Fast Matrix Factorization for Online Recommendation with Implicit Feedback*

Xiangnan He Hanwang Zhang Min-Yen Kan Tat-Seng Chua
School of Computing, National University of Singapore
{xiangnan, hanwang, kanmy, chuats}@comp.nus.edu.sg

ABSTRACT

This paper contributes improvements on both the effectiveness and efficiency of *Matrix Factorization* (MF) methods for implicit feedback. We highlight two critical issues of existing works. First, due to the large space of unobserved feedback, most existing works resort to assign a uniform weight to the missing data to reduce computational complexity. However, such a uniform assumption is invalid in real-world settings. Second, most methods are also designed in an offline setting and fail to keep up with the dynamic nature of online data.

We address the above two issues in learning MF models from implicit feedback. We first propose to weight the missing data based on item popularity, which is more effective and flexible than the uniform-weight assumption. However, such a non-uniform weighting poses efficiency challenge in learning the model. To address this, we specifically design a new learning algorithm based on the **e**lement-wise **A**lternating **L**east **S**quares (eALS) technique, for efficiently optimizing a MF model with variably-weighted missing data. We exploit this efficiency to then seamlessly devise an incremental update strategy that instantly refreshes a MF model given new feedback. Through comprehensive experiments on two public datasets in both offline and online protocols, we show that our eALS method consistently outperforms state-of-the-art implicit MF methods. Our implementation is available at https://github.com/hexiangnan/sigir16-eals.

Keywords

Matrix Factorization, Implicit Feedback, Item Recommendation, Online Learning, ALS, Coordinate Descent

1. INTRODUCTION

User personalization has become prevalent in modern recommender system. It helps to capture users' individualized preferences and has been shown to increase both satisfaction for users and revenue for content providers. Among its various methods, matrix factorization (MF) is the most popular and effective technique that characterizes users and items by vectors of latent factors [15, 32]. Early work on MF algorithms for recommendation [14, 27] have largely focused on explicit feedback, where users' ratings that directly reflect their preference on items are provided. These works formulated recommendation as a rating prediction problem for which the large volume of unobserved ratings (*i.e.,* missing data) are assumed to be extraneous for modeling user preference [12]. This greatly reduces the modeling workload, and many sophisticated methods have been devised, such as SVD++ [15] and timeSVD [14].

However, explicit ratings are not always available in many applications; more often, users interact with items through **implicit feedback**, *e.g.,* users' video viewing and product purchase history. Compared to explicit ratings, implicit feedback is easier to collect for content providers, but more challenging to utilize due to the natural scarcity of negative feedback. It has been shown that modeling only the observed, positive feedback results in biased representations in user profiles [4, 12]; *e.g.,* Marlin *et al.* [18] finds that users listen to music they expect to like and avoid the genres they dislike, leading to a severe bias in the observed data.

To solve the problem of lacking negative feedback (also known as the one-class problem [21]), a popular solution is to model all the missing data as negative feedback [12]. However, this adversely degrades the learning efficiency due to the full consideration of both observed and missing data. More importantly, the low efficiency makes it even more difficult to deploy implicit MF method **online** [29]. In practical recommender systems where new users, items and interactions are continuously streaming in, it is crucial to refresh the underlying model in real-time to best serve users. In this work, we concern the above two challenging problems of the MF method — implicit feedback and online learning. We note that we are not the first to consider both aspects for MF, as a recent work by Devooght *et al.* [4] has proposed an efficient implicit MF method for learning with dynamic data. However, we argue that Devooght's method [4] models missing data in an unrealistic, suboptimal way. Specifically, it assigns a uniform weight to the missing data, assuming that the missing entries are equally likely to be negative feedback. However, such an assumption limits model's fidelity and flexibility for real applications. For example, content providers usually know which items have been frequently featured to users but seldom clicked; such items are more likely to be

*NExT research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative.

SIGIR '16, July 17-21, 2016, Pisa, Italy

DOI: http://dx.doi.org/10.1145/2911451.2911489

true negative assessments and should be weighted higher than others. In addition, Devooght's method learns parameters through gradient descent, requiring an expensive line search to determine the best learning rate at each step.

We propose a new MF method aimed at learning from implicit feedback effectively while satisfying the requirement of online learning. We develop a new learning algorithm that efficiently optimizes the implicit MF model without imposing a uniform-weight restriction on missing data. In particular, we assign the weight of missing data based on the popularity of items, which is arguably more effective than the previous methods [4, 12, 23, 30, 31] that are limited by the uniformity assumption. Our eALS algorithm is fast in accounting for missing data — analytically K times faster than ALS [12] where K denotes number of latent factors — the same time complexity with the recent dynamic MF solution [4]. This level of efficiency makes eALS suitable for learning online, for which we develop an incremental update strategy that instantly refreshes model parameters given new incoming data. Another key advantage of eALS is that it works without learning rate, bypassing the well-known difficulty for tuning gradient descent methods such as [4] and *Stochastic Gradient Descent* (SGD) [25].

We summarize our key contributions as follows.

1. We propose an item popularity-aware weighting scheme on the full missing data that effectively tailors the MF model for learning from implicit feedback.
2. We develop a new algorithm for learning model parameters efficiently and devise an incremental update strategy to support real-time online learning.
3. We conduct extensive experiments with both offline and online protocols on two real-world datasets, showing that our method consistently outperforms state-of-the-art implicit MF methods.

2. RELATED WORK

Handling missing data is obligatory for learning from implicit data due to the lack of negative feedback. To this end, two strategies have been proposed — **sample based learning** [21, 25] that samples negative instances from missing data, or **whole-data based learning** [12, 30] that treats all missing data as negative. Both methods have pros and cons: sample-based methods are more efficient by reducing negative examples in training, but risk decreasing the model's predictiveness; whole-based methods model the full data with a potentially higher coverage, but inefficiency can be an issue. To retain model's fidelity, we persist in the whole-data based learning, developing a fast ALS-based algorithm to resolve the inefficiency issue.

For existing whole-data based methods [4, 12, 23, 30, 31], one major limitation is in the uniform weighting on missing entries, which favors algorithm's efficiency but limits model's flexibility and extensibility. The only works that have considered non-uniform weighting are from Pan *et al.* [20, 21]; however their cubic time complexity *w.r.t.* K makes it unsuitable to run on large-scale data [23], where a large number of factors needs to be considered to gain improved performance [31].

To optimize MF, various learners have been investigated, including SGD [14, 25], *Coordinate Descent* (CD) [4, 32], and *Markov Chain Monto Carlo* (MCMC) [26]. SGD is the most popular one owing to the ease of derivation, however, it is unsuitable for whole-data based MF [12] due to the large amount of training instances (the full user–item interaction matrix is considered). ALS can be seen as an instantiation of CD and has been widely used to solve the whole-based MF [12, 20, 21, 30]; however, its inefficiency is the main obstacle for practical use [23, 31]. To resolve this, [23] describes an approximate solution to ALS. Recently, [4] employs the *Randomized block Coordinate Descent* (RCD) learner [28], reducing the complexity and applying it to a dynamic scenario. Similarly, [31] enriches the implicit feedback matrix with neighbor-based similarly, followed by applying unweighted SVD. Distinct from previous works, we propose an efficient element-wise ALS solution for the whole-data based MF with non-uniform missing data, which has never been studied before.

Another important aspect for practical recommender system lies in handling the dynamic nature of incoming data, for which timeliness is a key consideration. As it is prohibitive to retrain the full model online, various works have developed incremental learning strategies for neighbor-based [13], graph-based [9], probabilistic [3] and MF [4, 5, 17, 27] methods. For MF, different learners have been studied for online updating, including SGD [5, 27], RCD [4] and dual-averaging [17]. To our knowledge, this work is the first attempt to exploit the ALS technique for online learning.

