Dynamic Matrix Factorization with Priors on Unknown Values

Robin Devooght[*]
IRIDIA, ULB
1050 Brussels, Belgium
robin.devooght@ulb.ac.be

Nicolas Kourtellis[†]
Telefonica Research
08019 Barcelona, Spain
nicolas.kourtellis@telefonica.com

Amin Mantrach
Yahoo Labs
08018 Barcelona, Spain
amantrac@yahoo-inc.com

ABSTRACT

Advanced and effective collaborative filtering methods based on explicit feedback assume that unknown ratings do not follow the same model as the observed ones (*not missing at random*). In this work, we build on this assumption, and introduce a novel dynamic matrix factorization framework that allows to set an explicit prior on unknown values. When new ratings, users, or items enter the system, we can update the factorization in time independent of the size of data (number of users, items and ratings). Hence, we can quickly recommend items even to very recent users. We test our methods on three large datasets, including two very sparse ones, in static and dynamic conditions. In each case, we outrank state-of-the-art matrix factorization methods that do not use a prior on unknown ratings.

1. INTRODUCTION

Personalizing the user experience is a continuous growing challenge for various digital applications. This is of particular importance when recommending releases on the Netflix platform, when digesting latest Yahoo news, or for helping users to find their next musical obsession.

Among the different approaches towards personalization, matrix factorization ranges among the most popular ones [12, 30]. In this line of work, data is represented in the form of a user-item matrix, encoding user-item interactions in the form of binary or real values. Matrix factorization aims at decomposing a matrix into latent representations designed to accurately reconstruct observed interaction values. Most interestingly, these latent features are also used to predict missing (or unknown) ratings (i.e. if item j is exposed to user i, what would be his rating). However, by trying to predict the unknown ratings based on a model trained on the observed ratings, the recommender systems implicitly assume that the distribution of the observed ratings is representative of the distribution of the unknown ones. This is called the *missing at random* assumption [17], and it is probably a wrong asumption in most real-world applications. In the case of a movie recommender system, for example, users rate movies that they have seen, and their choices are biased by their interests.

In this work, building on the *not missing at random assumption* [18, 28] we make the hypothesis that it is more likely for an unknown item to be weakly rated, this due to the huge amounts of existing items coupled to the limited number of items a user may be interested in. This translates into a strong prior suggesting that unknown ratings should be reconstructed from latent features as small values (i.e. close to 0). While this assumption may be wrong for specific cases, such constraints act as a good regularizer that helps in significantly improving the recommendations.

Our work is not the first to propose new interpretation of the missing data in a matrix factorization framework [11, 20, 21, 28]. However, to the best of our knowledge, we are the first to propose an *online learning* mechanism that sets an explicit prior on unknown values and this, without any significant additional cost. We introduce a method to update our model each time a new rating is observed with a time complexity independent of the size of the data (i.e. the total number of users, items, and ratings). This fast update mechanism allows keeping the model up to date when a flow of new users, items and ratings enters the system.

The contributions of this work are as follows:

- We extend the squared loss, the absolute loss and the generalized Kullback-Liebler divergence to take into account an explicit prior on unknown values.
- For each loss function, we derive an efficient *online learning algorithm* to update the parameters of the model with a complexity independent of the data size.
- We validate the hypothesis that applying an explicit prior on missing ratings improves the recommendations in a static and in a dynamic setting on three public datasets.
- Our methods are easy to implement and we provide an open-source implementation of the squared loss and absolute loss.

The rest of this paper is organized as follows. Section 2 summarizes the recommendation problem and Section 3 formulates how to apply priors on unknown values in the context of recommendation. Section 4 extends three loss functions and shows how they can be optimized in a static and dynamic fashion. Section 5 presents our experimental results

[*]This work was carried out while the author was a research intern at Yahoo Labs, Barcelona.

[†]This work was carried out while the author was with Yahoo Labs, Barcelona.

KDD'15, August 11–14, 2015, Sydney, NSW, Australia
Copyright 2015 ACM. ISBN 978-1-4503-3664-2/15/08 ...$15.00
DOI: http://dx.doi.org/10.1145/2783258.2783346.

and Section 6 discusses works related to our study. Section 7 concludes this paper.

2. THE RECOMMENDATION PROBLEM

Before addressing the challenge of interpreting missing data, let us state the standard recommendation problem.

We have at our disposal m items rated by n different users, where the rating given by the i^{th} user to the j^{th} item is denoted by r_{ij}. In many real applications, these ratings take an integer value between 1 and 5. In this work, we assume that ratings are positive and that an item rated by user i with a high numerical value is preferred by this user over items she ranked with lower numerical values. We denote by $\mathcal{R}$ the set of all known ratings, and by $\mathcal{R}_{i\bullet}$ and $\mathcal{R}_{\bullet j}$ the set of known ratings of user i and item j, respectively. If $r_{ij} \notin \mathcal{R}$ we say that the rating is *unknown*.

For a while, the objective of recommender systems has been to predict the *value* of unknown ratings [12]. It is now widely accepted that a more practical goal is to correctly *rank* the unknown ratings for each user, while the actual value of the rating is of little interest [1, 5, 15, 20]. This has led to a change in the way methods are evaluated (in terms of ranking metrics such as NDCG, AUC or MAP, instead of rating prediction metrics as measured by RMSE). We embrace that shift towards *ranking*, and the purpose of adding a prior on the unknown ratings is not to improve matrix factorization techniques in terms of RMSE, but in terms of ranking metrics.

Matrix factorization methods produce for each user and each item a vector of k ($<< n$ and m) real values that we call *latent features*. We denote by w_i the row vector containing the k features of the i^{th} user, and h_j the row vector, composed of k features, associated to the j^{th} item. Also, we denote by $\mathbf{W}$ the $n \times k$ matrix whose i^{th} row is $\mathbf{w}_i$, and $\mathbf{H}$ as the $k \times m$ matrix whose j^{th} column is $\mathbf{h}_j^T$. Matrix factorization is presented as an optimization problem, whose general form is:

$$\operatorname*{arg\,min}_{\mathbf{W},\mathbf{H}} \sum_{i,j|r_{ij}\in\mathcal{R}} E\left(r_{ij}, \mathbf{w}_i\mathbf{h}_j^T\right) + R(\mathbf{W},\mathbf{H}) \qquad (1)$$

where R is a regularization term (often L_1 or L_2 norms), and E measures the error that the latent model makes on the observed ratings. Most often, E is the squared error.

Using a matrix factorization approach for predicting unknown ratings relies on the hypothesis that a model accurately predicting observed rating generalizes well to unknown ratings. In the following section, we argue that the former hypothesis is easily challenged.

3. INTERPRETING MISSING DATA

LaunchCast is Yahoo's former music service, where users could, among other things, rate songs. In a survey of 2006, users were asked to rate randomly selected songs [18]. The distribution of ratings of random songs was then compared to the distribution of voluntary ratings. The experiment concluded that the distribution of the ratings for random songs was strongly dominated by low ratings, while the voluntary ratings had a distribution close to uniform [18].

