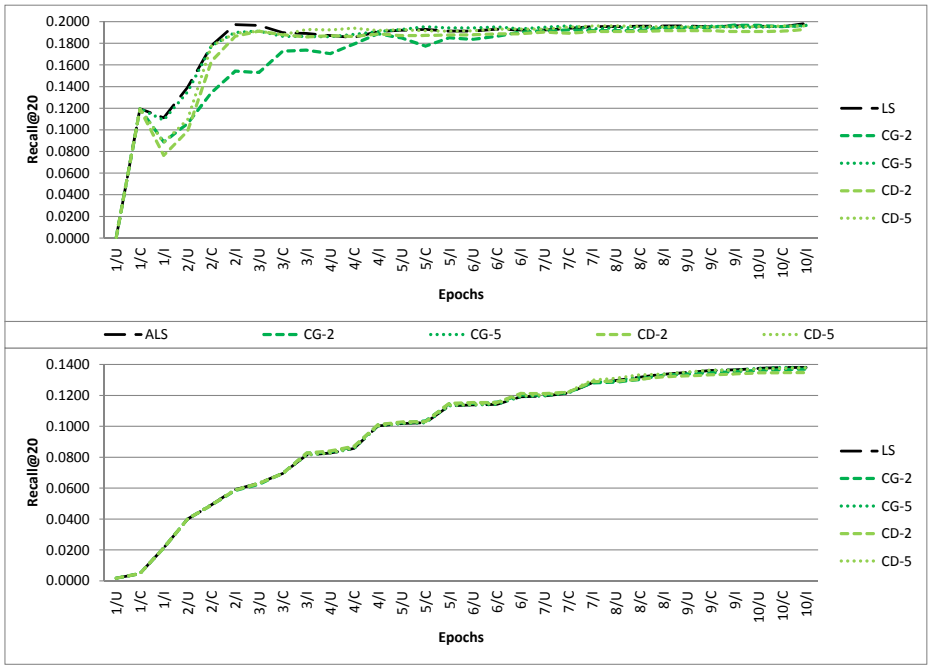

Fig. 6. Accuracy of ALS, CG and CD methods (2 and 5 inner iterations) through epochs. The horizontal axis denotes how far the algorithm is in the computations. N/X denotes that the algorithm is in the N^{th} epoch and finished computing the X matrix ($X \in (U, I, C)$ as in User, Item and Context feature matrix).

inner iterations is set higher ($N_I = 5$), approximate methods follow ALS quite nicely.

This suggests that $N_I = 5$ would be a better choice for CG as it follows ALS more closely. We suggest to use $N_I = 2$ if speed is important, because one mostly gets similar or better results than with ALS. If one prefers accuracy against training time then $N_I = 5$ (or higher) can be used. For CD we still suggest using $N_I = 2$ in every case, because of the fast increase of training time with larger N_I values.

6.5. Size of training data

In real life recommenders it is important to consider how much of the users' event history is to be used. Too much data not only increases training time, but it may mask changes in taste and behavior. On the other hand, using only the recent events results in noisy training data and does not allow for models that capture long time interests. The optimal trade-off depends on the domain, the dataset itself and even the contexts considered. Finding this optimal trade-off is beyond the scope of this paper. Here we therefore only investigate if the approximate methods behave similarly to ALS.

Figure 7 shows the accuracy (w.r.t. recall@20) with using different slices of the training data in three different settings. Although the graph varies from