3. PRELIMINARIES

We first introduce the whole-data based MF method for learning from implicit data, highlighting the inefficiency issue of the conventional ALS solution [12, 21]. Then we describe eALS, an element-wise ALS learner [26] that can reduce the time complexity to linearity *w.r.t.* number of factors. Although the learner is generic in optimizing MF with all kinds of weighting strategies, the form introduced is costly in accounting for all missing data and is thus unrealistic for practical use. This defect motivates us to further develop eALS to make it suitable for learning from implicit feedback (details in Section 4.2).

3.1 MF Method for Implicit Feedback

We start by introducing some basic notation. For a user–item interaction matrix $\mathbf{R} \in \mathbb{R}^{M \times N}$, M and N denote the number of users and items, respectively; $\mathcal{R}$ denotes the set of user–item pairs whose values are non-zero. We reserve the index u to denote a user and i to denote an item. Vector $\mathbf{p}_u$ denotes the latent feature vector for u, and set $\mathcal{R}_u$ denotes the set of items that are interacted by u; similar notations for $\mathbf{q}_i$ and $\mathcal{R}_i$. Matrices $\mathbf{P} \in \mathbb{R}^{M \times K}$ and $\mathbf{Q} \in \mathbb{R}^{N \times K}$ denote the latent factor matrix for users and items.

Matrix factorization maps both users and items into a joint latent feature space of K dimension such that interactions are modeled as inner products in that space. Mathematically, each entry r_{ui} of $\mathbf{R}$ is estimated as:

$$\hat{r}_{ui} =< \mathbf{p}_u, \mathbf{q}_i >= \mathbf{p}_u^T \mathbf{q}_i. \quad (1)$$

The item recommendation problem is formulated as estimating the scoring function $\hat{r}_{ui}$, which is used to rank items. Note that this basic model subsumes the biased MF [14], commonly used in modeling explicit ratings:

$$\hat{r}_{ui} = b_u + b_i + < \mathbf{p}_u^B, \mathbf{q}_i^B >,$$

where b_u (b_i) captures the bias of user u (item i) in giving (receiving) ratings. To recover it, set $\mathbf{p}_u \leftarrow [\mathbf{p}_u^B, b_u, 1]$ and

$\mathbf{q}_i \leftarrow [\mathbf{q}_i^B, 1, b_i]$. As such, we adopt the basic MF model to make notations simple and also to enable a fair comparison with baselines [4, 12] that also complied with the basic model.

To learn model parameters, Hu *et al.* [12] introduced a weighted regression function, which associates a confidence to each prediction in the implicit feedback matrix $\mathbf{R}$:

$$J = \sum_{u=1}^{M}\sum_{i=1}^{N} w_{ui}(r_{ui} - \hat{r}_{ui})^2 + \lambda(\sum_{u=1}^{M}||\mathbf{p}_u||^2 + \sum_{i=1}^{N}||\mathbf{q}_i||^2), \quad (2)$$

where w_{ui} denotes the weight of entry r_{ui} and we use $\mathbf{W} = [w_{ui}]_{M\times N}$ to represent the weight matrix. λ controls the strength of regularization, which is usually an L_2 norm to prevent overfitting. Note that in implicit feedback learning, missing entries are usually assigned to a zero r_{ui} value but non-zero w_{ui} weight, both crucial to performance.

3.2 Optimization by ALS

Alternating Least Square (ALS) is a popular approach to optimize regression models such as MF and graph regularization [10]. It works by iteratively optimizing one parameter, while leaving the others fixed. The prerequisite of ALS is that the optimization sub-problem can be analytically solved. Here, we describe how Hu's work [12] solves this problem.

First, minimizing J with respect to user latent vector $\mathbf{p}_u$ is equivalent to minimizing:

$$J_u = ||\mathbf{W}^u(\mathbf{r}_u - \mathbf{Q}\mathbf{p}_u)||^2 + \lambda||\mathbf{p}_u||^2,$$

where $\mathbf{W}^u$ is a $N \times N$ diagonal matrix with $W^u_{ii} = w_{ui}$. The minimum is where the first-order derivative is 0:

$$\begin{aligned} &\frac{\partial J_u}{\partial \mathbf{p}_u} = 2\mathbf{Q}^T\mathbf{W}^u\mathbf{Q}\mathbf{p}_u - 2\mathbf{Q}^T\mathbf{W}^u\mathbf{r}_u + 2\lambda\mathbf{p}_u = 0 \\ \Rightarrow \quad &\mathbf{p}_u = (\mathbf{Q}^T\mathbf{W}^u\mathbf{Q} + \lambda\mathbf{I})^{-1}\mathbf{Q}^T\mathbf{W}^u\mathbf{r}_u, \end{aligned} \quad (3)$$

where $\mathbf{I}$ denotes the identity matrix. This analytical solution is also known as the *ridge regression* [23]. Following the same process, we can get the solution for $\mathbf{q}_i$.

3.2.1 Efficiency Issue with ALS

As we can see, in order to update a latent vector, inverting a $K \times K$ matrix is inevitable. Matrix inversion is an expensive operation, usually assumed $O(K^3)$ in time complexity [12]. As such, updating one user latent vector takes time $O(K^3 + NK^2)$. Thus, the overall time complexity of one iteration that updates all model parameters once is $O((M+N)K^3 + MNK^2)$. Clearly, this high complexity makes the algorithm impractical to run on large-scale data, where there can be millions of users and items and billions of interactions.

Speed-up with Uniform Weighting. To reduce the high time complexity, Hu *et al.* [12] applied a uniform weight to missing entries; *i.e.*, assuming that all zero entries in $\mathbf{R}$ have a same weight w_0. Through this simplification, they can speed up the computation with memoization:

$$\mathbf{Q}^T\mathbf{W}^u\mathbf{Q} = w_0\mathbf{Q}^T\mathbf{Q} + \mathbf{Q}^T(\mathbf{W}^u - \mathbf{W}^0)\mathbf{Q}, \quad (4)$$

where $\mathbf{W}^0$ is a diagonal matrix that each diagonal element is w_0. As $\mathbf{Q}^T\mathbf{Q}$ is independent of u, it can be pre-computed for updating all user latent vectors. Considering the fact that $\mathbf{W}^u - \mathbf{W}^0$ only has $|\mathcal{R}_u|$ non-zero entries, we can compute Eq. (4) in $O(|\mathcal{R}_u|K^2)$ time. Thus, the time complexity of ALS is reduced to $O((M+N)K^3 + |\mathcal{R}|K^2)$.

Even so, we argue that the $O((M+N)K^3)$ term can be a major cost when $(M+N)K \geq |\mathcal{R}|$. In addition, the $O(|\mathcal{R}|K^2)$ part is still much higher than in SGD [25], which only requires $O(|\mathcal{R}|K)$ time. As a result, even with the acceleration, ALS is still prohibitive for running on large data, where large K is crucial as it can lead to better generalizability and thus better prediction performance. Moreover, the uniform weighting assumption is usually invalid in real applications and adversely degrades model's predictiveness. This thus motivates us to design an efficient implicit MF method not subject to uniform-weights.