Intuitively, a simple process could explain the results: users chose to rate songs they listen to, and listen to music they expect to like, while avoiding genres they dislike. Therefore, most of the songs that would get a bad rating are not voluntary rated by the users. Since people rarely listen to random songs, or rarely watch random movies, we should expect to observe in many areas a difference between the distribution of ratings for random items and the corresponding distribution for the items selected by the users. This observation has a direct impact on the presumed capacity of matrix factorization to generalize a model based on observed ratings to unknown ratings.

Building on the *not missing at random assumption* [18, 28], we propose to incorporate in the optimization problem stated in Equation 1 a prior about the unknown ratings, in order to limit the bias caused by learning on observed ratings:

$$\begin{aligned}\operatorname*{arg\,min}_{\mathbf{W},\mathbf{H}} &\sum_{i,j|r_{ij}\in\mathcal{R}} E\left(r_{ij}, \mathbf{w}_i\mathbf{h}_j^T\right) \\ &+ \alpha \sum_{i,j|r_{ij}\notin\mathcal{R}} E\left(\hat{r}_0, \mathbf{w}_i\mathbf{h}_j^T\right) + R(\mathbf{W},\mathbf{H})\end{aligned} \qquad (2)$$

The objective function (Equation 2) has now two parts (besides the regularization): the first part fits the model to the observed ratings, and the second part drives the model toward a prior estimate $\hat{r}_0$ on the unknown ratings. In absence of further knowledge about a specific dataset, we suggest to use $\hat{r}_0 = 0$, the worst rating, as a prior estimate. The coefficient α allows to balance the influence of the unknown ratings, and the original formulation is obtained with $\alpha = 0$. We expect α to be small to deal with the problem of class imbalance. Indeed, in real-life applications the number of known ratings $|\mathcal{R}|$ is very small in comparison to the number of unknown ratings ($nm - |\mathcal{R}|$), and if α is close to 1, or larger, the second term of the objective function will completely dominate the other parts and drive all the users' and items' features to zero. It is therefore important to find a right balance between the influence of the few known ratings and of the many unknown ones.

In order to have a more intuitive feeling of the influence of both parts of the objective function we introduce $\rho = \alpha(nm - |\mathcal{R}|)/|\mathcal{R}|$, which can be interpreted as an influence ratio between unknown and known ratings. If $\rho = 0$, the unknown ratings are ignored, if $\rho = 1$, both the known ratings and the unknown ratings have the same global influence on the objective function, if $\rho = 2$, the unknown ratings are twice as important as the known ratings, etc.

A more involved model could assume an adaptive ρ per user or item, which could lead to additional, albeit small, gains. However, this implies more parameters to tune, more cumbersome equations to explain and an involved process to prove that the complexity of the method remains the same. Due to limited space, instead, we provide a general demonstration of the method and leave the adaptive model for future work.

4. LOSS FUNCTIONS

An obvious difficulty raised by the new optimization problem introduced earlier is the apparent increase in complexity. The naive complexity of evaluating this objective function is $O(nmk)$, while it is $O(|\mathcal{R}|k)$ for classical matrix factorization approaches (Equation 1). In this section, we demonstrate how it is possible to use our new model without the naive additional cost, and present a way to perform fast updates to incorporate new ratings in the model.

To this end, we show the applicability of our method when E is the squared loss in Section 4.1 and the absolute loss in Section 4.2. For the sake of demonstration, we also discuss its applicability on the generalized Kullback-Liebler divergence in Section 4.3. Finally, in Section 4.4 we outline how the method can be enforced in a static setting, and a dynamic setting with continuous updates of new ratings, items and users.

4.1 Squared Loss

By considering E as the squared loss, and R as the L_1 regularization, the optimization problem becomes:

$$\begin{aligned}\underset{\mathbf{W},\mathbf{H}}{\arg\min} \sum_{i,j|r_{ij}\in\mathcal{R}} & \left(r_{ij} - \mathbf{w}_i\mathbf{h}_j^T\right)^2 \\ & + \alpha \sum_{i,j|r_{ij}\notin\mathcal{R}} \left(\mathbf{w}_i\mathbf{h}_j^T\right)^2 \\ & + \lambda\left(\sum_{i=1}^{n} ||\mathbf{w}_i||_1 + \sum_{j=1}^{m} ||\mathbf{h}_j||_1\right)\end{aligned} \tag{3}$$

For the sake of simplicity, let us forget about the regularization term of the objective function for now (adding it to the following development is trivial), and let us call $L(\mathbf{W},\mathbf{H},\mathcal{R})$ the objective function without regularization. We want to be able to update the features of one user or of one item in a time independent of the size of the dataset $(n, m, |\mathcal{R}|)$. In the remainder, we show that it is possible to compute $\partial L/\partial\mathbf{w}_i$ and $\partial L/\partial\mathbf{h}_j$ with a complexity linear in the number of ratings provided by user i ($|\mathcal{R}_{i\bullet}|$) or given to item j ($|\mathcal{R}_{\bullet j}|$), respectively. On most datasets, and for most users and items, we have $|\mathcal{R}_{i\bullet}| \ll m$ and $|\mathcal{R}_{\bullet j}| \ll n$, and, therefore computing the gradient for one user or one item is fast.

First, let us separate L in n blocks l_i^w that contain only the terms of L depending on $\mathbf{w}_i$:

$$l_i^w = \sum_{j|r_{ij}\in\mathcal{R}} \left(r_{ij} - \mathbf{w}_i\mathbf{h}_j^T\right)^2 + \alpha \sum_{j|r_{ij}\notin\mathcal{R}} \left(\mathbf{w}_i\mathbf{h}_j^T\right)^2 \tag{4}$$

Notice that we have:

$$L = \sum_i l_i^w \quad \text{and} \quad \frac{\partial L}{\partial \mathbf{w}_i} = \frac{\partial l_i^w}{\partial \mathbf{w}_i}$$

If we adopt a naive computation, the second term of Equation (4) is more time expensive because most items are not rated by the user. However, the sum on unknown ratings (i.e. $\sum_{j|r_{ij}\notin\mathcal{R}}$), can be formulated as the difference between the sum on all items (i.e. $\sum_{j=1}^m$) and the sum on rated items only (i.e. $\sum_{j|r_{ij}\in\mathcal{R}}$) . By so doing, the sum on unknown ratings disappears from the computations:

$$\begin{aligned}\sum_{j|r_{ij}\notin\mathcal{R}} \left(\mathbf{w}_i\mathbf{h}_j^T\right)^2 &= \sum_{j=1}^{m} \left(\mathbf{w}_i\mathbf{h}_j^T\right)^2 - \sum_{j|r_{ij}\in\mathcal{R}} \left(\mathbf{w}_i\mathbf{h}_j^T\right)^2 \quad (5) \\ &= \mathbf{w}_i\mathbf{S^h}\mathbf{w}_i^T - \sum_{j|r_{ij}\in\mathcal{R}} \left(\mathbf{w}_i\mathbf{h}_j^T\right)^2 \quad (6)\end{aligned}$$

where we have posed $\mathbf{S^h} = \sum_j \mathbf{h}_j^T\mathbf{h}_j$, a $k \times k$ matrix independent of i (i.e. it is the same matrix for all l_i^w). Assuming that $\mathbf{S^h}$ is known, we can now compute l_i^w and $\partial L/\partial\mathbf{w}_i$ with a complexity of $O(|\mathcal{R}_{i\bullet}|k + k^2)$. From Equations 4 and 6, we obtain:

$$l_i^w = \sum_{j|r_{ij}\in\mathcal{R}} \left[(r_{ij} - \mathbf{w}_i\mathbf{h}_j^T)^2 - \alpha(\mathbf{w}_i\mathbf{h}_j^T)^2\right] + \alpha\mathbf{w}_i\mathbf{S^h}\mathbf{w}_i^T \tag{7}$$