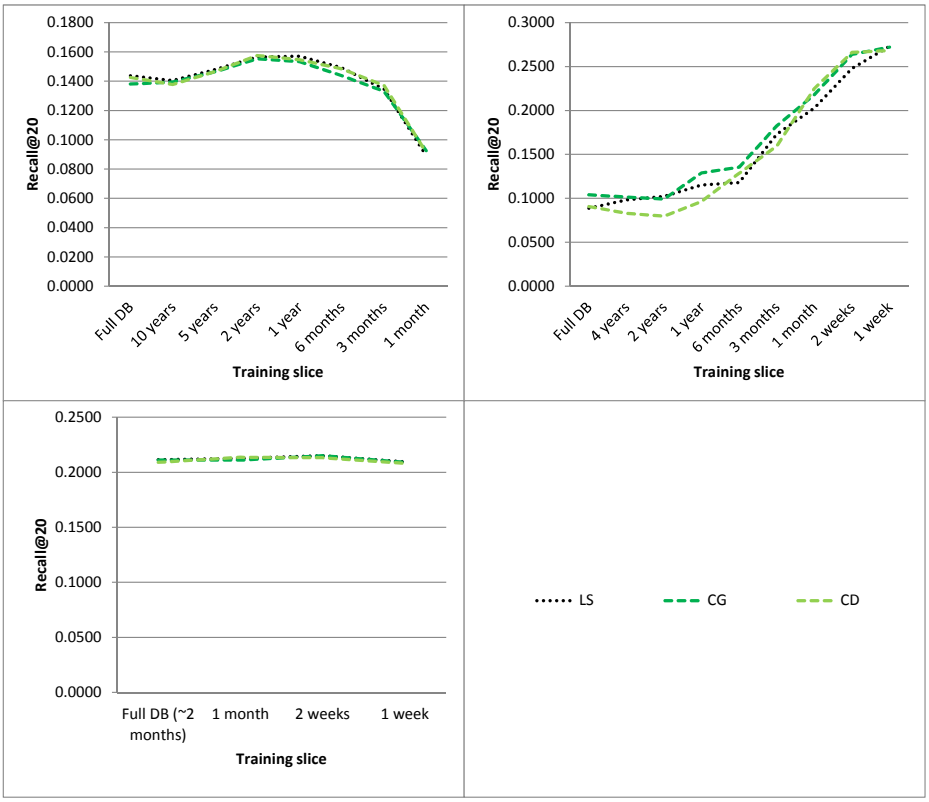

Fig. 7. Accuracy of ALS, CG and CD methods (2 inner iterations) using different subsets of the training data. The end of the training interval remains the same, only the starting date changes. The three settings depicted here are Grocery, iTALS, sequentiality, 80 features (top left); LastFM, iTALS, seasonality, 40 features (top right); VoD, iTALS, sequentiality, 200 features (bottom left).

setting to setting, both CG and CD follow ALS closely in all settings. Therefore we conclude that CG and CD behave similarly to ALS in this aspect as well.

7. Conclusion

The practically important recommendation problems with implicit data are often solved by ALS-based factorization approaches. We investigated the scalability of such approaches in the context-aware setting, since in practice, scalability and training time are of key importance. With efficient implementation, ALS-learning scales linearly with the number of *non-zeroes* in the tensor, but cubically with K, the number of features (for smaller K, it is quadratic in practice). This prevents the use of more accurate high factor models. Since for ALS-learning map-reduce technology is not an option for faster training, an alternative solution was proposed here to offer training speed-up and to overcome the scalability problem.

We presented two approximate learning schemes: coordinate descent (CD) and conjugate gradient (CG) and applied to the complex model of iTALS. We generalized said approaches to be compliant with every ALS based factorization method. Both CD and CG scale *linearly* with the number of features in practice for lower K values (while CG scales quadratically and CD scales cubically in theory).

We analyzed the performance of CG, CD, and ALS in terms or recommendation accuracy, training times, accuracy–training time trade-off and convergence. In most cases, CG and CD approximate the accuracy of ALS well or even outperform it. In other cases, increasing the number of inner iterations can improve their accuracy at the cost of lower speed. Both approximate methods are much faster and scale better than ALS, with CG being the faster of the two (more than 10 times faster than ALS by $K = 200$). We specified how the preferred number of inner iterations for CG and CD, offering a good trade-off between accuracy and speed. We found that CG is generally better than CD. It is more stable, generally more accurate, converges to ALS as the number of inner iterations increases, it is faster and scales better. On the other hand, CD can also be advantageous in some cases, because it is not a direct approximation of ALS. Therefore, CD can still provide good accuracy even if ALS does not work.

We can conclude that the proposed approximate solutions make it possible to apply ALS-based learning of high factor models efficiently for more complex context-aware methods. The solutions also offer trade-off between recommendation accuracy and speed training time.

Acknowledgements

The work leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under CrowdRec Grant Agreement n° 610594. The authors would like to thank Martha Larson for her useful comments on the paper.

