December, 2016

# Incremental Learning for Matrix Factorization in Recommender Systems

Ole J Mengshoel
Tong Yu, *Carnegie Mellon University*
Nimish Radia, *Ericsson*
Alvin Jude, *Ericsson*
Eugen Feller, *Ericsson*, et al.

# Incremental Learning for Matrix Factorization in Recommender Systems

Tong Yu, Ole J. Mengshoel
Electrical and Computer Engineering
Carnegie Mellon University
{tong.yu, ole.mengshoel}@sv.cmu.edu

Alvin Jude, Eugen Feller, Julien Forgeat, Nimish Radia
Ericsson Silicon Valley
{alvin.jude.hari.haran, eugen.feller}@ericsson.com
{julien.forgeat, nimish.radia}@ericsson.com

***Abstract*—Recommender systems play a key role in personalizing service experiences by recommending relevant items to users. One popular technique for producing such personalization at scale is collaborative filtering via Matrix Factorization (MF). The essence of MF is to train a model by factorizing a sparse rating matrix consisting of users' ratings of item. Unfortunately, existing MF methods require model Learning from Scratch when new data (for users, items, or user ratings) arrive. Learning large models from scratch incurs significant computation cost and typically also results in stale recommendations. With increasing amounts of data and a need for real-time recommendations, incremental learning is desirable. In this paper, we develop a novel but simple method for incremental learning of MF models, called One-sided Least Squares, and demonstrate its parallel implementation via Apache Spark. We also describe how to integrate it with batch learning via Alternating Least Squares (ALS). Unlike previous incremental learning methods, we study our method's approximation of the results of ALS, while significantly reducing compute and storage costs. Our theoretical analysis and experimental results on three real-world datasets suggest that One-sided Least Squares achieves prediction accuracy close to Learning from Scratch with ALS at substantially faster learning speeds. This fast and accurate method for incremental learning enables improved Web-scale recommender systems.**

***Keywords*-Incremental Learning; Least Squares; Matrix Factorization; Recommender Systems; Big Data; Spark**

## I. Introduction

Recommender systems produce individualized recommendations or guide a user to useful objects in a large space of possible options in a personalized way [1], thus addressing the problem of information overload [2]. Reducing information overload has several positive benefits for consumers and retailers alike. Amazon's growth of 29% in 2012 was largely attributed to recommenders being integrated into the purchasing pipeline [3]; Google News improved Google's traffic by 38% [4] partly due to recommenders; and Netflix claimed that 75% of movies watched were from their recommendations [5].

The accuracy of a recommender's model is important as it affects the user experience [6]. However, user-centric research in recommender systems revealed other essential factors [7]. One such factor is the *response time* of the system, defined as the time elapsed for a user to receive recommendations after providing input [8]. Faster response time is preferred. A related factor is the recency effect, where new items (products and services) tend to sell better [9]. We therefore conclude that successful recommender systems should have these features: (1) new users should promptly get relevant recommendations; (2) new items should quickly be recommended to current users; (3) newly rated existing items should immediately yield improved recommendations for a current user.

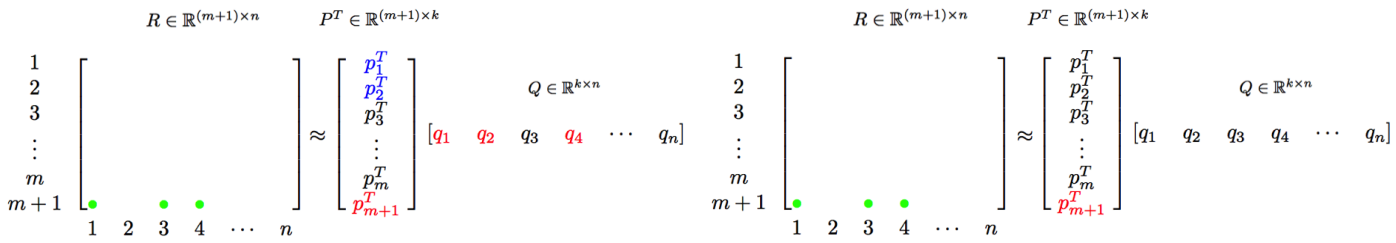
A popular approach to building recommender systems is to use collaborative filtering techniques, often realized through Matrix Factorization (MF) methods. MF often achieves superior rating prediction accuracy [10] but has one big drawback: the high cost of model-building [11]. Adding new information to the matrix—such as a new item, a new user, or a new user-item rating—requires the entire MF model to be learned from scratch. The rapid change in users and items commonly found in global Web deployments, along with the high cost of MF model-building, undermines the three features of successful recommender systems mentioned above.

In this paper, we propose a novel MF machine learning method for decreasing the response time while maintaining recommender accuracy. We achieve this by incrementally updating the MF model on the arrival of either a new user, a new item, or a new rating from an existing user to an existing item. Our method is derived from Alternating Least Squares (ALS) [12] and maintains its benefits such as prediction accuracy and scalability. Meanwhile, it complements ALS in handling new data. Our main contributions are:

- A novel incremental learning method for MF, called One-sided Least Squares (One-sided LS). We show how One-sided LS can be integrated with ALS in practice.
- Theoretical analysis and experimental results suggesting that One-sided LS provides accuracy near-equal to Learning from Scratch with ALS, at much faster learning speeds. Our method also compares advantageously to other incremental learning methods for MF.
- A parallel variant of One-sided LS via an implementation using Apache Spark [13], a prominent distributed data processing framework.

## II. The Matrix Factorization Problem

MF is a classical approach to collaborative filtering. In this section, we discuss both the MF problem and its incremental variant.




a Incremental MF via traditional Two-sided LS.

b Incremental MF via proposed One-sided LS.