3.3 Generic Element-wise ALS Learner

The bottleneck of the previous ALS solution lies in the matrix inversion operation, which is due to the design that updates the latent vector for a user (item) as a whole. As such, it is natural to optimize parameters at the element level — optimizing each coordinate of the latent vector, while leaving the others fixed [26]. To achieve this, we first get the derivative of objective function Eq. (2) with respect to p_{uf}:

$$\frac{\partial J}{\partial p_{uf}} = -2\sum_{i=1}^{N}(r_{ui} - \hat{r}^f_{ui})w_{ui}q_{if} + 2p_{uf}\sum_{i=1}^{N} w_{ui}q_{if}^2 + 2\lambda p_{uf},$$

where $\hat{r}^f_{ui} = \hat{r}_{ui} - p_{uf}q_{if}$, *i.e.,* the prediction without the component of latent factor f. By setting this derivative to 0, we obtain the solution of p_{uf}:

$$p_{uf} = \frac{\sum_{i=1}^{N}(r_{ui} - \hat{r}^f_{ui})w_{ui}q_{if}}{\sum_{i=1}^{N} w_{ui}q_{if}^2 + \lambda}. \quad (5)$$

Similarly, we can get the solver for an item latent factor:

$$q_{if} = \frac{\sum_{u=1}^{M}(r_{ui} - \hat{r}^f_{ui})w_{ui}p_{uf}}{\sum_{i=1}^{M} w_{ui}p_{uf}^2 + \lambda}. \quad (6)$$

Given the closed-form solution that optimizes one parameter with other fixed, the algorithm iteratively executes it for all model parameters until a joint optimum is reached. Due to the non-convexity of the objective function, critical points where gradients vanish can be local minima.

Time Complexity. As can be seen, by performing optimization at the element level, the expensive matrix inversion can be avoided. A raw implementation takes $O(MNK^2)$ time for one iteration, directly speeding up ALS by eliminating the $O(K^3)$ term. Moreover, by pre-computing $\hat{r}_{ui}$ [26], we can calculate $\hat{r}^f_{ui}$ in $O(1)$ time rather than $O(K)$. As such, the complexity can be further reduced to $O(MNK)$, which is the same magnitude with evaluating all the user–item predictions.

4. OUR IMPLICIT MF METHOD

We first propose an item-oriented weighting scheme on the missing data, and follow with a popularity-aware weighting strategy, which is arguably more effective than the uniform weighting for the recommendation task. Then, we develop a fast eALS algorithm to optimize the objective function that significantly reduces learning complexity comparing with the conventional ALS [12] and generic element-wise ALS learner [26]. Lastly, we discuss how to adjust the learning algorithm for real-time online learning.

4.1 Item-Oriented Weighting on Missing Data

Due to the large space of items, the missing entries for a user are a mixture of negative and unknown feedback. In specifying the weight w_{ui} of missing entries, it is desired to assign a higher weight to the negative feedback. However, it is a well-known difficulty to differentiate the two cases. In addition, as the interaction matrix $\mathbf{R}$ is usually large and sparse, it will be too consuming to store each zero entry an individualized weight. To this end, existing works [4, 12, 23, 30, 31] have applied a simple uniform weight on missing entries, which are, however, suboptimal and non-extendable for real applications.

Considering the ease of content providers in accessing negative information of the item side (*e.g.,* which items have been promoted to users but receive little interaction), we believe it is more realistic to weight missing data based on some item property. To capture this, we devise a more fine-grained objective function as follows:

$$L = \sum_{(u,i)\in\mathcal{R}} w_{ui}(r_{ui} - \hat{r}_{ui})^2 + \sum_{u=1}^{M}\sum_{i\notin\mathcal{R}_u} c_i \hat{r}_{ui}^2 + \lambda(\sum_{u=1}^{M}||\mathbf{p}_u||^2 + \sum_{i=1}^{N}||\mathbf{q}_i||^2), \tag{7}$$

where c_i denotes the confidence that item i missed by users is a true negative assessment, which can serve as a means to encode domain knowledge from practitioners. It is clear that the first term denotes the prediction error of the observed entries, which has been widely adopted in modeling explicit ratings [15, 27]. The second term accounts for the missing data, which acts as the role of negative instances and is crucial for recommendation from implicit feedback [12, 25]. Next, we present a domain-independent strategy to determine c_i by leveraging a ubiquitous feature of modern Web 2.0 systems.

4.1.1 Popularity-aware Weighting Strategy

Existing visual interfaces of many Web 2.0 systems showcase popular items in their recommendations. All other factors being equal, popular items are more likely to be known by users in general [10], and thus it is reasonable to think that a miss on a popular item is more probable to be truly irrelevant (as opposed to unknown) to the user. To account for this effect, we parametrize c_i based on item's popularity:

$$c_i = c_0 \frac{f_i^\alpha}{\sum_{j=1}^{N} f_j^\alpha}, \tag{8}$$

where f_i denotes the popularity of item i, given by its frequency in the implicit feedback data: $|\mathcal{R}_i|/\sum_{j=1}^{N}|\mathcal{R}_j|$, and c_0 determines the overall weight of missing data. Exponent α controls the significance level of popular items over unpopular ones — when $\alpha > 1$ the weights of popular items are promoted to strengthen the difference against unpopular ones; while setting α within the lower range of $(0, 1)$ suppresses the weight of popular items and has a smoothing effect. We empirically find $\alpha = 0.5$ usually leads to good results. Note that the uniform weighting is a special case by setting α to 0 with $w_0 = c_0/N$.

Relationship to Negative Sampling. Our proposed popularity-aware weighting strategy has the same intuition with Rendle's popularity-based oversampling [24] for learning BPR, which basically samples popular items as negative feedback with a higher probability. However, [24] empirically shows the oversampling method underperforms the basic uniform sampler. We suspect the reason comes from the SGD learner, which will result in more gradient steps on popular items, due to oversampling. As a result, popular items may be over-trained locally at the expense of less popular items which would then be under-trained. To resolve this, tricks like subsampling frequent items [19] and adaptive learning rates like Adagrad [6] have been adopted in other domains. As the focus of this paper is on whole-data based implicit MF, we do not further explore the details of SGD. It is worth pointing out that our proposed eALS learner avoids these learning issues by an exact optimization on each model parameter.

4.2 Fast eALS Learning Algorithm

We can speed up learning by avoiding the massive repeated computations introduced by the weighted missing data. We detail the derivation process for p_{uf}; where the counterpart for q_{if} is achieved likewise.

First, we rewrite the p_{uf} update rule Eq. (5) by separating the observed data part:

$$p_{uf} = \frac{\sum_{i\in\mathcal{R}_u}(r_{ui} - \hat{r}_{ui}^f)w_{ui}q_{if} - \sum_{i\notin\mathcal{R}_u}\hat{r}_{ui}^f c_i q_{if}}{\sum_{i\in\mathcal{R}_u} w_{ui}q_{if}^2 + \sum_{i\notin\mathcal{R}_u} c_i q_{if}^2 + \lambda}.$$

Clearly, the computational bottleneck lies in the summation over missing data portion, which requires a traversal of the whole negative space. We first focus on the numerator:

$$\begin{aligned}\sum_{i\notin\mathcal{R}_u}\hat{r}_{ui}^f c_i q_{if} &= \sum_{i=1}^{N} c_i q_{if}\sum_{k\neq f} p_{uk}q_{ik} - \sum_{i\in\mathcal{R}_u}\hat{r}_{ui}^f c_i q_{if} \\ &= \sum_{k\neq f} p_{uk}\sum_{i=1}^{N} c_i q_{if}q_{ik} - \sum_{i\in\mathcal{R}_u}\hat{r}_{ui}^f c_i q_{if}.\end{aligned} \tag{9}$$

By this reformulation, we can see that the major computation — the $\sum_{i=1}^{N} c_i q_{if} q_{ik}$ term that iterates over all items — is independent of u. However, a naïve implementation repeatedly computes it unnecessarily, when updating the latent factors for different users. Clearly, we can achieve a significant speed-up by memoizing it.

We define the $\mathbf{S}^q$ cache as $\mathbf{S}^q = \sum_{i=1}^{N} c_i \mathbf{q}_i \mathbf{q}_i^T$, which can be pre-computed and used in updating the latent factors for all users. Then, Eq. (9) can be evaluated as:

$$\sum_{i\notin\mathcal{R}_u}\hat{r}_{ui}^f c_i q_{if} = \sum_{k\neq f} p_{uk}s_{fk}^q - \sum_{i\in\mathcal{R}_u}\hat{r}_{ui}^f c_i q_{if}, \tag{10}$$

which can be done in $O(K + |\mathcal{R}_u|)$ time.

