Effective Latent Models for Binary Feedback in Recommender Systems

Maksims N. Volkovs
Milq Inc
151 Bloor Street West
Toronto, ON M5S 1S4
maks@milq.com

Guang Wei Yu
Milq Inc
151 Bloor Street West
Toronto, ON M5S 1S4
guangweiyu@milq.com

ABSTRACT

In many collaborative filtering (CF) applications, latent approaches are the preferred model choice due to their ability to generate real-time recommendations efficiently. However, the majority of existing latent models are not designed for implicit binary feedback (views, clicks, plays etc.) and perform poorly on data of this type. Developing accurate models from implicit feedback is becoming increasingly important in CF since implicit feedback can often be collected at lower cost and in much larger quantities than explicit preferences. The need for accurate latent models for implicit data was further emphasized by the recently conducted Million Song Dataset Challenge organized by Kaggle [18]. In this challenge, the results for the best latent model were orders of magnitude worse than neighbor-based approaches, and all the top performing teams exclusively used neighbor-based models. We address this problem and propose a new latent approach for binary feedback in CF. In our model, neighborhood similarity information is used to guide latent factorization and derive accurate latent representations. We show that even with simple factorization methods like SVD, our approach outperforms existing models and produces state-of-the-art results.

Categories and Subject Descriptors

H.4.m [**Information Systems Applications**]: Miscellaneous

Keywords

Collaborative Filtering; Binary Feedback; Latent Models

1. INTRODUCTION

Emerging popularity of e-commerce and social web has highlighted an important challenge of surfacing relevant content to consumers. Recommender systems have proven to be effective tools for this task receiving a lot of attention recently [2, 5, 14]. One approach that is commonly used to build accurate recommender models is collaborative filtering (CF). CF is a method of making predictions about an individual's preferences based on the preference information from many users. CF has been shown to work well across various domains [2], and many successful web-services such as Netflix, Amazon, YouTube and Yahoo! use CF to deliver personalized recommendations to their users.

The majority of the existing approaches in CF can be divided into two categories: neighbor-based approaches and model-based approaches (we review both types in detail in Section 3). Neighbor-based approaches estimate the item preferences for a target user using the similarities from neighboring users and/or items [23, 12]. In contrast, model-based approaches use the observed preferences to create a compact model of the data which is then used to predict the unobserved preferences.

While neighbor-based models can generate accurate and interpretable recommendations when sufficient data is available, they are very inefficient at test time. Most neighbor models require loading large portions of data to estimate user and/or item similarities, and involve several complex operations most of which are difficult to do efficiently in real-time. In many applications where recommendations need to be generated in real-time, model-based approaches are the preferred choice. These methods build compact memory-efficient representations of the data and can be easily scaled to handle millions of users and items. For this reason we primarily concentrate on model-based approaches in this work.

Preference data that is used to learn CF models can be partitioned into two types: *explicit feedback* and *implicit feedback*. Explicit feedback includes all explicit preference actions from users. Data of this type typically comes in the form of ratings (Netflix, Amazon etc.) or thumbs-up/down selection (Youtube, TiVo etc.). While explicit feedback generally provides high quality signal that accurately describes users' preferences, collecting large amounts of this data required to develop accurate recommender models is notoriously difficult and time consuming. For this reason, much of the recent attention have been devoted to implicit feedback where preferences are inferred indirectly by observing user behavior. Implicit feedback can come in many forms that include plays, purchases, browse histories and even mouse clicks and scrolls. Since no additional action is required from users beyond the normal use of the service, large amounts of implicit feedback can often be cheaply collected in a short amount of time. This advantage however, comes at the expense of increased signal noise.

SIGIR'15, August 09 - 13, 2015, Santiago, Chile.

ACM 978-1-4503-3621-5/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2766462.2767716.