We can easily derive:

$$\frac{\partial L}{\partial \mathbf{w}_i} = -2\sum_{j|r_{ij}\in\mathcal{R}} \left[r_{ij} - (1-\alpha)\mathbf{w}_i\mathbf{h}_j^T\right]\mathbf{h}_j + 2\alpha\mathbf{w}_i\mathbf{S^h} \tag{8}$$

Symmetrically, if $\mathbf{S^w} = \sum_i \mathbf{w}_i^T\mathbf{w}_i$, we have:

$$l_j^h = \sum_{i|r_{ij}\in\mathcal{R}} \left[(r_{ij} - \mathbf{w}_i\mathbf{h}_j^T)^2 - \alpha(\mathbf{w}_i\mathbf{h}_j^T)^2\right] + \alpha\mathbf{h}_j\mathbf{S^w}\mathbf{h}_j^T \tag{9}$$

and:

$$\frac{\partial L}{\partial \mathbf{h}_j} = -2\sum_{i|r_{ij}\in\mathcal{R}} \left[r_{ij} - (1-\alpha)\mathbf{w}_i\mathbf{h}_j^T\right]\mathbf{w}_i + 2\alpha\mathbf{h}_j\mathbf{S^w} \tag{10}$$

Assuming that $\mathbf{S^w}$ is known, the complexity of computing l_j^h or $\partial L/\partial\mathbf{h}_j$ is now $O(|\mathcal{R}_{\bullet j}|k + k^2)$, and the complexity of computing it for every $j \in \{1, \ldots, m\}$ is $O(|\mathcal{R}|k + k^2)$.

4.2 Absolute Loss

A similar development can be done when the squared loss is replaced by the absolute loss. With the absolute loss, L becomes:

$$L = \sum_{i,j|r_{ij}\in\mathcal{R}} \left|r_{ij} - \mathbf{w}_i\mathbf{h}_j^T\right| + \alpha\sum_{i,j|r_{ij}\notin\mathcal{R}} \left|\mathbf{w}_i\mathbf{h}_j^T\right|$$

As with the squared loss, we divide L into l_i^w and l_j^h.

$$l_i^w = \sum_{j|r_{ij}\in\mathcal{R}} \left|r_{ij} - \mathbf{w}_i\mathbf{h}_j^T\right| + \alpha\sum_{j|r_{ij}\notin\mathcal{R}} \left|\mathbf{w}_i\mathbf{h}_j^T\right|$$

As with the squared loss, we will change the expression of l_i^w to remove the sum over all unknown ratings, but in this case we have to impose non-negativity of the features to go further. If $\mathbf{W},\mathbf{H} \geq 0$, we have $\left|\mathbf{w}_i\mathbf{h}_j^T\right| = \mathbf{w}_i\mathbf{h}_j^T$, and therefore:

$$\begin{aligned}\sum_{j|r_{ij}\notin\mathcal{R}} \left|\mathbf{w}_i\mathbf{h}_j^T\right| &= \sum_{j=1}^{m}\mathbf{w}_i\mathbf{h}_j^T - \sum_{j|r_{ij}\in\mathcal{R}}\mathbf{w}_i\mathbf{h}_j^T \quad (11) \\ &= \mathbf{w}_i\left(\sum_{j=1}^{m}\mathbf{h}_j^T\right) - \sum_{j|r_{ij}\in\mathcal{R}}\mathbf{w}_i\mathbf{h}_j^T \quad (12)\end{aligned}$$

Here, instead of $\mathbf{S^w}$ and $\mathbf{S^h}$, we will define $\mathbf{s}_w = \sum_{i=1}^n \mathbf{w}_i$ and $\mathbf{s}_h = \sum_{j=1}^m \mathbf{h}_j$. We can now express l_i^w and $\partial L/\partial\mathbf{w}_i$ efficiently:

$$l_i^w = \sum_{j|r_{ij}\in\mathcal{R}} \left(\left|r_{ij} - \mathbf{w}_i\mathbf{h}_j^T\right| - \alpha\mathbf{w}_i\mathbf{h}_j^T\right) + \alpha\mathbf{w}_i\mathbf{s}_h^T \tag{13}$$

so that:

$$\frac{\partial L}{\partial \mathbf{w}_i} = \sum_{j|r_{ij}\in\mathcal{R}} \left(\operatorname{sign}\left(\mathbf{w}_i\mathbf{h}_j^T - r_{ij}\right) - \alpha\right)\mathbf{h}_j + \alpha\mathbf{s}_h^T \tag{14}$$

where $\text{sign}(x) = x/|x|$ if $x \neq 0$, and equals 0 otherwise. Assuming $\mathbf{s}_h$ is known, the complexity of computing l_i^w or $\partial L/\partial \mathbf{w}_i$ is now $O(|\mathcal{R}_{i\bullet}|k)$. The corresponding expression of l_j^h and $\partial L/\partial \mathbf{h}_j$ is trivial, and the complexity to compute them is $O(|\mathcal{R}_{\bullet j}|k)$.

4.3 Generalized Kullback-Leibler Divergence

For the sake of demonstration on other common loss functions in matrix factorization, we show here the applicability of the sparsity trick on the generalized Kullback-Leibler divergence (GKL) [14, 16]. We do not elaborate further on this function in the rest of the paper.

The generalized Kullback-Leibler divergence is defined as follows:

$$D(r_{ij}||\mathbf{w}_i\mathbf{h}_j^T) = r_{ij}\log(\frac{r_{ij}}{\mathbf{w}_i\mathbf{h}_j^T}) - r_{ij} + \mathbf{w}_i\mathbf{h}_j^T \quad (15)$$

The GKL is not defined when $r_{ij} = 0$. In the following we extend the GKL by using its limit value:

$$D(0||\mathbf{w}_i\mathbf{h}_j^T) := \lim_{r\to 0} D(r||\mathbf{w}_i\mathbf{h}_j^T) = \mathbf{w}_i\mathbf{h}_j^T \quad (16)$$

Using Equation 15 and 16, L becomes:

$$L = \sum_{i,j|r_{ij}\in\mathcal{R}} \left(r_{ij}\log(\frac{r_{ij}}{\mathbf{w}_i\mathbf{h}_j^T}) - r_{ij} + \mathbf{w}_i\mathbf{h}_j^T \right) + \alpha \sum_{i,j|r_{ij}\notin\mathcal{R}} \mathbf{w}_i\mathbf{h}_j^T$$