Similarly, we can apply the cache to speed up the calculation of denominator:

$$\sum_{i\notin\mathcal{R}_u} c_i q_{if}^2 = \sum_{i=1}^{N} c_i q_{if}^2 - \sum_{i\in\mathcal{R}_u} c_i q_{if}^2 = s_{ff}^q - \sum_{i\in\mathcal{R}_u} c_i q_{if}^2. \tag{11}$$

To summarize the above memoization strategy, we give the update rule for p_{uf} with the use of $\mathbf{S}^q$ cache:

$$p_{uf} = \frac{\sum_{i\in\mathcal{R}_u}[w_{ui}r_{ui} - (w_{ui} - c_i)\hat{r}_{ui}^f]q_{if} - \sum_{k\neq f} p_{uk}s_{kf}^q}{\sum_{i\in\mathcal{R}_u}(w_{ui} - c_i)q_{if}^2 + s_{ff}^q + \lambda}. \tag{12}$$

Algorithm 1: Fast eALS Learning algorithm.

Input: $\mathbf{R}$, K, λ, $\mathbf{W}$ and item confidence vector $\mathbf{c}$;
Output: Latent feature matrix $\mathbf{P}$ and $\mathbf{Q}$;

1 Randomly initialize $\mathbf{P}$ and $\mathbf{Q}$;
2 **for** $(u,i) \in \mathcal{R}$ **do** $\hat{r}_{ui} \leftarrow$ Eq. (1) ; ▷ $O(|\mathcal{R}|K)$
3 **while** *Stopping criteria is not met* **do**
    `// Update user factors`
4 $\mathbf{S}^q = \sum_{i=1}^{N} c_i \mathbf{q}_i \mathbf{q}_i^T$; ▷ $O(MK^2)$
5 **for** $u \leftarrow 1$ **to** M **do** ▷ $O(MK^2 + |\mathcal{R}|K)$
6 **for** $f \leftarrow 1$ **to** K **do**
7 **for** $i \in \mathcal{R}_u$ **do** $\hat{r}_{ui}^f \leftarrow \hat{r}_{ui} - p_{uf} q_{if}$;
8 $p_{uf} \leftarrow$ Eq. (12) ; ▷ $O(K + |\mathcal{R}_u|)$
9 **for** $i \in \mathcal{R}_u$ **do** $\hat{r}_{ui} \leftarrow \hat{r}_{ui}^f + p_{uf} q_{if}$;
10 **end**
11 **end**
    `// Update item factors`
12 $\mathbf{S}^p \leftarrow \mathbf{P}^T \mathbf{P}$; ▷ $O(NK^2)$
13 **for** $i \leftarrow 1$ **to** N **do** ▷ $O(NK^2 + |\mathcal{R}|K)$
14 **for** $f \leftarrow 1$ **to** K **do**
15 **for** $u \in \mathcal{R}_i$ **do** $\hat{r}_{ui}^f \leftarrow \hat{r}_{ui} - p_{uf} q_{if}$;
16 $q_{if} \leftarrow$ Eq. (13) ; ▷ $O(K + |\mathcal{R}_i|)$
17 **for** $u \in \mathcal{R}_i$ **do** $\hat{r}_{ui} \leftarrow \hat{r}_{ui}^f + p_{uf} q_{if}$;
18 **end**
19 **end**
20 **end**
21 **return** $\boldsymbol{P}$ *and* $\boldsymbol{Q}$

Table 1: Time complexity of implicit MF methods.

Method	Time Complexity
ALS (Hu *et al.*[12])	$O((M+N)K^3 + \|\mathcal{R}\|K^2)$
BPR (Rendle *et al.*[25])	$O(\|\mathcal{R}\|K)$
IALS1 (Pilászy *et al.*[23])	$O(K^3 + (M+N)K^2 + \|\mathcal{R}\|K)$
ii-SVD (Volkovs *et al.*[31])	$O((M+N)K^2 + MN \log K)$
RCD (Devooght *et al.*[4])	$O((M+N)K^2 + \|\mathcal{R}\|K)$
eALS (Algorithm 1)	$O((M+N)K^2 + \|\mathcal{R}\|K)$

$|\mathcal{R}|$ denotes the number of non-zeros in user–item matrix $\mathbf{R}$.

Similarly, we can derive the update rule for q_{if}:

$$q_{if} = \frac{\sum_{u \in \mathcal{R}_i} [w_{ui} r_{ui} - (w_{ui} - c_i)\hat{r}_{ui}^f] p_{uf} - c_i \sum_{k \neq f} q_{ik} s_{kf}^p}{\sum_{u \in \mathcal{R}_i} (w_{ui} - c_i) p_{uf}^2 + c_i s_{ff}^p + \lambda}, \tag{13}$$

where s_{kf}^p denotes the $(k,f)^{th}$ element of the $\mathbf{S}^p$ cache, defined as $\mathbf{S}^p = \mathbf{P}^T \mathbf{P}$.

Algorithm 1 summarizes the accelerated algorithm for our element-wise ALS learner, or **eALS**. For convergence, one can either monitor the value of objective function on training set or check the prediction performance on a hold-out validation data.

4.2.1 Discussion

Time Complexity. In Algorithm 1, updating a user latent factor takes $O(K + |\mathcal{R}_u|)$ time. Thus, one eALS iteration takes $O((M+N)K^2 + |R|K)$ time. Table 1 summarizes the time complexity (of one iteration or epoch) of other MF algorithms that are designed for implicit feedback.

Comparing with the vector-wise ALS [12, 21], our element-wise ALS learner is K times faster. In addition, our proposed eALS has the same time complexity with RCD [4], being faster than ii-SVD [31], another recent solution. RCD is a state-of-the-art learner for whole-data based MF, which performs a gradient descent step on a randomly chosen latent vector. Since it requires a good learning rate, the work [4] adaptively determines it by a line search in each gradient step, which essentially chooses the learning rate that leads to the steepest descent among pre-defined candidates. A major advantage of eALS has over RCD is that it avoids the need for a learning rate by an exact optimization in each parameter update, arguably more effective and easier to use than RCD. The most efficient algorithm is BPR, which applies the SGD learner on sampled, partial missing data only.

Computing the Objective Function. Evaluating the objective function is important to check the convergence of iterations and also to verify the correctness of implementation. A direct calculation takes $O(MNK)$ time, requiring a full estimation on the $\mathbf{R}$ matrix. Fortunately, with the item-oriented weighting, we can similarly exploit the sparseness of $\mathbf{R}$ for acceleration. To achieve this, we reformulate the loss of the missing data part that causes the major cost:

$$\sum_{u=1}^{M} \sum_{i \notin \mathcal{R}_u} c_i \hat{r}_{ui}^2 = \sum_{u=1}^{M} \mathbf{p}_u^T \mathbf{S}^q \mathbf{p}_u - \sum_{(u,i) \in \mathcal{R}} c_i \hat{r}_{ui}^2. \tag{14}$$

By reusing $\mathbf{S}^q$ and the prediction cache $\hat{r}_{ui}$, we can calculate the objective function in $O(|\mathcal{R}| + MK^2)$ time, much faster than with direct calculation.

Parallel Learning. The iterations of eALS can be easily parallelized. First, computing the $\mathbf{S}$ caches (line 4 and 12) is the standard matrix multiplication operation, for which modern matrix toolkits provide very efficient and parallelized implementation. Second, in updating the latent vectors for different users (line 5-11), the shared parameters are either independent with each other (*i.e.,* $\hat{r}_{ui}$) or remaining unchanged (*i.e.,* $\mathbf{S}^q$). This nice property means that an exact parallel solution can be obtained by separating the updates by users; that is, letting different workers update the model parameters for disjoint sets of users. The same parallelism can also be achieved in updating item latent vectors.