Fig. 1: Two incremental MF methods. The new ratings from new user $u_{m+1}$ are signified by the green dots in $R$. In the traditional Two-sided LS approach shown in (a), this requires an update of the user's latent vectors in $p_{m+1}$ and of item latent vectors $q_1$, $q_3$ and $q_4$ as marked in red. If users $u_1$ and $u_2$ have previously rated the first item, their respective latent vectors $p_1$ and $p_2$ (marked as blue) and related ratings $r_{11}$, $r_{13}$, $r_{14}$, $r_{21}$, $r_{23}$, and $r_{24}$ need to be accessed in-memory to update $q_1$, $q_3$, and $q_4$. In contrast, One-sided LS only requires an update of the user's latent vector $p_{m+1}$, as shown in (b).

### A. Traditional MF for Recommender Systems

*Definition 2.1:* (*Matrix factorization (MF) model*) In MF, matrices $R \in \mathbb{R}^{m\times n}$, $P \in \mathbb{R}^{m\times k}$ and $Q \in \mathbb{R}^{k\times n}$ represent users' ratings to items, users' latent vectors and items' latent vectors respectively. The entry in the $u$-th row and the $v$-th column of $R$, that is, $r_{u,v}$, is the rating user $u$ gives item $v$. The $u$-th row vector $p_u$ of $P$ and $v$-th column vector $q_v$ of $Q$ are user $u$ and item $v$'s latent vectors respectively.

*Definition 2.2:* (*Matrix factorization (MF) problem*) In the MF model, some $r_{u,v}$ are known while others are not. Based on known $r_{u,v}$, our goal is to factorize $R$ into $P$ and $Q$, under the constraint that $\tilde{R} = PQ$ should approximate $R$ well. Let $\lambda_P \in \mathbb{R}$ and $\lambda_Q \in \mathbb{R}$ be regularization parameters. In general, the MF problem to optimize

$$\min_{P,Q}\Big(\sum_{(u,v)\in R}(r_{u,v} - p_u q_v)^2 + \lambda_P \|p_u\|^2 + \lambda_Q \|q_v\|^2\Big). \quad (1)$$

Matrices $P$ and $Q$ are usually randomly initialized, and optimization methods are then applied to compute $P$ and $Q$.

### B. The Problem of Incremental Learning in MF

The traditional MF model assumes that the number of users $m$ and items $n$ are known and fixed in learning and prediction. However, in most real-world recommender systems, new users $\{u_{m+1}, u_{m+2}, \cdots\}$ and items $\{v_{n+1}, v_{n+2}, \cdots\}$ arrive incrementally. In these cases, the traditional MF model can not incrementally update the latent vectors $\{p_{u_{m+1}}, p_{u_{m+2}}, \cdots\}$ and $\{q_{v_{n+1}}, q_{v_{n+2}}, \cdots\}$ for these new users and items, because they are not contained in the original models.

*Definition 2.3:* (*Incremental MF model*) Let new users $\{u_{m+1}, u_{m+2}, \cdots\}$ and items $\{v_{n+1}, v_{n+2}, \cdots\}$ arrive incrementally. Incremental learning of an MF model is to learn latent vectors $P$ and $Q$ for these new users and items, to provide good recommendations for them.

Note that which parts of $P$ and $Q$ are updated vary between different incremental methods, as discussed in the remainder of this work. An incremental MF method with low computational cost but high accuracy is preferred.

## III. Related Work

There are several methods of performing the Matrix Factorization in Equation 1: Singular Value Decomposition (SVD), Stochastic Gradient Descent (SGD), and ALS [10], [12].

SVD is a traditional matrix decomposition method. Recommender systems usually contain sparse data, which typically can be handled better by SGD and ALS than by SVD [10]. SGD and ALS optimize $p_u$ and $q_v$ in an alternating manner. First, they fix all $q$ as constant and optimize $p$. SGD uses gradient information to find $p$ in an iterative way, while ALS computes a closed-form solution. Then they fix all $p$ and obtain $q$ similarly. The algorithms repeat and terminate when all $p$ and $q$ no longer change substantially. *ALS*, which is also referred as *Two-sided LS* in this paper, is described in Algorithm 1. When new data arrives, we can add this to the existing data and use Two-sided LS to learn an MF model from the full dataset. We call this *Learning from Scratch*.

**Algorithm 1:** Traditional ALS for MF, also referred to as Two-sided LS in this paper.

**Data**: Training data $D$ contains user $u \in \mathbb{N}^+$, item $v \in \mathbb{N}^+$ and rating $r_{uv} \in \mathbb{R}$.
**Input** : The dimension of latent vector $k \in \mathbb{N}^+$, randomly initialized $p_u \in \mathbb{R}^{k\times 1}$ and $q_v \in \mathbb{R}^{k\times 1}$, number of user $m \in \mathbb{N}^+$, and number of item $n \in \mathbb{N}^+$.
**Result**: Latent vector $p_u$ for users and $q_v$ for items.
**while** *the result does not converge* **do**
  **for** $u \leftarrow 1$ **to** $m$ **do**
    $p_u = (\sum_{r_{uv}\in r_{u*}} q_v q_v^T + \lambda I_k)^{-1} \sum_{r_{uv}\in r_{u*}} r_{uv} q_v$
  **for** $u \leftarrow 1$ **to** $n$ **do**
    $q_v = (\sum_{r_{uv}\in r_{*v}} p_u p_u^T + \lambda I_k)^{-1} \sum_{r_{uv}\in r_{*v}} r_{uv} p_u$

Brand [14] studied how to update MF via Singular Value Decomposition (SVD) in an online fashion, though SVD can not handle sparse data well [10]. Incremental update of the MF model by SGD has been studied [15], [16], [17]. Although these incremental learning methods seem promising, there are

some potential issues. First, the learning rate of incremental SGD needs to be carefully tuned. Second, the number of iterations of SGD is difficult to predetermine, as it depends on how the latent vector is initialized. Updating until convergence is a possible solution, but it could lead to an extremely high number of iterations. This in turn leads to high response time and poor interactivity. Other incremental learning methods emphasize different aspects. Agarwal et al. [18] study offline initialization of the problem. Wang et al. [19] focus on online multi-task collaborative filtering. In contrast, we focus on approximation of a classical MF method [10], with significantly lower computational and storage costs, and easy parallelization in big data scenarios.