References

Adomavicius, G. & Ricci, F. (2009), Workshop on context-aware recommender systems (CARS-2009), *in* 'Recsys'09: ACM Conf. on Recommender Systems', pp. 423–424.

Adomavicius, G., Sankaranarayanan, R., Sen, S. & Tuzhilin, A. (2005), 'Incorporating contextual information in recommender systems using a multidimensional approach', *ACM Trans. Inf. Syst.* **23**(1), 103–145.

Adomavicius, G. & Tuzhilin, A. (2008), Context-aware recommender systems, *in* 'Recsys'08: ACM Conf. on Recommender Systems', pp. 335–336.

Bader, R., Neufeld, E., Woerndl, W. & Prinz, V. (2011), Context-aware POI recommendations in an automotive scenario using multi-criteria decision making methods, *in* 'CaRR'11: Workshop on Context-awareness in Retrieval and Recommendation', pp. 23–30.

Balassi, M., Pálovics, R. & Benczúr, A. A. (2014), Distributed frameworks for alternating least squares, *in* '2nd Large Scale Recommender Systems Workshop at Recsys 2014', Foster City, CA, USA.

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

Cremonesi, P. & Turrin, R. (2009), Analysis of cold-start recommendations in IPTV systems, *in* 'Recsys'09: ACM Conf. on Recommender Systems'.

Dias, R. & Fonseca, M. J. (2013), Improving music recommendation in session-based collaborative filtering by using temporal context, *in* 'Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on', IEEE, pp. 783–788.

Hestenes, M. R. & Stiefel, E. (1952), 'Methods of conjugate gradients for solving linear systems', *Journal of Research of the National Bureau of Standards* pp. 409–436.

Hidasi, B. (2014), 'Factorization models for context-aware recommendations', *Infocommunications Journal* **VI**(4), 27–34.

Hidasi, B. & Tikk, D. (2012), Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback, *in* 'ECML-PKDD'12, Part II', number 7524 *in* 'LNCS', Springer, pp. 67–82.

Hu, Y., Koren, Y. & Volinsky, C. (2008), Collaborative filtering for implicit feedback datasets, *in* 'ICDM'08: IEEE Int. Conf. on Data Mining', pp. 263–272.

Jahrer, M. & Töscher, A. (2011), Collaborative filtering ensemble for ranking, *in* 'KDD Cup Workshop at 17^{th} ACM SIGKDD'11'.

Karatzoglou, A., Amatriain, X., Baltrunas, L. & Oliver, N. (2010), Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering, *in* 'Recsys'10: ACM Conf. on Recommender Systems', pp. 79–86.

Koren, Y. & Bell, R. (2011), Advances in collaborative filtering, *in* F. Ricci et al., eds, 'Recommender Systems Handbook', Springer, pp. 145–186.

Lathauwer, L., Moor, B. & Vandewalle, J. (2000), 'A multilinear singular value decomposition', *SIAM J. Matrix Anal. Appl.* **21**(4), 1253–1278.

Little, R. J. A. & Rubin, D. B. (1987), *Statistical Analysis with Missing Data*, John Willey & Sons, Inc.

Liu, N. N., Zhao, B. C. M. & Yang, Q. (2010), Adapting neighborhood and matrix factorization models for context aware recommendation, *in* 'CAMRa'10: Workshop on Context-Aware Movie Recommendation', pp. 7–13.

Liu, Q., Chen, T., Cai, J. & Yu, D. (2012), Enlister: Baidu's recommender system for the biggest Chinese Q&A website, *in* 'RecSys-12: Proc. of the 6th ACM Conf. on Recommender Systems', pp. 285–288.

Lommatzsch, A. (2014), Real-time news recommendation using context-aware ensembles, *in* M. de Rijke, T. Kenter, A. de Vries, C. Zhai, F. de Jong, K. Radinsky & K. Hofmann, eds, 'Advances in Information Retrieval', Vol. 8416 of *Lecture Notes in Computer Science*, Springer International Publishing, pp. 51–62.

Nguyen, T. V., Karatzoglou, A. & Baltrunas, L. (2014), Gaussian process factorization machines for context-aware recommendations, *in* 'SIGIR-14: ACM SIGIR Conference on Research & Development in Information Retrieval', pp. 63–72.