This is an advantage over the commonly-used SGD learner, which is a stochastic method that updates model parameters given a training instance. In SGD, different gradient steps can influence with each other and there is no exact way to separate the updates for workers. Thus, sophisticated strategies are required to control the possible losses introduced by parallelization [7]. Our proposed eALS solution optimizes by coordinate descent where in each step a dedicated parameter is updated, making the algorithm embarrassingly parallel without any approximate loss.

4.3 Online Update

In practice, after a recommender model is trained offline on historical data, it will be used online and will need to adapt to best serve users. Here, we consider the online learning scenario that refreshes model parameters given a new user–item interaction.

Incremental Updating. Let $\hat{\mathbf{P}}$ and $\hat{\mathbf{Q}}$ denote the model parameters learnt from offline training, and (u,i) denotes the new interaction streamed in. To approximate the model parameters in accounting for the new interaction, we perform optimization steps for $\mathbf{p}_u$ and $\mathbf{q}_i$ only. The underlying assumption is that the new interaction should not change $\hat{\mathbf{P}}$ and $\hat{\mathbf{Q}}$ too much from a global perspective, while it should

Algorithm 2: Online Incremental Updates for eALS.

Input: $\hat{\mathbf{P}}$, $\hat{\mathbf{Q}}$, new interaction (u, i) and its weight w_{new}
Output: Refreshed parameters $\mathbf{P}$ and $\mathbf{Q}$;

1 $\mathbf{P} \leftarrow \hat{\mathbf{P}}$; $\mathbf{Q} \leftarrow \hat{\mathbf{Q}}$; ;
2 **If** u is a new user **do** Randomly initialize $\mathbf{p}_u$;
3 **If** i is a new item **do** Randomly initialize $\mathbf{q}_i$;
4 $r_{ui} \leftarrow 1$; $\hat{r}_{ui} \leftarrow Eq.\ (1)$; $w_{ui} \leftarrow w_{new}$;
5 **while** *Stopping criteria is not met* **do**
`// Line 6-10 of Algorithm 1`
6 update_user(u); ; ▷ $O(K^2 + |\mathcal{R}_u|K)$
7 update_cache(u, $\mathbf{S}^p$); ; ▷ $O(K^2)$
`// Line 14-18 of Algorithm 1`
8 update_item(i); ; ▷ $O(K^2 + |\mathcal{R}_i|K)$
9 update_cache(i, $\mathbf{S}^q$); ; ▷ $O(K^2)$
10 **end**
11 **return** $\boldsymbol{P}$ *and* $\boldsymbol{Q}$

change the local features for u and i significantly. Particularly, when u is a new user, executing the local updates will force $\mathbf{p}_u$ close to $\mathbf{q}_i$, which meets the expectation of latent factor model. The new item case is similar.

Algorithm 2 summarizes the incremental learning strategy for eALS. For the stopping criteria, our empirical study shows that one iteration is usually sufficient to get good results. Moreover, it is important to note that after updating a latent vector, we need to update the $\mathbf{S}$ cache accordingly.

Weight of New Interactions. In an online system, new interactions are more reflective of a user's short-term interest. Comparing to the historical interactions used in offline training, fresh data should be assigned a higher weight for predicting user's future action. We assign a weight w_{new} to each new interaction (line 4 of Algorithm 2) as a tunable parameter. Later in Section 5.3, we investigate how the setting of this parameter impacts online learning performance.

Time Complexity. The incremental update for a new interaction (u, i) can be done in $O(K^2+(|\mathcal{R}_u|+|\mathcal{R}_i|)K)$ time. It is worth noting that the cost depends on the number of observed interactions for u and i, while being independent with number of total interactions, users and items. This localized complexity make the online learning algorithm suitable to deployment in industrial use, as the complex software stack that deals with data dependencies [29] can be avoided.

5. EXPERIMENTS

We begin by introducing the experimental settings. Then we perform an empirical study with the traditional offline protocol, followed by a more realistic online protocol.

5.1 Experimental Settings

Datasets. We evaluate on two publicly accessible datasets: Yelp[1] and Amazon Movies[2]. We transform the review dataset into implicit data, where each entry is marked as 0/1 indicating whether the user reviewed the item. Since the high sparsity of the original datasets makes it difficult to evaluate recommendation algorithms (*e.g.,* over half users have only one review), we follow the common practice [25] to filter out users and items with less than 10 interactions. Table 2 summarizes the statistics of the filtered datasets.

Table 2: Statistics of the evaluation datasets.

Dataset	Review#	Item#	User#	Sparsity
Yelp	731,671	25,815	25,677	99.89%
Amazon	5,020,705	75,389	117,176	99.94 %

Methodology. We evaluate using two protocols:

- **Offline Protocol**. We adopt the *leave-one-out* evaluation, where the latest interaction of each user is held out for prediction and the models are trained on the remaining data. Although it is a widely used evaluation protocol in the literature [9, 25], we point out that it is an artificial split that does not correspond to the real recommendation scenario. In addition, the new users problem is averted in this evaluation, as each test user has a training history. Thus this protocol only evaluates an algorithm's capability in providing one-shot recommendation for existing users by leveraging the static history data.

- **Online Protocol**. To create a more realistic recommendation scenario, we simulate the dynamic data stream. We first sort all interactions in chronological order, training models on the first 90% of the interactions and holding out the last 10% for testing. In the testing phase, given a test interaction (*i.e.,* a user–item pair) from the hold-out data, the model first recommends a ranked list of items to the user; the performance is judged based on the ranked list. Then the test interaction is fed into the model for an incremental update. Note that with the global split by time, 14% and 57% test interactions are from new users for the Yelp and Amazon dataset, respectively. Overall this protocol evaluates an online learning algorithm's effectiveness in digesting the dynamic new data.

To assess the ranked list with the ground-truth (GT) item that user actually consumed, we adopt *Hit Ratio* (HR) and *Normalized Discounted Cumulative Gain* (NDCG). We truncate the ranked list at 100 for both metrics. HR measures whether the ground truth item is present on the ranked list, while NDCG accounts for the position of hit [9]. We report the score averaged by all test interactions.

Baselines. We compare with the following methods:

- **ALS** [12]. This is the conventional ALS method that optimizes the whole-data based MF. Due to the high time complexity, this method is infeasible in a real-time dynamic updating scenario, so we only evaluate it with the offline protocol.

- **RCD** [4]. This is the state-of-the-art implicit MF method that has the same time complexity with eALS and is suitable for online learning. For the line search parameters, we use the suggested values in the authors' implementation[3].

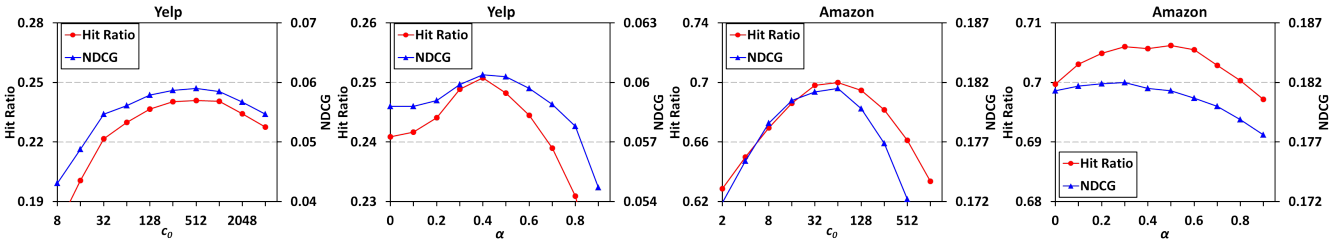
- **BPR** [25]. This is a sample-based method that optimizes the pair-wise ranking between the positive and negative samples. It learns by SGD, which can be adjusted to online incremental learning by [27]. We use a fixed learning rate, varying it and reporting the best performance.