Most of the available implicit data is binary where user either conducts the action (purchase, browse, click etc.) on a given item or no data is available. [1] Binary feedback makes the recommendation problem particularly challenging since it is impossible to gauge the degree of preference from such data. Furthermore, the majority of existing model-based approaches are developed for non-binary data and employ objective functions that rely on graded relevance to distinguish positive preferences from negative ones. Such models tend to perform very poorly on binary data. Recently, several model-based approaches have been developed specifically for binary feedback [13, 20], however results on CF challenges have shown that these models don't perform as well as neighbor-based approaches.

The difference in performance between neighbor- and model-based methods on binary data became especially evident during the Million Song Dataset Challenge (MSD) which was recently conducted by Kaggle [18]. In this challenge, song listening histories were made available for 1.2M users and 380K songs, and the goal was to predict the next 500 songs for a subset of 110K users. Full song listening counts were made available but the organizers found these to correlate poorly with user preferences and binarized the data (1 if user listened to a song and 0 otherwise), using Mean Average Precision to evaluate submissions.

From the winners' reports [3] and the challenge forum [2], it became evident that all of the top scoring teams used variations of neighbor-based methods. Moreover, results for the best performing model-based approach were orders of magnitude worse than the best neighbor-based methods. Similar findings were reported by the challenge organizers who found that recommending songs from the same artist outperformed a complex model-based approach by a significant margin [18]. It was also reported that most model-based methods had run-time and/or memory problems on data of this size, and were either too slow to converge or used too much RAM. Given that 150 teams participated in this challenge and its recency, it can be concluded that there currently is no readily-available model-based approach that can perform well on binary data of this size.

Since the majority of implicit feedback is binary, it is highly desirable to develop accurate model-based methods that scale well to large datasets. In this paper we propose one such approach. The main idea behind our approach is to use neighbor similarities to enrich binary preferences and guide latent factor learning for both users and items. We empirically demonstrate that this approach produces latent representations that outperform neighbor-based methods while being considerably more efficient during inference time.

2. BINARY COLLABORATIVE FILTERING FRAMEWORK

In a typical binary collaborative filtering problem we have a set of N users $\mathcal{U} = \{u_1, ..., u_N\}$ and a set of M items $\mathcal{V} = \{v_1, ..., v_M\}$. The users' binary feedback for the items can be represented by an $N \times M$ matrix $\mathbf{R}$ where $\mathbf{R}(u_n, v_m) = 1$ if user u_n expressed preference (played, purchased, clicked etc.) for item v_m and $\mathbf{R}(u_n, v_m) = 0$ otherwise. We use $\mathcal{U}(v_m)$ to denote the set of all users that preferred v_m and $\mathcal{V}(u_n)$ to denote the set of items that u_n has expressed preference for. We use vector notation: $\mathbf{R}(u_n, :)$ denotes the n'th row of $\mathbf{R}$ ($1 \times M$ vector), and $\mathbf{R}(:, v_m)$ denotes the m'th column ($N \times 1$ vector).

Unlike the traditional CF where the goal is to accurately predict ratings for every user-item pair, in binary domain the aim is to produce a top-$\mathcal{T}$ ranking of the items that the user is most likely to prefer next. A ranking of the items $\mathcal{V}$ can be represented as a permutation $\pi : \{1, ..., M\} \rightarrow \{1, ..., M\}$ where $\pi(m) = l$ denotes the rank of the item v_m and $m = \pi^{-1}(l)$. A number of evaluation metrics have been proposed in information retrieval to evaluate the performance of the ranker. Here we concentrate on two of the more commonly used metrics, Normalized Discounted Cumulative Gain (NDCG) [15] and Mean Average Precision (MAP) [4]. For a given user u and ranking π the NDCG is given by:

$$\begin{aligned}\text{NDCG}(u, \pi, \mathbf{R})@\mathcal{T} &= \frac{1}{G_{\mathcal{T}}(u, \mathbf{R})} \sum_{m=1}^{\mathcal{T}} \frac{2\hat{}\mathbf{R}(u, v_{\pi^{-1}(m)}) - 1}{\log_2(m+1)} \\ &= \frac{1}{G_{\mathcal{T}}(u, \mathbf{R})} \sum_{m=1}^{\mathcal{T}} \frac{\mathbf{R}(u, v_{\pi^{-1}(m)})}{\log_2(m+1)}\end{aligned}$$