## IV. One-sided Methods

In this section, we introduce two One-sided methods: One-sided LS and One-sided SGD. One-sided LS is often preferred due to its several advantages over One-sided SGD, discussed in detail in this section. A theoretical upper bound shows that One-sided LS enjoys very similar expected loss to Two-sided LS, under certain conditions. We also discuss how One-sided LS can be integrated with Two-sided LS, enabling Web-scale recommender systems.

### A. Motivations of One-sided LS

ALS can possibly overcome the issues in incremental learning by SGD as discussed in Section III. First, it is free of learning rate. Second, in each round of learning alternately, when we fix One-sided latent vectors, the problem is convex. Two-sided LS computes closed form solutions, so that in each round of learning only one iteration is needed and the results are independent of the latent vector initialization.

However, there are also some potential issues when making Two-sided LS incremental, which are not well studied in previous work, as detailed below.

First, when the latent vectors for some items are updated, all the users' latent vectors and ratings of these items are required from the previous dataset (*i.e.,* before incremental learning by Two-sided LS). For example, in Figure 1a, new ratings to items $v_1$, $v_3$, and $v_4$ are given by a new user $u_{m+1}$. As a result, vectors $p_{m+1}$, $q_1$, $q_3$, and $q_4$ need to be updated. In addition, if $v_1$, $v_3$ and $v_4$ was rated by user $u_1$ and $u_2$ before, $p_1$, $p_2$, $r_{11}$, $r_{13}$, $r_{14}$, $r_{21}$, $r_{23}$, and $r_{24}$ need to be accessed in order to update $q_1$, $q_3$ and $q_4$. Thus, previous data may need to be stored and accessed for Two-sided LS, adding storage and computation cost.

The second problem is that Two-sided LS needs to recalculate the latent vectors for some users and items that are irrelevant to the incrementally added new users. This is an unnecessary computation cost. The updating operation of $p_u = (\sum_{r_{uv} \in r_{u*}} q_v q_v^T + \lambda I_k)^{-1} \sum_{r_{uv} \in r_{u*}} r_{uv} q_v$ has a complexity of $O(n_u k^2 + k^3)$, where $n_u$ is the number of items rated by user $u$ and $k$ is the dimensionality of latent vectors. In a real situation, $n_u$ might be much larger than $k$. In this case the algorithm's computational bottleneck is updating the latent vectors of the user with the largest number of ratings. For example, suppose that a new user $u_{m+1}$ has 5 ratings and an old user $u_b$ has 50000 ratings. Then Two-sided LS uses $\frac{50000}{5} = 10000$ times more computations than strictly needed.

### B. One-sided LS

**Algorithm 2:** One-sided LS for incremental updates of user latent vectors $p_u$ in MF.

**Input** : Training data $D$ with triples $(u, v, r_{uv})$ representing user $u$'s rating $r_{uv}$ to item $v$. User $u$ is new and item $v$ is old. Item $v$ has latent vector $q_v$, with dimension $k$. Number of new users $m$.
**Result**: Latent vector $p_u$ for new user $u$. $u \in [1, m]$
**for** $u \leftarrow 1$ **to** $m$ **do**
  $p_u = (\sum_{r_{uv} \in r_{u*}} q_v q_v^T + \lambda I_k)^{-1} \sum_{r_{uv} \in r_{u*}} r_{uv} q_v$

To mitigate the problems discussed in Section IV-A, when a new user arrives, latent vectors for items may be fixed, so that only this particular user's latent vectors need to be updated in incremental learning. This is described in Algorithm 2, named as *One-sided LS*. We name ALS as Two-sided LS, as it involves updating by LS on two sides: user and item. One-sided LS has three potential advantages over Two-sided LS in incremental learning of an MF model.

First, One-sided LS can efficiently reduce computation, disk and memory costs. ,An example in Figure 1b shows how costs are saved with our approach, compared with Two-sided LS. When a new user $u_{m+1}$ adds ratings for items $v_1$, $v_3$ and $v_4$, only $p_{m+1}$ needs to be updated. This avoids the computation of $q_1$, $q_2$ and $q_3$ required in the traditional Two-sided approach in Figure 1a. Besides, we conserve the memory required for $p_1$, $p_2$, $r_{11}$, $r_{13}$, $r_{14}$, $r_{21}$, $r_{23}$, and $r_{24}$.

Second, compared to Learning from Scratch, One-sided LS also early and efficiently reduces the learning time. Here is a detailed analysis. Let us assume that the number of iterations of the outer loop is $T_{max}$ in Learning from Scratch, and define $N_b$ as the number of ratings of the user $u_b$ who rates the most in the previous dataset. If a new user $u_{m+1}$ rates $N_{m+1}$ items, the computation cost ratio between incremental learning with One-sided LS and Learning from Scratch with Two-sided LS (with a good parallelization) will be upper bounded by $\frac{N_{m+1}}{N_b \times T_{max}}$, where typically $N_{m+1} \ll N_b$.

Third, for One-sided LS it is easier to parallelize the update for multiple new users. When we update the latent vectors of user $u$, we do not need to access other users' latent vectors. However, the parallel implementation of traditional ALS (*e.g.*, Spark MLlib) would face this issue. With this advantage, we can update the model faster compared to traditional ALS, as seen in our experiments.