Parameter Settings. For the weight of observed interactions, we set it uniformly as 1, a default setting by previous works [4, 25]. For regularization, we set λ as 0.01 for all methods for a fair comparison. All methods are implemented in Java and running on the same machine (Intel

[1]We used the Yelp Challenge dataset downloaded on October 2015 that contained 1.6 million reviews: http://www.yelp.com/dataset_challenge

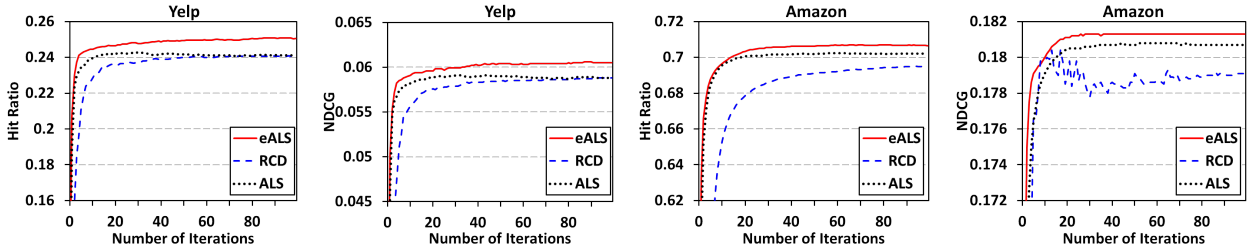
[2]http://snap.stanford.edu/data/web-Amazon-links.html

[3]https://github.com/rdevooght/MF-with-prior-and-updates



(a) eALS vs. c_0 ($\alpha = 0$) **(b) eALS vs. α ($c_0 = 512$)** **(c) eALS vs. c_0 ($\alpha = 0$)** **(d) eALS vs. α ($c_0 = 64$)**

Figure 1: Impact of weighting parameters c_0 and α on eALS's performance evaluated by offline protocol.



(a) Iterations vs. HR **(b) Iterations vs. NDCG** **(c) Iterations vs. HR** **(d) Iterations vs. NDCG**

Figure 2: Prediction accuracy of three whole-data based MF methods in each iteration ($K = 128$).

Xeon 2.67GHz CPU and 24GB RAM) in a single-thread for a fair comparison on efficiency. As the findings are consistent across the number of factors K, without any particular outlier, we only show the results of $K = 128$, a relatively large number that maintains good accuracy.

5.2 Offline Protocol

We first study how does the weighting scheme on missing data impact eALS's performance. Then we compare with the whole-data based implicit MF methods ALS and RCD, as well as the sample-based ranking method BPR.

5.2.1 *Weight of Missing Data*

In Section 4.1, we propose an item popularity-aware weighting strategy, which has two parameters: c_0 determines the overall weight of missing data and α controls the weight distribution. First, we set a uniform weight distribution (*i.e.,* $\alpha = 0$), varying c_0 to study how does the weight of missing data impact the performance. For Yelp (Figure 1a), the peak performance is achieved when c_0 is around 512, corresponding to that the weight of each zero entry is 0.02 ($w_0 = c_0/N$); similarly for Amazon (Figure 1c), the optimal c_0 is around 64, corresponding to $w_0 = 0.0001$. When c_0 becomes smaller (where w_0 is close to 0), the performance degrades significantly. This highlights the necessity of accounting for the missing data when modeling implicit feedback for item recommendation. Moreover, when c_0 is set too large, the performance also suffers. Based on this observation, we believe the traditional SVD technique [2] that treats all entries equally weighted will be suboptimal here.

Then, we set c_0 to the best value (in the case of $\alpha = 0$), varying α to check the performance change. As can be seen from Figure 1b and 1d, the performance of eALS is gradually improved with the increase of α, and the best result is reached around 0.4. We further conducted the one-sample paired t-test, verifying that the improvements are statistically significant (p-value < 0.01) for both metrics on the two datasets. This indicates the effectiveness of our popularity-biased weighting strategy. Moreover, when α is larger than 0.5, the performance starts to drop significantly. This reveals the drawback of over-weighting popular items as negative instances, thus the importance of accounting for less popular items with a proper weight.

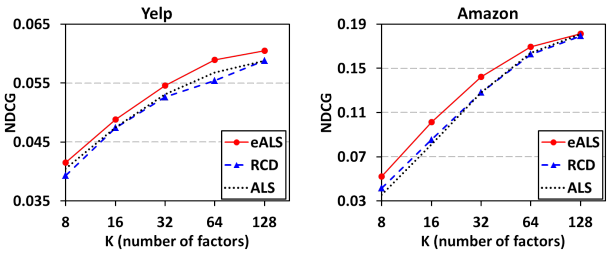


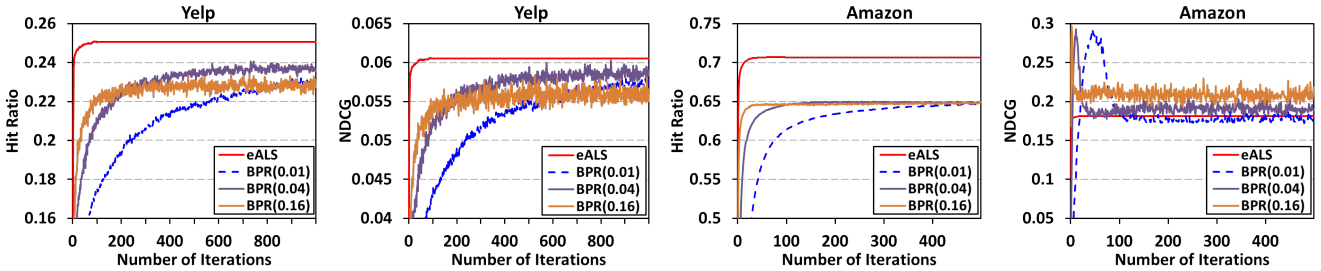
Figure 3: NDCG of whole-data based MF across K.

In the following experiments, we fix c_0 and α according to the best performance evaluated by HR, *i.e.,* $c_0 = 512, \alpha = 0.4$ for Yelp and $c_0 = 64, \alpha = 0.5$ for Amazon.

5.2.2 *Compare Whole-data based MF Methods*

We performed the same grid search of w_0 for RCD and ALS and reported the best performance.

Convergence. Figure 2 shows the prediction accuracy with respect to number of iterations. First, we see that eALS achieves the best performance after converge. All improvements are statistically significant evidenced by the one-sample paired t-test ($p < 0.01$). We believe the benefits mainly come from the popularity-aware objective function, as both ALS and RCD apply a uniform weighting on the unknowns. Second, eALS and ALS converge faster than RCD. We think the reason is that (e)ALS updates a parameter to minimize the objective function of the current status, while RCD updates towards the direction of the negative gradient, which can be suboptimal. On Amazon, RCD shows high but turbulent NDCG in early iterations, while the low hit ratio and later iterations indicate the high NDCG is unstable. Finally, we point out that in optimizing the same objective function, ALS outperforms RCD in most cases, demonstrating the advantage of ALS over the gradient descent learner.



(a) Iterations vs. HR (b) Iterations vs. NDCG (c) Iterations vs. HR (d) Iterations vs. NDCG

Figure 4: Accuracy and convergence comparison of eALS with BPR of different learning rate.