Pan, R., Zhou, Y., Cao, B., Liu, N. N., Lukose, R. M., Scholz, M. & Yang, Q.

(2008), One-class collaborative filtering, *in* 'ICMD'08: 8$^{\text{th}}$ IEEE Int. Conf. on Data Mining', pp. 502–511.

Pilászy, I., Zibriczky, D. & Tikk, D. (2010), Fast ALS-based matrix factorization for explicit and implicit feedback datasets, *in* 'Recsys'10: ACM Conf. on Recommender Systems', pp. 71–78.

Rendle, S. (2012), 'Factorization machines with libFM', *ACM Transactions on Intelligent Systems and Technology (TIST)* **3**(3), 57.

Rendle, S., Freudenthaler, C., Gantner, Z. & Schmidt-Thieme, L. (2009), BPR: Bayesian personalized ranking from implicit feedback, *in* 'UAI'09: 25th Conf. on Uncertainty in Artificial Intelligence', pp. 452–461.

Rendle, S., Gantner, Z., Freudenthaler, C. & Schmidt-Thieme, L. (2011), Fast context-aware recommendations with factorization machines, *in* 'SIGIR'11: ACM Int. Conf. on Research and Development in Information', pp. 635–644.

Rendle, S. & Schmidt-Thieme, L. (2010), Pairwise interaction tensor factorization for personalized tag recommendation, *in* 'WSDM'10: ACM Int. Conf. on Web Search and Data Mining', pp. 81–90.

Ricci, F. et al., eds (2011), *Recommender Systems Handbook*, Springer.

Said, A., Berkovsky, S. & Luca, E. W. D. (2010), Putting things in context: Challenge on context-aware movie recommendation, *in* 'CAMRa'10: Workshop on Context-Aware Movie Recommendation', pp. 2–6.

Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Hanjalic, A. & Oliver, N. (2012), TFMAP: Optimizing MAP for top-N context-aware recommendation, *in* 'SIGIR'12: ACM SIGIR Conf. on Research and Development in Information Retrieval', ACM, New York, NY, USA, pp. 155–164.

Takács, G., Pilászy, I. & Tikk, D. (2011), Applications of the conjugate gradient method for implicit feedback collaborative filtering, *in* 'RecSys'11: ACM Conf. on Recommender Systems', pp. 297–300.

Takács, G. & Tikk, D. (2012), Alternating least squares for personalized ranking, *in* 'Recsys'12: 6th ACM Conf. on Recommender Systems', pp. 83–90.

Zarka, R., Cordier, A., Egyed-Zsigmond, E. & Mille, A. (2012), Contextual trace-based video recommendations, *in* 'Proceedings of the 21st International Conference Companion on World Wide Web', WWW '12 Companion, ACM, New York, NY, USA, pp. 751–754.
URL: *http://doi.acm.org/10.1145/2187980.2188196*

Balázs Hidasi is the Head of Data Mining and Research at Gravity R&D, a recommender system vendor company. He joined the company in 2010. His research interests cover a broad spectrum of machine learning and data mining algorithms and problems. In the last five years, his research revolved around recommender algorithms for real life recommendation problems with the main focus on context-aware factorization algorithms on implicit feedback data. He is the first author of several papers in this topic. He obtained his MSc in 2011 in computer science and engineering from the Budapest University of Technology and Economics with highest honors. He is currently finishing his PhD.

Domonkos Tikk is CEO and co-founder of Gravity R&D, a recommender system vendor company. He obtained his PhD in 2000 in computer science from Technical University of Budapest. He worked on machine learning and data and text mining topics in the last decade. He received Humboldt-scholarship in 2008. His team, Gravity, participated at the Netflix Prize challenge, and finished tied at the first position of the challenge, and funded Gravity R&D to exploit this achievement. Domonkos authored about 200 papers, about 30 of them on recommender systems. His paper was awarded at Recsys 2012. He also acted as the co-chair of the recommender system-related KDD-Cup 2007, RecsysChallenge 2012 and 2014. He is industry chair of Recsys 2015. In 2012 he gave tutorial on best practices in Recommender System Challenges at ACM RecSys.