### C. One-sided SGD

When using SGD to solve the MF problem, we distinguish between Two-sided SGD and One-side SGD, according to whether the one-sided latent vector is fixed or not. Traditional SGD [16] is called Two-sided SGD in the rest of our paper and

| Notation | Meaning |
|---|---|
| $(P_P, R_P)$ | latent vectors and ratings of Previous users |
| $(P_N, R_N)$ | latent vectors and ratings of New users |
| $(P_B, R_B)$ | $(P_P, R_P) \cup (P_N, R_N)$ |
| $m$ | # ratings by previous users (the size of $R_P$) |
| $n$ | # ratings by new users (the size of $R_N$) |
| $h$ | a possible hypothesis (regression model) |
| $\hat{L}_B(h)$ | empirical loss of $h$ on $(P_B, R_B)$ |
| $\hat{L}_P(h)$ | empirical loss of $h$ on $(P_P, R_P)$ |
| $\hat{L}_N(h)$ | empirical loss of $h$ on $(P_N, R_N)$ |
| $L_B(h)$ | expected loss of $h$ on $(P_B, R_B)$ |
| $L_P(h)$ | expected loss of $h$ on $(P_P, R_P)$ |
| $L_N(h)$ | expected loss of $h$ on $(P_N, R_N)$ |
| $M$ | upper bound of the loss function |

TABLE I: For learning the latent vector $q_{v_j}$ of an item $v_j$, the notations used in theoretical analysis in Section IV-D.

used to benchmark our other approaches. In Two-sided SGD, the latent vectors of users and items are updated by SGD in an alternating way (similar to Two-sided LS). If instead the one-sided latent vector is fixed, we have One-sided SGD, which to our knowledge has not been studied before.

Compared to One-sided SGD, One-sided LS may have several advantages: (i) Initialization of the latent vectors is less important, as the closed form solution in One-sided LS is independent of initialization. In contrast, the initialization impacts the convergence of One-sided SGD. (ii) The learning rate of One-sided SGD needs to be carefully selected to obtain good results. Due to the closed form solution in One-sided LS we do not need to have this concern.

### D. Expected Loss of One-sided LS

Given a rating $r_{uv}$ of user $u$ to item $v$, when One-sided LS updates the latent vector $p_u$ for this new user $u$, it assumes that the updating procedure for item $v$'s latent vector $q_v$ converges if there are enough previous users ratings of $v$. In this section, we analyze for One-sided LS and Two-sided LS (i) the connection between the number of previous user ratings for item $v$ and (ii) the difference in expected loss. The notation used is summarized in Table I.

For item $v_j$, let $P_N$ and $R_N$ be the latent vectors and ratings for the new users. Similarly, let $P_P$ and $R_P$ be the latent vectors and ratings for the previous users. We assume that the new users' data $(P_N, R_N)$ and the previous users' data $(P_P, R_P)$ are sampled i.i.d. from the same distribution. To predict the ratings of new users to item $v_j$, we train a regression model, where the input is $P_N$ and the output is $R_N$. The regression model is exactly the latent vector $Q_{v_j}$ in the MF model. First we introduce Lemma 4.1, which follows from Hoeffding's inequality. Then we prove Theorem 4.2.

*Lemma 4.1:* Consider a least squares regression problem with input space $P$ and output space $R$. There is a regression[1] function $f : P \to R$. Assume that $(p_1, r_1), \cdots, (p_m, r_m)$ are data sampled i.i.d. from an unknown distribution $D$. The loss function is defined as $\ell(r, \hat{r}) = (r - \hat{r})^2$, the empirical loss is defined as $\hat{L}(h) = \frac{1}{m}\sum_{i=1}^{m} \ell(h(p_i), r_i)$ and the expected loss is defined as $L(h) = \mathrm{E}_{x\sim D}[\ell(h(p), f(p))]$. Assume that the hypothesis set $H$ is finite in the least square regression problem. Then, for any $\delta > 0$, with probability at least $1 - \delta$, the following inequality holds for all $h \in H$:

$$\left\| L(h) - \hat{L}(h) \right\| \leq M\sqrt{\frac{\log|H| + \log\frac{2}{\delta}}{2m}}.$$

*Proof:* Following Hoeffding's inequality and the proof in [20], we easily have, with probability $1 - \delta$,

$$\left\| L(h) - \hat{L}(h) \right\| \leq M\sqrt{\frac{\log|H| + \log\frac{2}{\delta}}{2m}}. \tag{2}$$

■

Lemma 4.1 indicates that the difference between the expected loss $L(h)$ on new users' data (*e.g.*, $(P_N, R_N)$) and empirical loss $\hat{L}(h)$ on previous users' data (*e.g.*, $(P_P, R_P)$) is well bounded, if there are enough ratings from previous users. Let $P_B$ and $R_B$ be the latent vectors and ratings for both previous users and new users to item $v_j$. In the following Theorem, we aim to show that the expected loss $L_B(h)$ on $(P_B, R_B)$ and the expected loss $L_N(h)$ on $(P_N, R_N)$ is well bounded, if there are enough ratings from previous users in $(P_B, R_B)$. That is, if the number of ratings in $(P_B, R_B)$ is large enough, the latent vector $Q_{v_j}$ can be fixed, which leads to our One-sided LS.

*Theorem 4.2:* In an MF model $P, Q$ (see Definition 2.1), suppose for a particular item $v_j$ the latent vectors of users who rate $v_j$ are sampled i.i.d. from the same unknown distribution. Assume before incremental date arrives that $v_j$ was rated by $m$ users, and that the latent vector $q_{v_j}$ of $v_j$ was obtained by least squares based on those $m$ users. Now we have $n$ new users rating item $v_j$. Assume that $L_P(h)$ is the expected loss of least squares when fitting $m$ data points $(P_P, R_P)$, and $L_B(h)$ is the expected loss of least squares fitting $m + n$ samples $(P_B, R_B)$. When $m$ is big enough, with probability at least $(1-\delta)^3$, the following inequality holds for all $h \in H$:

$$\|L_B(h) - L_P(h)\| \leq 3M\sqrt{\frac{\log|H| + \log\frac{2}{\delta}}{2m}}. \tag{3}$$

*Proof:* According to the triangle inequality, we have

$$\begin{aligned}
&\|L_B(h) - L_P(h)\| \\
&\leq \left\| L_B(h) - L_P(h) + \hat{L}_P(h) - \hat{L}_B(h) \right\| \\
&\quad + \left\| \hat{L}_B(h) - \hat{L}_P(h) \right\| \\
&\leq \left\| L_B(h) - \hat{L}_B(h) \right\| + \left\| \hat{L}_P(h) - L_P(h) \right\| \\
&\quad + \left\| \hat{L}_B(h) - \hat{L}_P(h) \right\|.
\end{aligned} \tag{4}$$