We now follow the same development as with the other losses. We define l_i^w:

$$l_i^w = \sum_{j|r_{ij}\in\mathcal{R}} \left(r_{ij}\log(\frac{r_{ij}}{\mathbf{w}_i\mathbf{h}_j^T}) - r_{ij} + \mathbf{w}_i\mathbf{h}_j^T \right) + \alpha \sum_{j|r_{ij}\notin\mathcal{R}} \mathbf{w}_i\mathbf{h}_j^T$$

In the case of the GKL, the process to remove the sum on unknown ratings is the same as with the absolute loss, except that in absence of absolute value we do not have to impose non-negativity of the features:

$$\sum_{j|r_{ij}\notin\mathcal{R}} \mathbf{w}_i\mathbf{h}_j^T = \mathbf{w}_i\mathbf{s}_h^T - \sum_{j|r_{ij}\in\mathcal{R}} \mathbf{w}_i\mathbf{h}_j^T \quad (17)$$

This leads to:

$$l_i^w = \sum_{j|r_{ij}\in\mathcal{R}} \left(r_{ij}\log(\frac{r_{ij}}{\mathbf{w}_i\mathbf{h}_j^T}) - r_{ij} + (1-\alpha)\mathbf{w}_i\mathbf{h}_j^T \right) + \alpha\mathbf{w}_i\mathbf{s}_h^T \quad (18)$$

Now we can easily derive $\partial L/\partial \mathbf{w}_i$:

$$\frac{\partial L}{\partial \mathbf{w}_i} = \sum_{j|r_{ij}\in\mathcal{R}} \left(-\frac{r_{ij}}{\mathbf{w}_i\mathbf{h}_j^T} + (1-\alpha) \right) \mathbf{h}_j + \alpha\mathbf{s}_h^T \quad (19)$$

The corresponding expression of l_j^h and $\partial L/\partial \mathbf{h}_j$ is obtained symmetrically. As with the absolute loss, the complexity of computing l_i^w or $\partial L/\partial \mathbf{w}_i$ is now $O(|\mathcal{R}_{i\bullet}|k)$ (it is $O(|\mathcal{R}_{\bullet j}|k)$ for l_j^h and $\partial L/\partial \mathbf{h}_j$).

Algorithm 1 Randomized block coordinate descent

```
Require:
   – The ratings $\mathcal{R}$.
   – The number of features $k$.
 1: Initialize $\mathbf{W}$ and $\mathbf{H}$.
 2: Compute $\mathbf{S^w}$ and $\mathbf{S^h}$ ($\mathbf{s}_w$ and $\mathbf{s}_h$)
 3: while not converged do
 4:   for all user and item, traversed in a random order do
 5:     In the case of a user ($i$) do
 6:       Perform a gradient step on $\mathbf{w}_i$ using line search
 7:       Update $\mathbf{S^w}$ ($\mathbf{s}_w$)
 8:     In the case of an item ($j$) do
 9:       Perform a gradient step on $\mathbf{h}_j$ using line search
10:       Update $\mathbf{S^h}$ ($\mathbf{s}_h$)
11:   end for
12: end while
```

4.4 Static and Dynamic Factorization

We introduce an online algorithm to learn the latent factors from the input data in a static setting, and show how it can accommodate updates in a dynamic setting.

4.4.1 *Static Factorization*

In order to factorize a whole new set of data we propose to use a randomized block coordinate descent [26]. At each iteration, all the users and items are traversed in a random order. For each of them a gradient step is performed on their features while keeping the other features constant.

We can use a line search [2] to determine the size of the gradient step because the variation of L for a modification of $\mathbf{w}_i$ is entirely determined by l_i^w and can therefore be computed efficiently. Line search allows to avoid the burden of tuning the step size, proper to stochastic gradient descent (SGD) methods [7]. Moreover, using line search guarantees the convergence of the value of the objective function. Indeed, each gradient step decreases (or rather *cannot increase*) the objective function which is bounded from below. This implies that the variation of the objective function converges to zero.

The complete procedure for the factorization through randomized block coordinate descent is summarized in Algorithm 1.

Complexity. In the case of the squared loss, the computation of fast gradient step relies on knowing $\mathbf{S^w}$ and $\mathbf{S^h}$. Their initial value is computed in $O(nk^2)$ and $O(mk^2)$, respectively, and the cost of updating them after each gradient step is $O(k^2)$. The total complexity of an iteration of our algorithm is therefore $O(|\mathcal{R}|k + (n+m)k^2)$, as good as the best factorization methods that do not use priors on unknown ratings [9].

In the case of the absolute loss and generalized Kullback-Liebler divergence, the computation uses $\mathbf{s}_w$ and $\mathbf{s}_h$. Their initial value is computed in $O(nk)$ and $O(mk)$, while the cost of updating them is $O(k)$. The total complexity of one iteration then becomes $O(|\mathcal{R}|k + (n+m)k)$, which is lower than the squared loss' complexity. However, this usually comes at a cost on the performance of the results, as we will show in the experiments in Section 5.

4.4.2 *Fast Updates*

The expressions of l_i^w, l_j^h, and their gradients (Equations

Algorithm 2 Update algorithm

```
Require:
    – The new rating $r_{ij}$.
    – The ratings of user $i$ ($\mathcal{R}_{i\bullet}$) and of item $j$ ($\mathcal{R}_{\bullet j}$).
1: If $\mathbf{w}_i$ ($\mathbf{h}_j$) does not exist, initialize it (for example by
   setting a random feature to 1).
2: Add $r_{ij}$ to $\mathcal{R}_{i\bullet}$ and $\mathcal{R}_{\bullet j}$.
3: while not converged do
4:     Perform a gradient step on $\mathbf{w}_i$ using line search
5:     Update $\mathbf{S}^{\mathbf{w}}$ ($\mathbf{s}_w$)
6:     Perform a gradient step on $\mathbf{h}_j$ using line search
7:     Update $\mathbf{S}^{\mathbf{h}}$ ($\mathbf{s}_h$)
8: end while
```

(7), (8), (9) and (10)) allow us to compute the latent representations of one user or one item in a time independent of the number of users and items in the system. We can use that ability to design a simple algorithm for updating an existing factorization when a new rating is added to $\mathcal{R}$: If user i rates item j, we iteratively perform gradient steps for $\mathbf{w}_i$ and $\mathbf{h}_j$, keeping all other features constant. This relies on the assumption that a new rating will only affect significantly the user and item that are directly concerned with it. Although this assumption can be disputed, we will show in our experiments (Section 5.3) that our update algorithm produces recommendations of stable quality, indicating that limiting our updates to the directly affected users and items does not degrade the factorization over time.

When ratings are produced by new users or given to new items, a new set of features for that user or item is created before performing the local optimization. Various initialization strategies could be explored here. However, as we show in our experimental results, assigning a random value to one of the features and setting the others to zero performs well in practice. The update procedure is summarized in Algorithm 2.

Complexity. As mentioned earlier, our update algorithm is independent of the number of users or items in the system, making it suitable for very large datasets. Each iteration of the update algorithm is composed of two gradient steps (one on the user's features, and one on the item's features). In particular, the complexity of one iteration is $O((|\mathcal{R}_{i\bullet}| + |\mathcal{R}_{\bullet j}|)k + k^2)$ for the squared loss, and only $O((|\mathcal{R}_{i\bullet}| + |\mathcal{R}_{\bullet j}|)k)$ for the absolute loss and the GKL. This difference in complexity becomes significant when k is large with regards to the average number of ratings per user and per item.