where $\mathcal{T}$ is a truncation constant, $v_{\pi^{-1}(m)}$ is the item in position m in π and $G_{\mathcal{T}}(u, \mathbf{R})$ is a normalizing term which ensures that NDCG $\in [0, 1]$ for all rankings. $\mathcal{T}$ is typically set to a small value to emphasize that the user will only be shown the top-$\mathcal{T}$ ranked items and the items below the top-$\mathcal{T}$ are not evaluated.

MAP is defined in terms of average precision (AP):

$$\text{AP}(u, \pi, \mathbf{R})@\mathcal{T} = \frac{\sum_{m=1}^{\mathcal{T}} \text{P}(u, \pi, \mathbf{R})@m \cdot \mathbf{R}(u, v_{\pi^{-1}(m)})}{\sum_{m=1}^{\mathcal{T}} \mathbf{R}(u, v_{\pi^{-1}(m)})}$$

where P@m is the precision at m:

$$\text{P}(u, \pi, \mathbf{R})@m = \sum_{l=1}^{m} \frac{\mathbf{R}(u, v_{\pi^{-1}(l)})}{m}$$

MAP is then computed by averaging AP over all users. Both NDCG and MAP have similar characteristics and discount relevance of each item by its position in the ranking and are maximized when all relevant items are ranked at the top. In binary CF, all items v with $\mathbf{R}(u, v) = 0$ are assumed to be not relevant to u. This relevance assignment is not entirely accurate since u might not have seen v, however, in the absence of graded relevance most frameworks follow this evaluation scheme.

3. EXISTING WORK

In this section we describe existing neighbor- and model-based approaches developed for binary data.

3.1 Neighbor-Based Approaches

In a binary setting, neighborhood-based CF approaches estimate item scores for a target user using the similarity from neighboring users and/or items. Formally, given an item v and target user u, the *user-user* approach estimates the score for v by comparing u with other users that ex-

[1] Note that some explicit data can also come in binary format e.g., Facebook's "likes".

[2] `www.kaggle.com/c/msdchallenge/forums/t/2365/challenge-retrospective-methods-difficulties`

pressed preference for v:

$$\textbf{user-user:}\ \ \mathbf{S}(u,v) = \sum_{u' \in \mathcal{U}(v)} \mathbf{R}(u,:)\mathbf{R}(u',:)^T \qquad (1)$$

The main idea behind this approach is based on the assumption that if users similar to u prefer v then $\mathbf{S}(u,v)$ should be high and v should be recommended to u. Similarly, in the *item-item* approach v is compared to all items that u has expressed preference for:

$$\textbf{item-item:}\ \ \mathbf{S}(u,v) = \sum_{v' \in \mathcal{V}(u)} \mathbf{R}(:,v)^T\mathbf{R}(:,v') \qquad (2)$$

This method follows a similar idea and aims to recommend items that are similar to items that u has expressed preference for.

After estimating scores for all items either via user-user or item-item method the resulting score vector $\mathbf{S}(u,:)$ is sorted and top-$\mathcal{T}$ items are presented to u. In practice it is often found that the accuracy of each method can be significantly improved if dot product is replaced with a more complex metric like cosine similarity. We found this to be especially true for binary domain, where row and column L^2 normalizations tend to work particularly well [3]:

$$\textbf{row-norm:}\ \ \mathbf{R}(u,:) = \frac{\mathbf{R}(u,:)}{\sqrt{\sum_{n=1}^{N} \mathbf{R}(u,v_n)^2}} \qquad (3)$$

$$\textbf{col-norm:}\ \ \mathbf{R}(:,v) = \frac{\mathbf{R}(:,v)}{\sqrt{\sum_{m=1}^{M} \mathbf{R}(u_m,v)^2}} \qquad (4)$$

We found that for most datasets applying *both* normalizations before computing similarities produced gains of up to 30%; further justification for using normalization can be found in [12]. We also found that the order in which these normalizations are applied is very important and should be validated separately for each dataset. Note that when only row-norm or col-norm normalization is applied, item-item (user-user) scores become the sums of cosine similarities.