According to (2), with probability $1 - \delta$ we have

$$\left\| L_B(h) - \hat{L}_B(h) \right\| \leq M\sqrt{\frac{\log|H| + \log\frac{2}{\delta}}{2(m+n)}}, \tag{5}$$

[1]For the item $v_j$, the corresponding latent vector $q_{v_j}$ is the parameter of the regression model here.

where $m+n$ is the amount of training data $(P_B, R_B)$. Again, using (2), with probability $1-\delta$ we have

$$\left\| L_P(h) - \hat{L}_P(h) \right\| \leq M \sqrt{\frac{\log |H| + \log \frac{2}{\delta}}{2m}}. \tag{6}$$

If we consider the first $m$ samples as training data and the additional $n$ samples as validation data, with probability $1-\delta$,

$$\left\| \hat{L}_N(h) - \hat{L}_P(h) \right\| \leq M \sqrt{\frac{\log |H| + \log \frac{2}{\delta}}{2m}}. \tag{7}$$

According to the definition of empirical loss We also have

$$\begin{aligned} \hat{L}_B(h) &= \frac{m\hat{L}_P(h) + n\hat{L}_N(h)}{m+n}, \\ \hat{L}_N(h) &= \frac{(m+n)\hat{L}_B(h) - m\hat{L}_P(h)}{n}. \end{aligned} \tag{8}$$

According to (7), (8), and since $\frac{m+n}{n} \geq 1$, we have

$$\left\| \hat{L}_B(h) - \hat{L}_P(h) \right\| \leq M \sqrt{\frac{\log |H| + \log \frac{2}{\delta}}{2m}}. \tag{9}$$

When the left hand side of (9) is well bounded and $m$ is big enough, approximately $\hat{L}_B(h) \approx \hat{L}_P(h)$.

When $h^* = \arg\min_h \hat{L}_P(h)$, we prove that $h^* = \arg\min_h \hat{L}_B(h)$ by contradiction. If there exits $\tilde{h}$ that $\hat{L}_B(\tilde{h}) < \hat{L}_B(h)$, then there exits $\tilde{h}$ such that

$$\hat{L}_P(\tilde{h}) \approx \hat{L}_B(\tilde{h}) < \hat{L}_B(h) \approx \hat{L}_P(h), \tag{10}$$

which contradicts $h^* = \arg\min_h \hat{L}_P(h)$. Therefore we have $h^* = \arg\min_h \hat{L}_B(h)$. It means that when $m$ is big enough, the optimal hypothesis $h^*$ for the previous dataset $(P_P, R_P)$ with size $m$, and the optimal hypothesis $h^*$ for the dataset $(P_B, R_B)$ containing both previous data $(P_P, R_P)$ and added new data $(P_N, R_N)$ are the same.

With probability $1-\delta$, we have (5), (6) and (9). By substituting (5), (6) and (9) into (4), with probability $(1-\delta)^3$,

$$\begin{aligned} &\|L_B(h) - L_P(h)\| \\ &\leq M \sqrt{\frac{\log |H| + \log \frac{2}{\delta}}{2(m+n)}} + 2M \sqrt{\frac{\log |H| + \log \frac{2}{\delta}}{2m}} \\ &\leq 3M \sqrt{\frac{\log |H| + \log \frac{2}{\delta}}{2m}}. \end{aligned} \tag{11}$$

■

In (11), $M$'s scale is usually very similar to the scale of the ratings in the regression problem. The bound in (11) is similar to the bound of the difference between the expected loss and the empirical loss of the least square regression model in (2), except a constant ratio of 3. Consider an MF model with new users' ratings of previous items, if the previous items are rated by enough previous users, the difference between the expected losses of One-sided LS and Two-sided LS is well bounded.

### *E. Incremental Learning Example*

We illustrate the difference between fixing latent vectors (One-sided LS) and updating latent vectors (Two-sided LS) in a simple example with dimension of latent vector $k$ = 1. The synthetic data is sampled i.i.d. from a distribution. Figure 2 (a)(c)(e) show an example with a small amount of previous data (blue triangles) before incremental learning. In contrast, Figure 2 (b)(d)(f) show an example with a larger amount of previous data (blue triangles). In (a), the blue triangles and line show the initial data and line fitted by least squares. In (c), some new data (red dots) arrives. In (e), the blue line fits to previous blue triangles and the magenta line fits to both blue triangles and red dots. (b)(d)(f) show the same procedure as (a)(c)(e), except that the number of initial data points is larger. We can observe a much smaller difference between the blue line and the magenta line in (f) compared to in (e). This example illustrates that if the latent vectors in MF are learned on a reasonably large amount of data, they can be fixed even if new data is available.

### *F. Towards Large-scale Recommender Systems*

In practice, One-sided LS needs to be integrated with Two-sided LS for two reasons. First, one input to One-sided LS is an MF model learned on previous data, typically done by Two-sided LS. Second, Theorem 4.2 assumes that previous users' and new users' ratings of items are sampled i.i.d. from the same distribution. This is reasonable when the previous training data size is large enough and the distribution of new data is similar to that of previous data. After a period of time, however, the i.i.d. assumption may not continue to hold. In this case, we need to train a new MF model on the whole dataset, or a recent subset thereof, by Two-sided LS.