Updates based on classic SGD methods have an even smaller complexity ($O(k)$)), but we will show in Section 5 that our method produces recommendations of much higher quality, while still being able to satisfy applications requiring low-latency updates.

5. EXPERIMENTS

We perform several experiments to demonstrate the following key points:

- Using priors on the unknown values leads to overall improved quality of ranking, in a static or dynamic setting.
- The quality does not degrade with time, i.e., as more updates are added, the model does not lose accuracy.

Table 1: Characteristics of the datasets used.

Dataset	Users	Items	Ratings
Movielens	6,040	3,706	1,000,209
FineFoods	256,059	74,258	568,454
AmazonMovies	889,176	253,059	7,831,442

- Our methods can outperform traditional techniques on various large datasets.

In our experiments, we test the performance of the squared loss (SL) and the absolute loss (AL) with and without prior on unknown values. In Section 5.1 we describe our experimental setup: the benchmarked datasets used, the performance metrics recorded and how we tune the various parameters of the models tested during the experiments. Then, in Sections 5.2 and 5.3 we describe the results of our methods in a *static* and *dynamic* learning setting, respectively, and how they compare with state-of-the-art methods. In Section 5.4 we illustrate the importance of fast updates by studying the impact of having a delay between the arrival of new ratings and the update of the factorization. In Section 5.5 we investigate in depth the influence of parameter values selected in the two loss functions (squared and absolute loss). Finally, details allowing the reproducibility of the results are given in Section 5.6.

5.1 Experimental Setup

Here we briefly describe the experimental setup used for the static and dynamic learning and how the parameters of the different methods are tuned.

5.1.1 Datasets

During the experiments, we use three datasets with distinct features. Table 1 summarizes the characteristics of these datasets which provide different challenges to the recommendation task:

- Movielens: This is the well-known movie ratings dataset produced by the Grouplens project. We use the version containing 1 million ratings, with at least 20 ratings for each user.
- FineFoods: This is a collection of ratings about food products extracted from the Amazon comments [19]. The dataset is much sparser than Movielens, with most users having only a handful of ratings, making it a very hard dataset for the recommendation task.
- AmazonMovies: This is a larger collection of ratings extracted from the movie section of Amazon [19]. This dataset is also sparser than Movielens, although not as sparse as FineFoods.

5.1.2 Evaluation Metrics

We measure two standards metrics used in ranking evaluation: (1) Normalized Discounted Cumulative Gain (NDCG) [1, 15] and, (2) area under ROC curve (AUC) [20, 23, 27].

NDCG rewards methods that rank items with the highest observed rating at the top of the ranking. The *discounted* aspect of NDCG comes from the fact that relevant items ranked at low positions of the ranking contribute less to the final score than relevant items at top positions.

In the static experiments, we also report the NDCG computed on the rated items only. This metric does not con-

Table 2: List of the parameters of each method, and set of values tested during the parameters tuning of the squared loss (SL) and absolute loss (AL), with and without prior on unknown values, as well as the multiplicative update algorithm (Mult-NMF), Alternating Least Square (ALS-UV), and Vowpal Wabbit (VW). k: number of features, λ: regularization coefficient, ρ: unknown/known influence ratio, γ: learning rate.

Method	Parameter	Tested Values
SL/AL with prior	k	5, 10, 20, 50, 100, 200
	λ	0, 0.01, 0.1, 1, 10
	ρ	0.3, 0.7, 1, 2
SL/AL without prior	k	5, 10, 20, 50, 100, 200
	λ	0, 0.01, 0.1, 1, 10
ALS-UV	k	20, 50, 100, 200, 500
	λ	0, 0.001, 0.01, 0.05, 0.1
Mult-NMF	k	20, 50, 100, 200, 500
VW	k	20, 50, 100, 200, 500
	λ	0 1e-5 1e-2
	γ	0.01, 0.02, 0.05, 0.1, 0.2

sider the real world case scenario which consists of ranking all items since we do not know in advance which item will be rated or not. Intuitively, by biasing our objective through the introduction of priors on unknown rating we may loose performance when ranking rated items only, while performing better when considering all the items.

We use AUC to evaluate the ability of the different methods to predict which items are going to be rated. AUC measures whether the items whose ratings were held out during learning are ranked higher than unrated items. The perfect ranking has an AUC of 1, while the average AUC for random ranking is 0.5.

5.1.3 Parameter Tuning

Table 2 shows the parameters of the various models and the values tested during parameter tuning. For each test, the parameters' values producing the best ranking on the validation sets (measured by NDCG for the static test and AUC for the dynamic test) were selected to be used. See Sections 5.2 and 5.3 for the description of the validation sets.

5.2 Static Learning

Research question. In a static mode, we test to which extent using a prior on unknown ratings improves the ranking of items when recommended to users.

Process followed. The test set was constructed by randomly selecting 1000 users, and splitting the ratings of those users in half, keeping the first 50% of the ratings in the training set, according to timestamp, and the last 50% in the test set. The same process (selecting 1000 users and splitting their ratings) was then applied three times on the training set in order to create three training/validation pairs of sets. On each run, the parameters producing on average the best NDCG over the three validation sets were then used to factorize the full training set, and evaluated on the test set.

Table 3: Comparison of our introduced algorithm in static learning on the datasets Movielens, FineFoods and AmazonMovies. Values in bold hold for the method that outperform all the other methods according to a Mann-Withney U test with a confidence level of 1%. Average values are shown alongside their standard deviation over 10 runs.

	Movielens		
	NDCG-RI	NDCG	AUC
SL w/ prior	0.885 ± 0.0014	**0.5046 ± 0.0013**	**0.8695 ± 0.0012**
SL w/o prior	0.886 ± 0.0015	0.3597 ± 0.0012	0.6548 ± 0.0014
AL w/ prior	0.8683 ± 0.0030	0.4452 ± 0.0009	0.8134 ± 0.0011
AL w/o prior	0.8794 ± 0.0031	0.3801 ± 0.0106	0.6927 ± 0.0322
Mult-NMF	0.8433 ± 0.0007	0.3758 ± 0.0006	0.7011 ± 0.0009
ALS-UV	0.8332 ± 0.0014	0.3292 ± 0.0004	0.5839 ± 0.0005
	FineFoods		
	NDCG-RI	NDCG	AUC
SL w/ prior	0.887 ± 0.0016	**0.1237 ± 0.0039**	**0.8452 ± 0.0074**
SL w/o prior	0.888 ± 0.0158	0.1023 ± 0.0022	0.8314 ± 0.0058
AL w/ prior	0.8722 ± 0.0142	0.1026 ± 0.0030	0.8412 ± 0.0047
AL w/o prior	0.8730 ± 0.0260	0.0923 ± 0.0008	0.7294 ± 0.0143
Mult-NMF	0.8476 ± 0.0084	0.0830 ± 0.0008	0.3403 ± 0.0052
ALS-UV	0.8653 ± 0.025	0.0873 ± 0.0009	0.5485 ± 0.0114
	AmazonMovies		
	NDCG-RI	NDCG	AUC
SL w/ prior	0.8992 ± 0.0101	**0.1887 ± 0.0088**	**0.9276 ± 0.0031**
SL w/o prior	0.9035 ± 0.0089	0.1103 ± 0.0008	0.8656 ± 0.0033
AL w/ prior	0.8804 ± 0.0077	0.1348 ± 0.0035	0.8634 ± 0.0045
AL w/o prior	0.8854 ± 0.0102	0.1002 ± 0.0012	0.7625 ± 0.0051
Mult-NMF	0.8498 ± 0.0026	0.0959 ± 0.0004	0.6330 ± 0.0040
ALS-UV	0.8658 ± 0.0034	0.0906 ± 0.0003	0.6601 ± 0.0061

Baseline. We report the results achieved by two traditional well-known algorithms: UV matrix decomposition solved with Alternating Least Square (ALS-UV) [30], and non-negative matrix factorization with the multiplicative update algorithm (Mult-NMF) [14]. Both ALS-UV and Mult-NMF use the squared loss.