3.2 Model-Based Approaches

In contrast to neighbor-based approaches, model-based approaches use the observed preferences to create a compact model of the data which is then used to predict the unobserved preferences. The most popular methods in this category are latent models that derive compact latent factors for both users and items and then use these factors to predict preference. Latent models are often the default choice for many CF problems due to their accuracy and efficiency. Once the factors are estimated, recommendations can be generated very efficiently by computing simple dot products between latent factors, allowing these models to be applied in real-time.

Several model-based approaches have been recently proposed for binary setting, here we review two of the more popular approaches WRMF [13] and BPR-MF [20]. WRMF is a regression method and learns user-item factors by minimizing the weighted reconstruction error:

$$\sum_{n=1}^{N}\sum_{m=1}^{M} c_{nm}(\mathbf{R}(u_n,v_m) - \mathbf{U}_r(u_n,:)\mathbf{V}_r(v_m,:)^T)^2 \qquad (5)$$

[3]To avoid introducing extra notation we use self assignment of the form x = f(x) where possible.

where $\mathbf{U}_r$ is an $N \times r$ user factor matrix, $\mathbf{V}_r$ is an $M \times r$ item factor matrix, and c_{nm} is a weight that is set separately for every user-item pair. For binary data c_{nm} is set to:

$$c_{nm} = 1 + \alpha\mathbf{R}(u_n,v_m)$$

This formulation adds an extra α weight to every pair with $\mathbf{R}(u_n,v_m) = 1$ forcing optimization to concentrate on those pairs. Authors of WRMF propose to use alternating least squares to optimize this model with an overall complexity of $O(\mathcal{N}r^2 + (N+M)r^3)$ [13] where $\mathcal{N}$ is the total number of non-zero entries in $\mathbf{R}$. Both $\mathcal{N}r^2$ and $(N+M)r^3$ grow very quickly with r and several teams in the MSD challenge ($\mathcal{N} \approx 50$ million and $N + M \approx 1.6$ million) reported that WRMF took prohibitively long to train even for moderate r sizes.

BPR-MF is a ranking approach and optimizes pairwise objective that attempts to place pairs with observed preference above the unobserved ones:

$$\sum_{(u_n,v_m,v_l)} \log(1 + e^{-\mathbf{U}_r(u_n,:)(\mathbf{V}_r(v_m,:)-\mathbf{V}_r(v_l,:))^T}) \qquad (6)$$

where triplets (u_n, v_m, v_l) are sampled with a condition that $\mathbf{R}(u_n,v_m) = 1$ and $\mathbf{R}(u_n,v_l) = 0$. Similarly to WRMF, this implies an assumption that all items v_l with $\mathbf{R}(u_n,v_l) = 0$ are not relevant and should be ranked below the relevant items v_m with $\mathbf{R}(u_n,v_m) = 1$. Moreover, all relevant (irrelevant) items are assumed to be *equally* relevant (irrelevant). These assumptions often don't hold in real-life scenarios where noisy implicit signals tend to produce incorrect/outlier preferences that should be discounted.

Due to its pairwise nature BPR-MF is also expensive to optimize with runtime complexity of $O(\mathcal{N}M)$. While authors claim that satisfactory convergence can be achieved significantly faster through sampling, no theoretical guarantees are provided. Another disadvantage of BPR-MF is that when the number of items is large, gradient updates are distributed very unevenly (even with sampling). Popular items with many non-zero entries receive the bulk of updates while the tail-region items receive almost none. This can potentially explain the poor performance of this approach on the MSD dataset where challenge organizers found that it was unable to outperform the simple songs by the same artist baseline [18].