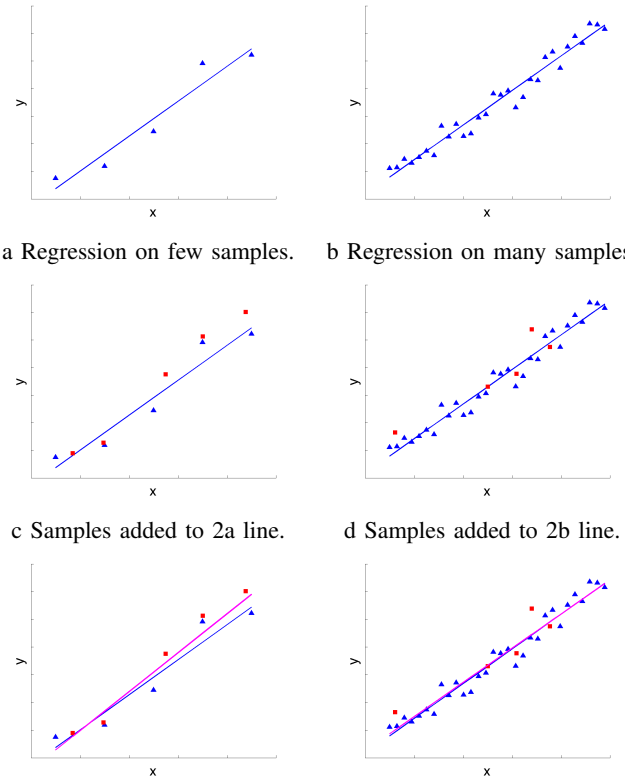
Algorithm 3, which we call *Integrated LS*, shows how to integrate Two-sided LS and One-sided LS. First, we train an MF model by Two-sided LS on the current dataset. In the event of new data arrival, we update $p$ and $q$ using One-sided LS. When the amount of incremental data reaches some threshold, we retrain the model using Two-sided LS. Practically, we may use a validation dataset to predetermine this threshold. This Integrated LS framework aligns with the Netflix Architecture [21]. However, we are not aware of any empirical results for the Netflix Architecture. In contrast, we validate our method via experiments, see Section V.

## V. Experiments

We performed several experiments to study the efficiency of our One-sided LS method in terms of prediction accuracy and learning time.

### *A. Experimental Setup*

We used three different datasets in our experiments: Jester Dataset 2+ [22], MovieLens 10M, and MovieLens 20M [23]. To simulate incremental learning with these datasets, we first removed users or items with ratings fewer than some threshold



a Regression on few samples. b Regression on many samples.

c Samples added to 2a line. d Samples added to 2b line.

e New and old lines are different. f New and old lines are very similar.

Fig. 2: Intuition behind One-sided LS. *Left column:* With regression from a small sample, an updated model may change substantially after samples are added. *Right column:* When the model is built from a large sample, adding more samples will not alter the model much. Theorem 4.2 provides analysis.

**Algorithm 3:** The Integration of Two-sided LS and One-sided LS. Prev. is abbreviation for previous.

**Input** : Training data $D$ with triples $(u, v, r_{uv})$ representing user $u$'s rating $r_{uv}$ to item $v$.
**Result**: Latent vector $p$ for users and $q$ for items.
**while** *the model is kept being updated* **do**
  $P$, $Q$ = **Two_sided_LS** $(D)$
  **while** *the incremental data is not many enough* **do**
    **if** *new user $i$ rates prev. item $j$ as $r_{ij}$* **then**
      $p_i$ = **One_sided_LS**$(null, Q, r_{ij})$;
      $P = P \cup p_i$; $D = D \cup \{(i, j, r_{ij})\}$
    **if** *prev. user $i$ rates new item $j$ as $r_{ij}$* **then**
      $q_j$ = **One_sided_LS**$(P, null, r_{ij})$;
      $Q = Q \cup q_j$; $D = D \cup \{(i, j, r_{ij})\}$
    **if** *prev. user $i$ rates prev. item $j$ as $r_{ij}$* **then**
      /*$r_{ij}^{prev.}$ is the prev. rating of item $j$ by user $i$.*/
      $Q = Q \backslash q_j$; $P = P \backslash p_i$;
      $D = D \backslash \{(i, j, r_{ij}^{prev.})\}$;
      $p_i$ = **One_sided_LS**$(null, Q, r_{ij})$;
      $P = P \cup p_i$;
      $q_j$ = **One_sided_LS**$(P, null, r_{ij})$;
      $Q = Q \cup q_j$; $D = D \cup \{(i, j, r_{ij})\}$

| | Adding New Users | | | Adding New Items | | |
|---|---|---|---|---|---|---|
| | Jester | 10M | 20M | Jester | 10M | 20M |
| # Users | 26151 | 69878 | 138493 | 59132 | 69878 | 138493 |
| # Items | 140 | 10677 | 26744 | 140 | 8940 | 13132 |
| # Ratings | 1690525 | 10000054 | 20000263 | 1761439 | 9984502 | 19933089 |

TABLE II: After removing users or items with fewer than $T = 20$ ratings, the experimental datasets have the number of users, items, and ratings reported here.

$T$. We set $T = 20$ in our experiments.[2] We then partitioned each dataset by user: 80% of users' data was used as training data $D_{tr}$ and 20% of users' data as test data $D_{te}$. We then further partitioned each user's data in $D_{te}$, by rating. For each user or item in the incremental learning, their $N_{inc}$ ratings are randomly selected for incremental learning, and the rest is used for evaluation. The exact value of $N_{inc}$ influences the results, and is discussed as part of the evaluation. The data after processing is described in Table II.

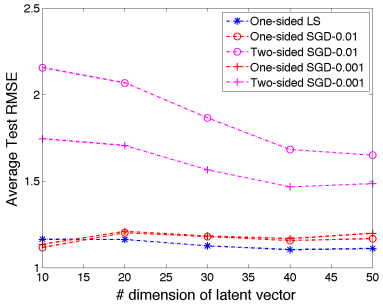
The experiments were run on a server with 8 processors and 64 GB memory. We implemented One-sided LS in Apache Spark [24], to evaluate the execution time in this platform. We also used Spark MLlib's implementation of ALS to generate the original model with $P$ and $Q$, before incremental learning by One-sided LS, One-sided SGD, Two-sided SGD and several baselines described in V-B.