Results. The results, averaged over 10 runs, are shown in Table 3. We can observe that for both the squared loss and the absolute loss, and on all datasets, by adding a prior on the unknown ratings we improve significantly the rankings of the items recommended to users over rankings obtained by the same techniques when they do not put a prior on the unknown ratings (and also over rankings obtained by state-of-the-art approaches ALS-UV and Mult-NMF). In particular, our implementation of the squared loss with prior outperforms all other methods, as confirmed by a Mann-Withney U test with a confidence level of 1%.

On the Movielens dataset, the results of Mult-NMF and ALS-UV are, as expected, similar to the ones of our implementation of the squared loss without prior. Indeed, those methods optimize the same objective function, and differ only by their algorithm. Interestingly, on the sparser FineFoods and AmazonMovies dataset, our randomized block coordinate gradient descend method outperforms Mult-NMF and ALS-UV, even without prior on the unknown ratings.

Furthermore, in the three tested data sets, when only considering the rated items, the loss in ranking performance is never significant (see NDCG-RI with and without prior in Table 3). In other words, while improving on the global ranking, the performance does not deteriorate when considering only the subset of rated items.

5.3 Dynamic Learning

Research question. In this section, we target two research questions:

1. We test whether our update algorithm is able to sustain stable quality of recommendations over time;

2. We test to which extent using priors on unknown ratings improves the ranking of recommended items when the model is updated each time a new instance is encountered. By so doing, the system is evaluated on more realistic scenarios where the cases of cold items and users are considered as well.

Process followed. We order the ratings by timestamps and separate the ratings in three blocks: first the training, then the validation and finally the test block (see Table 4 for the size of each block in the different datasets).

Table 4: Number of ratings in each block for the dynamic learning.

Dataset	Training	Validation	Test
Movielens	500,000	100,000	100,000
FineFoods	400,000	100,000	68,000
AmazonMovies	5,000,000	1,000,000	1,000,000

The evaluation is performed as follows: an initial model is built based on all the ratings present before the test block, then, for each rating of the test block, two steps are performed in the following order:

1. The current model is evaluated by computing the AUC over the new (user, item) pair. Notice that in this case, computing the AUC means computing the proportion of items not yet rated by the user that the model ranks lower than the item that was just rated. An AUC of 1 means that the new item was the top recommendation of the method for that user.

2. The model is updated using the new rating. It is worth noticing that the rating may concern a new user or item, and, therefore, features for that new user/item have to be added to the model.

Parameter tuning is done as described above, but starting at the beginning of the validation block and ending before the test block. The values of parameters tested are the same as in the static test (see Table 2).

Baseline. We compared our methods to Vowpal Wabbit (VW). VW is a machine learning framework solving different optimization problems for classification and ranking, by implementing a carefully optimized, stochastic gradient descent (SGD) using feature hashing [29] and adaptive gradient steps [7]. We are using the VW's implementation of low-rank interactions[1] based on factorization machines [22].

Results. Figure 1 shows how the average AUC evolves as new ratings enter the system. We first observe that the quality of the results does not decrease over time, indicating that our update algorithm can work for long periods of time without propagating or amplifying errors. As in the static experiment, we confirm that adding a prior on unknown ratings improves the quality of the ranking and, again, this is maintained across time. Moreover, the SGD approach of VW is outranked in each dataset by our approach with prior.

[1]https://github.com/JohnLangford/vowpal_wabbit/tree/master/demo/movielens

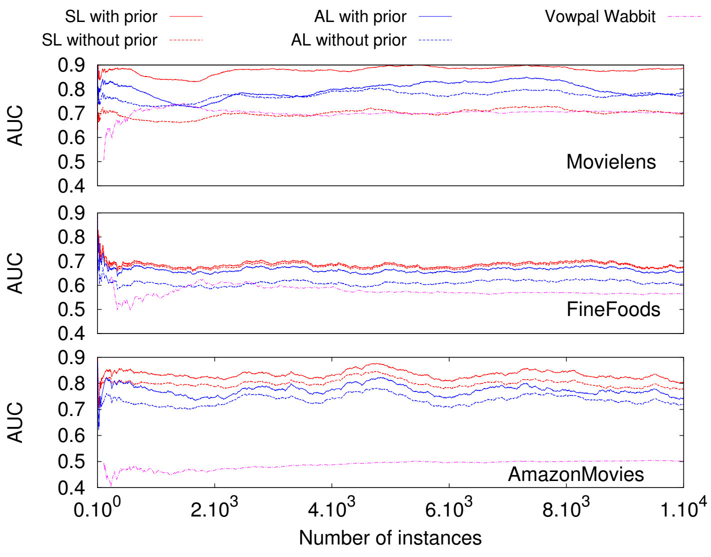

Figure 1: Performance comparison with respect to average AUC for the various methods tested in dynamic learning on Movielens, FineFoods and AmazonMovies. Results are averaged over 20 runs.

5.4 Impact of delayed updates

Research Question. We test the performance of delayed models produced by our methods in delivering recommendations to users.

Process followed. In order to address this question, we simulate a recommender system that is not able to incorporate new ratings in the model as soon as they enter the system. To do so, we modify the process of dynamic learning presented in Section 5.3 to impose a delay between the arrival of a new rating and the update of the factorization. More precisely, after the i^{th} rating is given by a user, the model is updated up to the $(i-d)^{\text{th}}$ rating (d being the arbitrary delay). This way, the model is always d ratings behind the last one arrived (the ratings are sorted by real time of arrival). In real applications, the delay would probably vary, depending on the level of activity of the users. However, this experiment gives a first impression of the impact of delays on the recommendation task.

Results. Figure 2 shows the impact of a delay on the average AUC of the squared loss and absolute loss with prior for a dense dataset like Movielens and a sparse dataset like FineFoods. We observe that even a small delay can affect the quality of the recommendation, depending on the characteristics of the data. For Movielens, if the model is behind by $5-10$ ratings, the average AUC drops by 3%, and it goes down by about 14% when the model is behind by 1000 ratings, and this applies to both loss functions. On the other hand, for the much sparser FineFoods, the effect is more apparent. With only $5-10$ ratings behind, the model's AUC already drops by 10%.