4. OUR APPROACH

We discussed earlier that treating all observed (unobserved) items as equally relevant (irrelevant) is not optimal in the binary CF setting. In this section we further build on this idea and propose a model-based approach that applies neighbor similarities to further distinguish relevant items from irrelevant ones. This information is then used to infer accurate user and item factors.

Before we delve into model description, consider the following simplified example: a fantasy fan purchases "The Lord Of The Rings" trilogy, both "Hobbit" releases and "The Devil Wears Prada" (a present for a friend). If we could request explicit preference data from this user we would immediately know that (s)he does not enjoy movies like "The Devil Wears Prada". However, given that we only have access to purchase history, objectives in WRMF and BPR-MF would treat all purchases as equally relevant. During optimization for this user, both methods would aim to derive

latent representations that rank fantasy movies and movies like "The Devil Wears Prada" at the top, which is clearly suboptimal. One way of dealing with this problem is through model regularization and both methods apply strict regularization to penalize user and item factors. However, as the number of outliers increases the problem becomes more severe and might no longer be fixable by heavy regularization.

In contrast, by applying neighbor methods like item-item, we immediately get a lower score for "The Devil Wears Prada" since it's not similar to any other purchase by this user. From this example it is evident that the neighbor approach provides an effective way to resolve ties in the binary preference matrix, and neighbor score matrix $\mathbf{S}$ reflects user preferences more accurately than the raw binary matrix $\mathbf{R}$. Neighbor methods, however, are also not immune to noise. Users/items with very few ratings can heavily skew similarities and result in incorrect rankings. Row and column normalizations can partially alleviate this problem but don't eliminate it completely.

In our model we propose to utilize these advantages of neighbor approaches and factorize the neighbor score matrix $\mathbf{S}$ instead of the original binary matrix $\mathbf{R}$. In addition to producing models that support very efficient inference, applying low-rank factorization to $\mathbf{S}$ can significantly reduce the noise that is often present when similarities are computed on highly sparse data. One method that readily lends itself for the factorization task is truncated Singular Value Decomposition (SVD). A number of techniques that use truncated SVD such as the Principal Component Analysis and the Latent Semantic Analysis have been found to be very robust to noise and generalize well. Moreover, SVD factorization is a well studied problem and efficient distributed implementations exist that can factorize matrices with hundreds of millions of rows (e.g., Mahout's stochastic SVD [17]), making it applicable to virtually any CF problem.

Encouraged by these results we apply truncated SVD to the neighbor score matrix $\mathbf{S}$. We begin by normalizing all rows of $\mathbf{S}$ to have unit norms:

$$\mathbf{S}(u,:) = \frac{\mathbf{S}(u,:)}{\sqrt{\sum_{m=1}^{M} \mathbf{S}(u,v_m)^2}}$$

The L^2 normalization rescales scores to have comparable ranges across users making factorization more stable. This is especially useful when some users/items have considerably more data than others (common in many CF applications) resulting in highly variable score ranges across users. After row normalization we approximate $\mathbf{S}$ with a product of three low rank matrices via SVD:

$$\mathbf{S} \approx \mathbf{U}_r \mathbf{\Sigma}_r \mathbf{V}_r^T \tag{7}$$

Here r is SVD rank, $\mathbf{U}_r$ is an $N \times r$ matrix, $\mathbf{\Sigma}_r$ is an $r \times r$ diagonal matrix and $\mathbf{V}_r$ is an $M \times r$ matrix.

Once the factorization is completed $\mathbf{\Sigma}_r$ is absorbed into $\mathbf{U}_r$, $\mathbf{U}_r = \mathbf{U}_r\mathbf{\Sigma}_r$, and the scores for every user-item pair are calculated by computing dot product between the corresponding user and item factors:

$$\textbf{svd:}\ \ \mathbf{S}(u,v) \approx \mathbf{U}_r(u,:)\mathbf{V}_r(v,:)^T \tag{8}$$

Sorting these scores gives top-$\mathcal{T}$ item recommendations for u. Note that for most r this operation is considerably more efficient than computing and summing neighbor similarities (Equations 1 and 2), especially when the number of users or items is large.