Algorithms were evaluated by prediction accuracy and execution time spent on incremental learning, measured in wall-clock time. The metric used to evaluate prediction accuracy is Root Mean Square Error (RMSE), $\sqrt{\frac{\sum_{(u,v)\in D_{te}} (r_{u,v} - \hat{r}_{u,v})^2}{|D_{te}|}}$, where $r_{u,v}$ is the rating from test data $D_{te}$ with $|D_{te}|$ instances, and $\hat{r}_{u,v}$ is the corresponding rating prediction. We calculated RMSE on each test user's data and averaged over all RMSEs of all users, and report average test RMSE. Some users have 1000 times more ratings than others, and we wanted to avoid having users with these high numbers of ratings dominate the evaluation. We were more concerned about the prediction accuracy under incremental learning, as opposed to

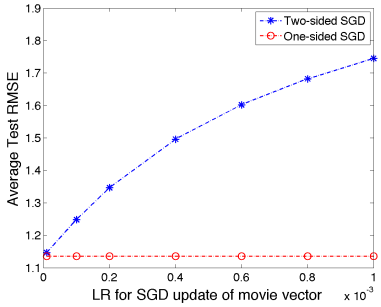
| | Adding New Users | | | Adding New Items | | |
|---|---|---|---|---|---|---|
| | Jester | 10M | 20M | Jester | 10M | 20M |
| Random Initialization | 5.380 | 3.807 | 3.777 | 5.726 | 3.714 | 3.556 |
| Random Sampling | 6.062 | 1.335 | 1.221 | 5.492 | 1.643 | 3.058 |
| Learning from Scratch | 4.174 | 1.179 | 1.158 | 4.445 | 1.171 | 1.203 |
| One-sided LS | 4.277 | 1.173 | 1.156 | 4.435 | 1.187 | 1.223 |

TABLE III: A comparison between our One-sided LS method and three others in terms of accuracy (in RMSE). One-sided LS is almost as good as Learning from Scratch, using Two-sided LS, when adding new users and adding new items.

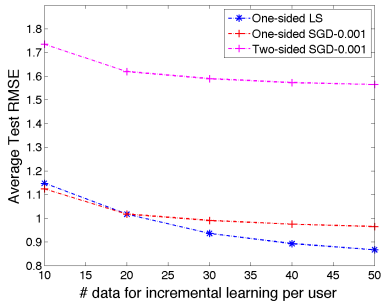
[2]The MovieLens 10M and 20M datasets were preprocessed by the provider, such that all users had rated at least 20 movies. We therefore removed movies with fewer than 20 ratings. The same preprocessing was applied to the Jester Data.




a Comparison of One-sided LS, One-sided SGD and Two-sided SGD. The $x$-axis shows the dimension $k$ of the latent vector.




b Comparison of One-sided SGD and Two-sided SGD. The $x$-axis shows the learning rate for updating the user vector.




c The effect of number of known ratings $N_{inc}$ per new user. The $x$-axis shows the number of known ratings per user.

Fig. 3: Comparing SGD methods and One-sided LS on MovieLens 10M. The average test RMSE is shown on the $y$-axis.

the prediction of the entire test dataset.

### B. Accuracy Comparison with ALS

We compared the accuracy of One-sided LS in two scenarios: (1) adding new users and (2) adding new items. It is benchmarked against three existing methods:

- Random Initialization: User's or item's latent vectors are randomly initialized.
- Random Sampling: Latent vectors are randomly selected from an existing user or item's latent vector [25].
- Learning from Scratch: Two-sided LS (ALS [10]).

Table III shows that One-sided LS outperforms Random Initialization and Random Sampling, and performs similar to Learning from Scratch. The performance of One-sided LS is independent of the latent vector initialization, as the model is updated by closed form solution with only one iteration.

### C. Accuracy Comparison with SGD

We studied the performance of One and Two-sided SGD, and One-sided LS on the MovieLens 10M dataset. Three questions are studied. First, how well One-sided LS can predict, compared to Two-sided LS and Two-sided SGD. Second, how SGD's performance changes with different learning rates. Third, how One-sided SGD and One-sided LS perform with varying amount of incremental learning data.

Figure 3(a) shows that SGD is very sensitive to its learning rate, and that One-sided SGD and One-sided LS are generally better than Two-sided LS. In Figure 3(b), we fixed the learning rate for the user vector and gradually increased the learning rate for the movie vector from 0 to 0.0001. As a result, the performance gap between One-sided SGD and Two-sided SGD became larger. In Figure 3(c), we increased $N_{inc}$, and observed that the RMSE improves. However, new users usually have few ratings at the start. So we fix $N_{inc} = 10$ for the rest of the experiment.

One-sided LS and One-sided SGD outperform Two-sided SGD in our experiments, especially due to Two-sided SGD's sensitivity to its learning rate (around 40% decrease of average test RMSE is seen in Figure 3(a)). If the learning rate of Two-sided SGD is not carefully tuned, an unfortunate update by SGD may jeopardize the finely learned model.

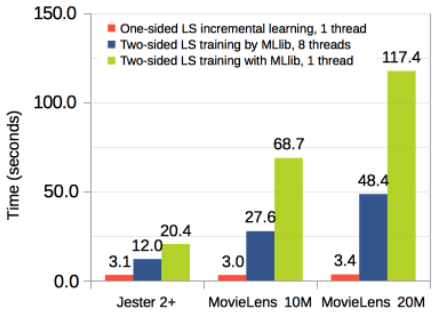



Fig. 4: A comparison of One-sided LS and Learning from Scratch with Two-sided LS on Spark MLlib using 1 or 8 threads.

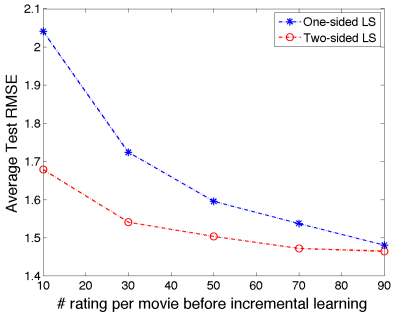



Fig. 5: For MovieLens 10M, when the number of previous ratings for a movie increases, the RMSE gap between One-sided LS and Two-sided LS decreases.