To show the effect of fast updates on weakly-engaged (or cold) users, we also report the impact of delays on those users for both Movielens and FineFoods with the squared loss which performs best (Figure 2, Cold Users). We define such users as the ones that rated at most two items. As

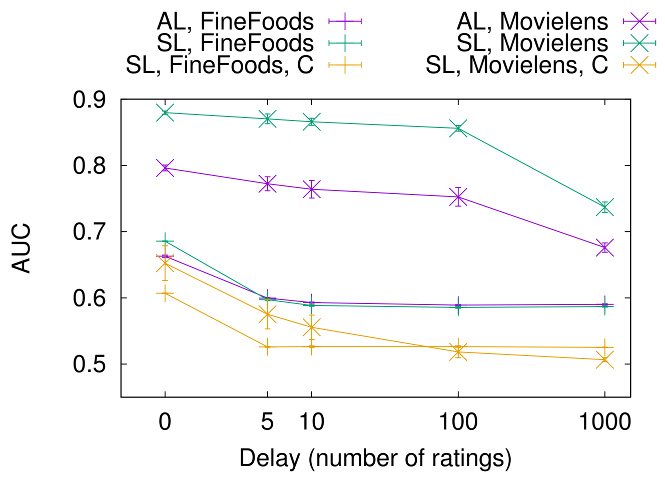

Figure 2: Average AUC of the squared loss and absolute loss with prior on Movielens and FineFoods for various delays d, imposed as a number of ratings that the model is behind the current rating.

hypothesized, the cost induced by delayed predictions (for five ratings delayed) is higher for cold users. We observe a relative drop in AUC of 11.8% and 13.4% for weakly-engaged users on Movielens and FineFoods, respectively, while when considering all the users, the relative drop is 1.1% and 12.9%, respectively.

In such sparse scenarios, cold users perform only a handful of actions before deciding to abandon the site or not. Therefore, it is important to consider cold users in the model as soon as they arrive, to keep them engaged by fast, efficient and good recommendations.

5.5 Parameter Analysis

Research Question. We test to which extent the number of features (k), the weight of the prior (ρ) and the regularization coefficient (λ) affect the AUC and the runtime per update on our loss functions.

Figure 3 shows the results of this investigation for different values of these parameters, for both squared loss and absolute loss and on each dataset. The results are obtained using the dynamic learning process.

Number of features. Concerning the quality of ranking (AUC), we observe the usual overfitting/under-fitting trade-off (Figure 3(a)). The optimal number of features depends on the dataset as well as on the loss function used, suggesting that a careful tuning of that parameter is always needed.

In some cases, speed constraints will force the use a sub-optimal number of features. Indeed, the update runtime heavily depends on the number of features. Figure 3(d) suggests a linear relationship between runtime and number of features. For both losses there is indeed a linear role of the number of features in the theoretical complexity (Section 4.4). Notice, however, that the theoretical complexity of the squared loss also has a quadratic term that becomes dominant for large number of features (with regards to the number of ratings per users). Also note that while the squared loss produces better AUC, the absolute loss is able to sustain higher update rates, and can therefore be the loss of choice when speed is the first criterion.

Regularization coefficient. The influence of λ seems rather limited, except for high values that cause both the AUC and the update runtime to drop (Figure3(b) and (e)). A small regularization is supposed to increase the quality of the model by reducing overfitting, but this effect is not visible here. The reason may be that the role of regularization is already taken by the prior on unknown ratings. Introducing the prior seems to have the side effect of making the regularization obsolete (or redundant). In fact, we confirm this with the results for $\lambda = 0$ which demonstrate no impact on the quality or runtime. Again, we see that setting a prior on unknown ratings increases the quality of recommendations without increasing the complexity of the solution. While it adds a term and a parameter to the objective function, it allows to remove one and its associated parameters.

Unknown/known influence ratio. The ratio ρ influences the performance of the squared loss algorithm in the following way: the AUC increases when a prior on unknown values is added ($\rho > 0$), but the exact value of ρ has little influence (in the observed range) (Figure 3(c)). The absolute loss is more sensitive to the value of ρ, with the AUC decreasing when ρ becomes too large (on Movielens and AmazonMovies). However, in both cases, and on all datasets, giving the same weight to the known and unknown ratings ($\rho = 1$) offers a significant improvement over not using a prior, suggesting that $\rho = 1$ can be used as a first guideline, avoiding the burden of further parameter tuning.

The update runtime is also affected by ρ, decreasing when ρ increases (Figure 3(f)). The explanation can be that the prior on unknown ratings acts as a regularizer, driving features towards 0, and in doing so speeding up the convergence.

Runtime. In general, our technique demonstrates low running time which is heavily dependent on the number of features used, and less on the regularization applied or the ratio of unknown over known values. These results demonstrate that our method can satisfy applications requiring low-latency updates.

5.6 Reproducibility of Results

The implementation of the algorithms introduced in Section 4 is available on Github:
`https://github.com/rdevooght/MF-with-prior-and-updates`.

For both ALS-UV and Mult-NMF we use the implementation of GraphChi, an open source tool for graph computation with impressive performance [13].

The code and documentation of Vowpal Wabbit is available on its Github page:
`https://github.com/JohnLangford/vowpal_wabbit/wiki`.

The Amazon datasets are available on the SNAP webpage:
`http://snap.stanford.edu/data/index.html`

The Movielens dataset is available on the Grouplens page:
`http://www.grouplens.org/datasets/movielens/`.

6. RELATED WORK

The problem of recommending products based on the actions and feedback from other users (rather than based on content similarity) is often called *collaborative filtering*, and dates back 20 years ago, with works such as Tapestry [10] and Grouplens [25]. The field is now dominated by methods based on matrix factorization, with algorithms such as ALS [30], the multiplicative update rule [14], and the