Accuracy vs. Number of Factors. Figure 3 shows the prediction accuracy with varying number of factors K. We only show the evaluation by NDCG as HR admits the same trend. First, eALS consistently outperforms ALS and RCD across K, demonstrating the effectiveness of our eALS method (*n.b.* although the three methods seem to perform on par for Amazon at $K = 128$, their difference can be clearly seen in Figure 2d). Second, all methods can be improved significantly with a larger K. Although a large K might have the risk of overfitting, it can increase model's representation ability thus better prediction. Especially for large datasets that can have millions of users and billions of interactions, a large K is particularly important for the accuracy of MF methods.

Efficiency. Analytically, ALS's time complexity is $O((M+N)K^3 + |\mathcal{R}|K^2)$, while eALS and RCD are K times faster. To compare their efficiency empirically, we show the actual training time per iteration in Table 3.

Table 3: Training time per iteration of different whole-based MF methods with varying K.

	Yelp			Amazon		
K	**eALS**	**RCD**	**ALS**	**eALS**	**RCD**	**ALS**
32	1s	1s	10s	9s	10s	74s
64	4s	3s	46s	23s	17s	4.8m
128	13s	10s	221s	72s	42s	21m
256	1m	0.9m	23m	4m	2.8m	2h
512	2m	2m	2.5h	12m	9m	11.6h
1024	7m	11m	25.4h	54m	48m	74h

s, m, and h denote seconds, minutes and hours, respectively.

As can be seen, with the increase of K, ALS takes much longer time than eALS and RCD. Specifically, when K is 512, ALS requires 11.6 hours for one iteration on Amazon, while eALS only takes 12 minutes. Although eALS does not empirically show K times faster than ALS due to the more efficient matrix inversion implementation (we used the fastest known algorithm [1] with time complexity around $O(K^{2.376})$), the speed-up is already very significant. Moreover, as RCD and eALS have the same analytical time complexity, their actual running time are in the same magnitude; the minor difference can be caused by some implementation details, such as the data structures and caches used.

5.2.3 *eALS vs. BPR (sample-based)*

Figure 4 plots the performance of BPR with different learning rates[4] in each iteration. Note that we only run eALS for 100 iterations, which are enough for eALS to converge. First, it is clear that BPR's performance is subjected to the choice of learning rate — Figure 4a and 4b show that a higher learning rate leads to a faster convergence, while the final accuracy may be suffered. Second, we see that eALS significantly outperforms BPR on the Yelp dataset evaluated by both measures ($p < 0.001$). For Amazon, eALS obtains a much higher hit ratio but a lower NDCG score, indicating that most hits occur at a relatively low ranks for eALS. Comparing with the performance of other whole-based MF methods ALS and RCD (Figure 2), we draw the conclusion that BPR is a weak performer in terms of the prediction recall, while being a strong performer in terms of the precision at top ranks. We think BPR's strength in ranking top items is due to its optimization objective, which is a pairwise ranking function tailored for ranking correct item high. In contrast, the regression-based objective is not directly optimized for ranking; instead, by account for all missing data in regression, it better predicts user's preference on unconsumed items, leading to a better recall. This is consistent with [2]'s finding in evaluating top-K recommendation.

We notice that BPR shows unusual NDCG spike in early iterations on the Amazon dataset, however the performance is unstable and goes down with more iterations. The same phenomenon was also observed for another gradient descent method RCD on the same dataset (see Figure 2d). We hypothesize that it might be caused by some regularities in the data. For example, we find some Amazon users review on a movie multiple times[5]. In early iterations, BPR ranks these repeated items high, leading to a high but unstable NDCG score. There might be other reasons responsible for this, and we do not further explore here.

5.3 Online Protocol

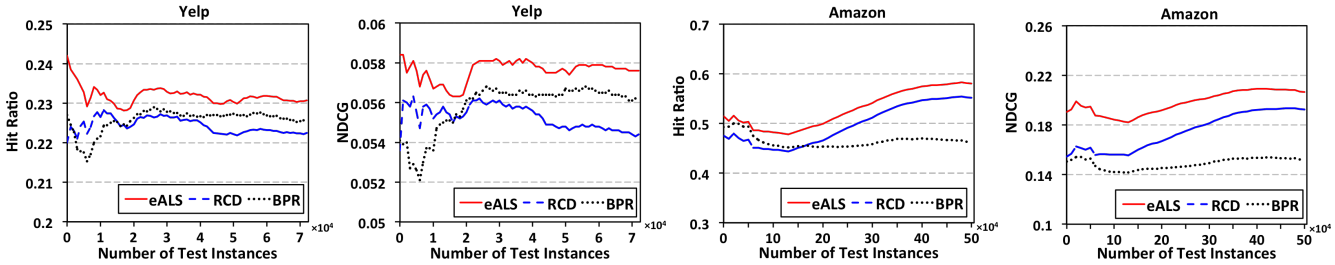
In the evaluation of online protocol, we hold out the latest 10% interactions as the test set, training all methods on the remaining 90% data with the best parameter settings evidenced by the offline evaluation. We first study the number of online iterations required for eALS to converge. Then we show how does the weight of new interactions impact the performance. Lastly, we compare with dynamic MF methods RCD and BPR in the online learning scenario.

5.3.1 *Number of Online Iterations*

Figure 6 shows how does eALS's accuracy change with number of online iterations. Results at the 0-th iteration benchmark the performance of the offline trained model, as no incremental update is performed. First, we can see that the offline trained model performs very poorly, highlight-

[4]We have also tried other intermediate values of learning rates, and the findings are consistent. Thus, to make the figure more clear, we only show three selected values.

[5]Due to user's repeat consumption behaviours, we do not exclude training items when generating recommend list.



(a) Test # vs. HR (b) Test # vs. NDCG (c) Test # vs. HR (d) Test # vs. NDCG

Figure 5: Performance evolution of eALS and other dynamic MF methods in online learning.

ing the importance of refreshing recommender model for an online system with dynamic data. Second, we find most performance gain comes from the first iteration, and more iterations do not further improve. This is due to the fact that only the local features regarding to the new interaction are updated, and one eALS step on a latent factor can find the optimal solution with others fixed. Thus, one iteration is enough for eALS to learn from a new interaction incrementally, making eALS very efficient for learning online.

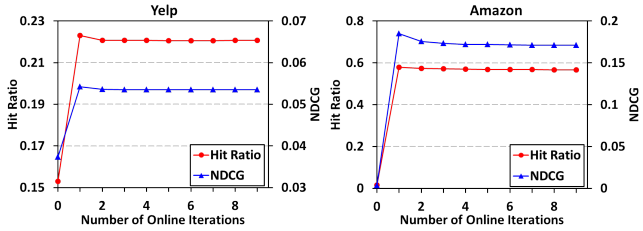


Figure 6: Impact of online iterations on eALS.

We have also investigated number of online iterations required for baselines RCD and BPR. RCD shows the same trend that good prediction is obtained in the first iteration. While BPR requires more iterations, usually 5-10 iterations to get a peak performance and more iterations will adversely hurt the performance due to the local over-training.

5.3.2 *Weight of New Interactions*

To evaluate how does the weight of new interactions effect the online learning algorithms, we also apply the same weight w_{new} on RCD. Note that the original RCD paper [4] does not consider the weight of interactions; we encode w_{new} the same way with eALS, revising the RCD learner to optimize the weighted regression function.

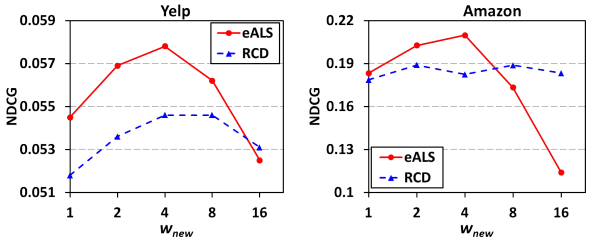


Figure 7: Impact of w_{new} on eALS and RCD in online learning evaluated by NDCG.