### D. Execution Time Comparison

This section compares the execution time of incremental learning by One-sided LS in Spark and Learning from Scratch with ALS in MLlib, when we have one new user. The experiments use Jester Dataset 2+, MovieLens 10M, and MovieLens 20M. Figure 4 shows the results. One-sided LS can integrate this user into an MF model within 4 seconds. But if we use Learning from Scratch, the time is much longer, even with 8 threads. Thus, One-sided LS enables much better user interactivity. As more data becomes available, the time of Two-sided LS learning increases from $12.0s$ for Jester, to $27.6s$

for MovieLens 10M, and to $48.4s$ for MovieLens 20M. In contrast, for One-sided LS, the time is $3.1s$ for Jester, $3.0s$ for MovieLens 10M, and $3.4s$ for MovieLens 20M. In One-sided LS, the updates of user or item latent vectors are totally independent. Thus the computation times for one new user and multiple new users are about the same through the parallel computing techniques discussed in Section IV-A.

### E. Varying Amounts of New Data

The theoretical analysis in Section IV-D shows that One-sided LS can achieve similar accuracy to Two-sided LS, given enough known ratings in the MF model input to One-sided LS. This was validated with an experiment, with results illustrated in Figure 5.

The gap between One-sided LS and Two-sided LS decreases as the number of ratings increases, and converges at 90. This suggest that given an item with around 90 ratings, our approach predicts with similar accuracy to that of Learning from Scratch and we will not need to update the item's latent vector during incremental learning. In fact, $100\%$ of items in Jester, $57\%$ of items in MovieLens 10M, and $57\%$ of items in MovieLens 20M have been rated by more than 90 users.

### F. Summary

Three key outcomes of our experiments are as follows:

- Our incremental learning approach, One-sided LS, achieved accuracy very close to a model learned from scratch on three datasets, with a difference in RMSE between 0.01 to 0.1.
- Our approach required shorter learning time than Learning from Scratch. On average, it took 10% of the time spent by ALS in Spark MLlib with 8 threads.
- Our approach outperformed Two-sided SGD on accuracy, decreasing the RMSE of the MovieLens 10M dataset by about 40%. Moreover, we achieved similar accuracy to One-sided SGD, but without the need for a carefully tuned parameter.

## VI. Conclusions

In this paper, we address the problem of incrementally learning MF models. Incremental learning is important in improving Web users' experience with recommender systems, as it reduced response time. We develop a novel incremental learning method, One-sided LS. Both theoretical analysis and experimental results show that our method achieves very similar test RMSE to learning MF models from scratch with ALS, but at much lower learning time. Moreover, our method does not require complex learning rate tuning as SGD, and can be parallelized by implementation using Apache Spark.

## References


[1] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370, Nov. 2002.

[2] J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang, "Recommender system application developments: A survey," *Decision Support Systems*, vol. 74, pp. 12–32, 2015.

[3] J. Mangalindan, "Amazon's recommendation secret," http://fortune.com/2012/07/30/amazons-recommendation-secret/, 2015, accessed: 2015-07-30.

[4] X. Zhao, W. Zhang, and J. Wang, "Interactive collaborative filtering," in *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM, 2013, pp. 1411–1420.

[5] X. Amatriain and J. Basilico, "Netflix recommendations: Beyond the 5 stars – part 1," http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html, 2015.

[6] B. P. Knijnenburg, M. C. Willemsen, Z. Gantner, H. Soncu, and C. Newell, "Explaining the user experience of recommender systems," *User Modeling and User-Adapted Interaction*, vol. 22, no. 4-5, pp. 441–504, Oct. 2012.

[7] P. Pu, L. Chen, and R. Hu, "A user-centric evaluation framework for recommender systems," in *RecSys-2011*. ACM, 2011, pp. 157–164.

[8] B. Xiao and I. Benbasat, "E-commerce product recommendation agents: Use, characteristics, and impact," *Mis Quarterly*, vol. 31, no. 1, pp. 137–209, 2007.

[9] B. Pathak, R. Garfinkel, R. D. Gopal, R. Venkatesan, and F. Yin, "Empirical analysis of the impact of recommender systems on sales," *Journal of Management Information Systems*, vol. 27, no. 2, pp. 159–188, 2010.

[10] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[11] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in artificial intelligence*, vol. 2009, p. 4, 2009.

[12] 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*. Springer, 2008, pp. 337–348.

[13] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12, 2012.

[14] M. Brand, "Fast online svd revisions for lightweight recommender systems." in *SDM*. SIAM, 2003, pp. 37–46.

[15] S. Rendle and L. Schmidt-Thieme, "Online-updating regularized kernel matrix factorization models for large-scale recommender systems," in *RecSys-2008*. ACM, 2008, pp. 251–258.

[16] J. Vinagre, A. M. Jorge, and J. Gama, "Fast incremental matrix factorization for recommendation with positive-only feedback," in *UMAP-2014*. Springer, 2014, pp. 459–470.

[17] X. Luo, Y. Xia, and Q. Zhu, "Incremental collaborative filtering recommender based on regularized matrix factorization," *Knowledge-Based Systems*, vol. 27, pp. 271–280, 2012.

[18] D. Agarwal, B.-C. Chen, and P. Elango, "Fast online learning through offline initialization for time-sensitive recommendation," in *KDD-2010*. ACM, 2010, pp. 703–712.

[19] J. Wang, S. C. Hoi, P. Zhao, and Z.-Y. Liu, "Online multi-task collaborative filtering for on-the-fly recommender systems," in *RecSys-2013*. ACM, 2013, pp. 237–244.

[20] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*, 2012.

[21] X. Amatriain, "Big & personal: data and models behind netflix recommendations," in *Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*. ACM, 2013, pp. 1–6.

[22] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A constant time collaborative filtering algorithm," *Information Retrieval*, vol. 4, no. 2, pp. 133–151, 2001.

[23] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 5, no. 4, p. 19, 2015.

[24] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10.

[25] N. N. Liu, X. Meng, C. Liu, and Q. Yang, "Wisdom of the better few: cold start recommendation via representative based rating elicitation," in *RecSys-2011*. ACM, 2011, pp. 37–44.