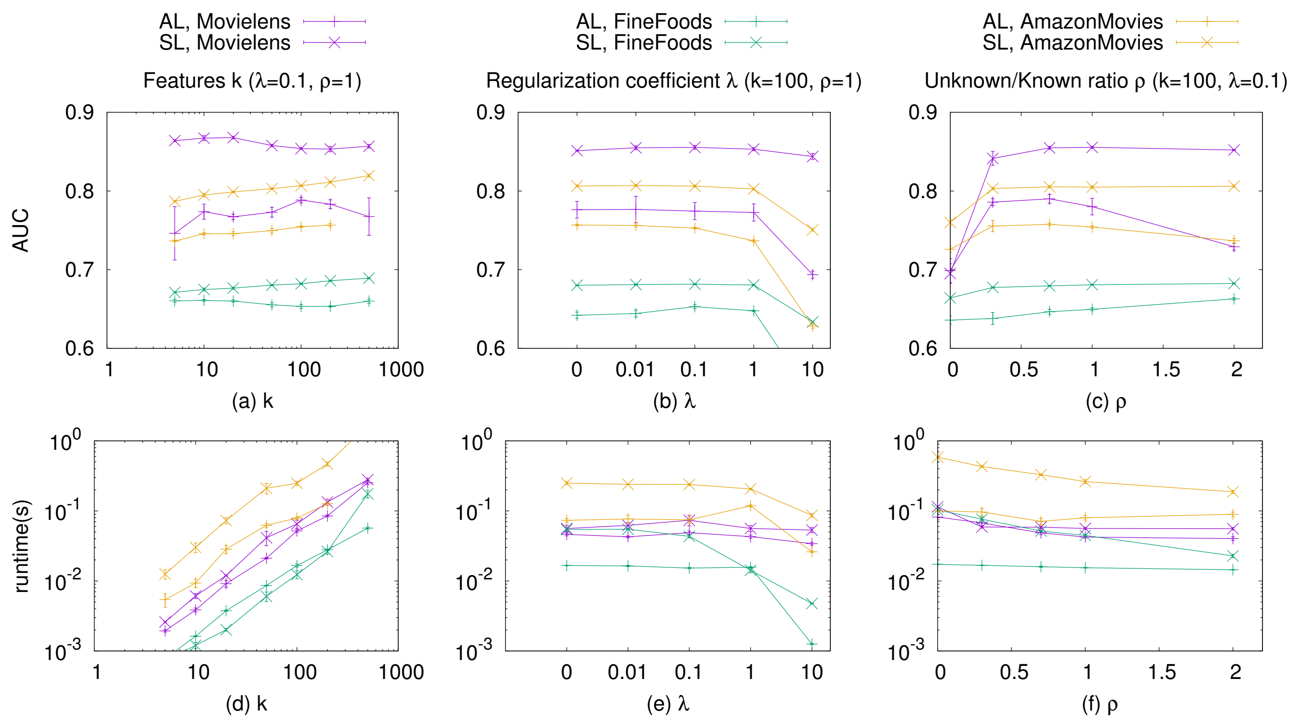

Figure 3: Influence of parameter values on the AUC and runtime for the models produced by the squared loss (SL) and absolute loss (AL). Y-error bars declare a standard deviation on the average value of each metric.

stochastic gradient descent method (SGD) [9, 12].

The missing at random assumption has yet to get the attention it deserves in collaborative filtering. Both [18] and [28] have validated the hypothesis of ratings missing not at random. Practical propositions for the interpretation of missing data can be found in the fields of one-class collaborative filtering and collaborative filtering based on implicit feedback, where the missing at random assumption is often obviously untenable [11, 20, 21]. [27] offers an interesting approach where missing ratings are considered as optimization variables; they use an EM algorithm to optimize in turn the factorization and the estimation of missing values. Unfortunately, that method has a high complexity, and the proposed approximations that work with large problems remove some of the method's appeal.

None of those works, however, consider the real world, dynamic scenario of continuously observing new ratings, users and items. Other works [8, 24] focus on the dynamic update of matrix factorization (mainly through the use of SGD), but those, on the other hand, implicitly rely on the missing at random assumption, and therefore suffer from lower accuracy in predictions. Other state-of-art methods for matrix factorization scale by relying on stochastic gradient computation [3, 4], while we rely on exact gradient approach. In this work, at the difference of what is mostly seen on scalable machine learning techniques nowadays [6], we base our approach on coordinate random block descent to compute exact gradient in order to deal with missing data of large scale matrices.

7. CONCLUSIONS

In this work we proposed a new, simple, and efficient, way to incorporate a prior on unknown ratings in several loss functions commonly used for matrix factorization. We experimentally demonstrated the importance of adding such a prior to solve the problem of collaborative ranking.

We also tackled the problem of updating the factorization when new users, items and ratings enter the system. We believe that this problem is central to real applications of recommendation systems, because new users constantly enter those systems and the factorization must be kept up to date to give them recommendations immediately after their first few interactions with the platform. We offer an update algorithm whose complexity is independent of the size of the data, making it a good approach for large datasets.

In the future, we would like to explore how our methods perform under real workloads of updates with variable arrival rates of ratings per user and item. Furthermore, we would like to test the performance of our methods in platforms built to analyze streams of data such as Storm, Twitter's Distributed Processing Engines platform.

8. ACKNOWLEDGEMENTS

R. Devooght is supported by the Belgian Fonds pour la Recherche dans l'Industrie et l'Agriculture (FRIA, 1.E041.14).

References

[1] S. Balakrishnan and S. Chopra. Collaborative ranking. In *Proc. of the 5th ACM WSDM*, pages 143–152, 2012.

[2] S. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge university press, 2009.

[3] W.-S. Chin, Y. Zhuang, Y.-C. Juan, and C.-J. Lin. A fast parallel stochastic gradient method for matrix factorization in shared memory systems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(1):2, 2015.

[4] W.-S. Chin, Y. Zhuang, Y.-C. Juan, and C.-J. Lin. A learning-rate schedule for stochastic gradient methods to matrix factorization. In *Advances in Knowledge Discovery and Data Mining*, pages 442–455. Springer, 2015.

[5] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proc. of the 4th ACM RecSys*, pages 39–46, 2010.

[6] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[7] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[8] J. Gaillard and J.-M. Renders. Time-sensitive collaborative filtering through adaptive matrix completion. In *Advances in Information Retrieval*, pages 327–332. Springer, 2015.

[9] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proc. of the 17th ACM SIGKDD*, pages 69–77, 2011.

[10] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[11] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proc. of the 8th IEEE ICDM*, pages 263–272, 2008.

[12] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[13] A. Kyrola, G. E. Blelloch, and C. Guestrin. Graphchi: Large-scale graph computation on just a pc. In *OSDI*, volume 12, pages 31–46, 2012.

[14] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2000.

[15] J. Lee, S. Bengio, S. Kim, G. Lebanon, and Y. Singer. Local collaborative ranking. In *Proc. of the 23rd ACM WWW*, pages 85–96, 2014.

[16] C.-J. Lin. Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19(10):2756–2779, 2007.

[17] R. J. Little and D. B. Rubin. Statistical analysis with missing data. 2002.

[18] B. Marlin, R. S. Zemel, S. Roweis, and M. Slaney. Collaborative filtering and the missing at random assumption. In *Proc. of the 23rd Conference on Uncertainty in Artificial Intelligence*, 2007.

[19] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proc. of the 7th ACM RecSys*, pages 165–172, 2013.

[20] R. Pan and M. Scholz. Mind the gaps: weighting the unknown in large-scale one-class collaborative filtering. In *Proc. of the 15th ACM SIGKDD*, pages 667–676. ACM, 2009.

[21] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *Proc. of the 8th IEEE ICDM*, pages 502–511, 2008.

[22] S. Rendle. Factorization machines. In *Proc. of ICDM 2010*, pages 995–1000, 2010.

[23] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proc. of the 25th Conference on Uncertainty in Artificial Intelligence*, pages 452–461. AUAI Press, 2009.

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

[25] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proc. of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, 1994.

[26] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38, 2014.

[27] V. Sindhwani, S. S. Bucak, J. Hu, and A. Mojsilovic. One-class matrix completion with low-density factorizations. In *Proc. of the 10th IEEE ICDM*, pages 1055–1060, 2010.

[28] H. Steck. Training and testing of recommender systems on data missing not at random. In *Proc. of the 16th ACM SIGKDD*, pages 713–722. ACM, 2010.

[29] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proc. of the 26th ACM ICML*, pages 1113–1120. ACM, 2009.

[30] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Algorithmic Aspects in Information and Management*, pages 337–348. Springer, 2008.