Figure 7 shows the performance evaluated by NDCG (results of HR show the same trend thus omitted for space). Setting w_{new} to 1 signifies that new interaction is assigned a same weight with the old training interaction. As expected, with a modest increasing on w_{new}, the prediction of both models is gradually improved, demonstrating the usefulness of strengthening user's short-term interest. The peak performance is obtained around 4, where eALS shows better prediction than RCD. Overly increasing w_{new} will adversely hurt the performance, admitting the utility of user's historical data used in offline training. Overall, this experiment indicates the importance of balancing user's short-term and long-term interest for quality recommendation.

5.3.3 *Performance Comparison*

With the simulated data stream, we show the performance evolution with respect to number of test instances in Figure 5. First, eALS consistently outperforms RCD and BPR evidenced by both measures, and one-sample paired t-test verifies that all improvements are statistically significant with $p < 0.001$. BPR betters RCD for Yelp, while underperforms for Amazon. Second, we observe the trend that the performance of dynamic learning first decreases, and then increases before becoming stable. This is caused by the new users problem — when there are few feedback for a user, the model can not personalize the user's preference effectively; with more feedbacks streaming in, the model can adapt itself to improve the preference modeling accordingly. To show this, we further breakdown the results of eALS by number of past interactions of test user in Figure 8.

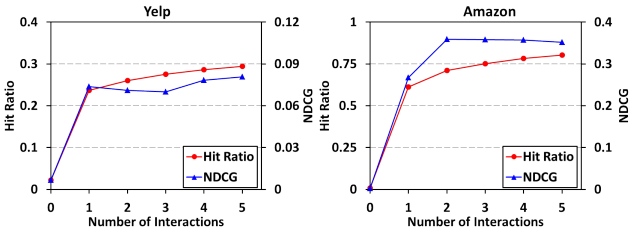


Figure 8: Results breakdown of eALS by # of past interactions of test user. Note: Interaction # > 0 denotes the performance for non-cold-start users.

It is clear that when there are no historical feedback for a test user (*i.e.,* user cold-start cases), the performance is very poor — no better than random. After the first interaction streams in, the prediction is significantly improved; and with more interactions, the performance is further improved. This highlights the importance of incorporating instantaneous user feedback into the model, especially for cold-start or sparse users that have few history in training.

6. CONCLUSION AND FUTURE WORK

We study the problem of learning MF models from implicit feedback. In contrast to previous work that applied

a uniform weight on missing data, we propose to weight missing data based on the popularity of items. To address the key efficiency challenge in optimization, we develop a new learning algorithm — eALS — which effectively learns parameters by performing coordinate descent with memoization. For online learning, we devise an incremental update strategy for eALS to adapt dynamic data in real-time. Experiments with both offline and online protocols demonstrate promising results. Importantly, our work makes MF more practical to use for modeling implicit data, along two dimensions. First, we investigate a new paradigm to deal with missing data which can easily incorporate prior domain knowledge. Second, eALS is embarrassingly parallel, making it attractive for large-scale industrial deployment.

We plan to study the optimal weighting strategy for online data as a way to explore user's short-term interest. Along the technical line, we explored the element-wise ALS learner in its basic MF form and solved the efficiency challenge in handling missing data. To make our method more applicable to real-world settings, we plan to encode side information such as user social contexts [8] and reviews [9] by extending eALS to more generic models, such as collective factorization [11] and Factorization machines [26]. In addition, we will study binary coding for MF on implicit data, since a recent advance [32] has shown that discrete latent factors are beneficial to collaborative filtering for explicit ratings.

The strength of eALS can be applied to other domains, owing to the universality of factorizing sparse data matrices. For example, recent advances in natural language processing [16] have shown the connection between neural word embeddings and MF on the word–context matrix. This bridge nicely motivates several proposals to use MF to learn word embeddings; however, when it comes to handling missing data, they have either ignored [22] or equally weighted the missing entries, similar to traditional SVD [16]. It will be interesting to see whether eALS will also improve these tasks.

Acknowledgement

The authors would like to thank the additional discussion and help from Steffen Rendle, Bhargav Kanagal, Immanuel Bayer, Tao Chen, Ming Gao and Jovian Lin.

7. REFERENCES

[1] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[2] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys 2010*, pages 39–46.

[3] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: Scalable online collaborative filtering. In *WWW 2007*, pages 271–280.

[4] R. Devooght, N. Kourtellis, and A. Mantrach. Dynamic matrix factorization with priors on unknown values. In *KDD 2015*, pages 189–198.

[5] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl. Real-time top-n recommendation in social streams. In *RecSys 2012*, pages 59–66.

[6] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011.

[7] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *KDD 2011*, pages 69–77.

[8] X. Geng, H. Zhang, J. Bian, and T.-S. Chua. Learning image and user features for recommendation in social networks. In *ICCV 2015*, pages 4274–4282.

[9] X. He, T. Chen, M.-Y. Kan, and X. Chen. Trirank: Review-aware explainable recommendation by modeling aspects. In *CIKM 2015*, pages 1661–1670.

[10] X. He, M. Gao, M.-Y. Kan, Y. Liu, and K. Sugiyama. Predicting the popularity of web 2.0 items based on user comments. In *SIGIR 2014*, pages 233–242.

[11] X. He, M.-Y. Kan, P. Xie, and X. Chen. Comment-based multi-view clustering of web 2.0 items. In *Proc. of WWW '14*, pages 771–782, 2014.

[12] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM 2008*, pages 263–272.

[13] Y. Huang, B. Cui, W. Zhang, J. Jiang, and Y. Xu. Tencentrec: Real-time stream recommendation in practice. In *SIGMOD 2015*, pages 227–238.

[14] Y. Koren. Collaborative filtering with temporal dynamics. In *KDD 2009*, pages 447–456.

[15] Y. Koren and R. Bell. Advances in collaborative filtering. In *Recommender systems handbook*, pages 145–186. Springer, 2011.

[16] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *NIPS 2014*, pages 2177–2185.

[17] G. Ling, H. Yang, I. King, and M. R. Lyu. Online learning for collaborative filtering. In *IJCNN 2012*, pages 1–8.

[18] B. M. Marlin, R. S. Zemel, S. Roweis, and M. Slaney. Collaborative filtering and the missing at random assumption. In *UAI 2007*, pages 267–276.

[19] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS 2013*, pages 3111–3119.

[20] R. Pan and M. Scholz. Mind the gaps: Weighting the unknown in large-scale one-class collaborative filtering. In *KDD 2009*, pages 667–676.

[21] R. Pan, Y. Zhou, B. Cao, N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *ICDM 2008*, pages 502–511.

[22] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP 2014*, pages 1532–1543.

[23] I. Pilászy, D. Zibriczky, and D. Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *RecSys 2010*, pages 71–78.

[24] S. Rendle and C. Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *WSDM 2014*, pages 273–282.

[25] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI 2009*, pages 452–461.

[26] S. Rendle, Z. Gantner, C. Freudenthaler, and L. Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *SIGIR 2011*, pages 635–644.

[27] S. Rendle and L. Schmidt-Thieme. Online-updating regularized kernel matrix factorization models for large scale recommender systems. In *RecSys 2008*, pages 251–258.

[28] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Math. Prog.*, 2014.

[29] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young. Machine learning: The high interest credit card of technical debt. In *SE4ML (NIPS 2014 Workshop)*, 2014.

[30] H. Steck. Training and testing of recommender systems on data missing not at random. In *KDD 2010*, pages 713–722.

[31] M. Volkovs and G. W. Yu. Effective latent models for binary feedback in recommender systems. In *SIGIR 2015*, pages 313–322.

[32] H. Zhang, F. Shen, W. Liu, X. He, H. Luan, and T.-S. Chua. Discrete collaborative filtering. In *SIGIR 2016*.