While SVD provides an effective way to derive latent representations, current formulation requires computing and storing the full score matrix $\mathbf{S}$ which is not practical for most large-scale applications. In the following sections we describe several ways of dealing with this problem.

4.1 Block Update

The naive SVD model has runtime complexity of $O(NM\log(r) + (N+M)r^2)$ [4] using the stochastic SVD algorithm [11], and requires $O(NM)$ space. Recently, a number of advances have been made on incremental SVD [7, 1] where factorization is built in stages, allowing to process large amounts of data without explicitly storing it. One of the more efficient algorithms developed by Brand [7] takes as input "current" factorization $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \approx \mathbf{X}$ and matrix $\mathbf{A}$, and produces updated SVD factorization $\mathbf{U}_{new}\mathbf{\Sigma}_{new}\mathbf{V}_{new}^T \approx [\mathbf{X}, \mathbf{A}]$. Note that in this formulation $\mathbf{V}^T$ gets expanded to have the same number of columns as $[\mathbf{X}, \mathbf{A}]$ whereas $\mathbf{U}$ and $\mathbf{\Sigma}$ only get updated but don't change in size. Analogous algorithm can be used to update $\mathbf{U}$: using the fact that $\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T = \mathbf{X}^T$ we get that $\mathbf{V}_{new}\mathbf{\Sigma}_{new}\mathbf{U}_{new}^T \approx [\mathbf{X}^T, \mathbf{A}^T]$.

We can readily apply these ideas to CF and factorize users incrementally achieving considerable savings in space requirements. Formally, we partition users into blocks of size N_b and for each block we iteratively (1) calculate $N_b \times M$ neighbor score matrix and (2) update SVD factorization to include latent factors for users in the block. The runtime complexity of this block-factorization algorithm is still $O(NM\log(r) + (N+M)r^2)$ for $r < \sqrt{\min(N,M)}$ [11, 7], but the space requirement reduces to $O(N_bM + (N+M)r))$ where $O(N_bM)$ is the block size and $O((N+M)r)$ is the space required to store the final $\mathbf{U}_r$ and $\mathbf{V}_r$ factors. This is a very significant reduction from the original $O(NM)$ and makes the algorithm practical for most CF applications. Moreover, block update provides an effective way to update the SVD model when new users and/or items are added to the system. Finally, note that we concentrated on incrementally processing users here but the same algorithm can be straightforwardly modified to incrementally process items instead. Thus depending on whether $N \gg M$ or $N \ll M$ we can either choose user oriented or item oriented approach (or alternate between the two).

4.2 Sparse Similarity

In the previous section we demonstrated that by using incremental block updates we can significantly reduce the space complexity to $O(N_bM)$ where N_b is the block size. However, for large scale applications the number of items can reach millions or hundreds of millions so even $O(N_bM)$ can become too large. To deal with this problem we propose to selectively use only a small subset of similarity scores for each user. This is achieved by storing only the largest ρ scores for each user and zeroing out the rest.

The motivation behind using only the largest scores comes from the fact that we want to concentrate all model effort on maximizing the accuracy at the top of the ranking. Items with large scores are thus particularly important since they would appear at the top for each user. Furthermore, it is well known that SVD produces factorizations that are closest to the target matrix in Frobenius norm (i.e., root squared error). By zeroing out low scores we force SVD to concentrate

[4] This excludes the cost of computing $\mathbf{S}$.

Algorithm 1 SVD block-factorization

Input: $\mathbf{R}$
Parameters: rank r, block size N_b, sparsity factor ρ
for $i = 1$ **to** N/N_b **do**
  init block score matrix $\mathbf{S}_b = \text{sparse}(N_b, M)$
  for $u = (i-1)N_b + 1$ **to** iN_b **do**
    calculate top-ρ scores for u using Eq. 1 or 2
    store top-ρ scores in $\mathbf{S}_b(u,:)$
  end for
  if $i == 1$ **then**
    compute initial factorization:
      $[\mathbf{U}_r, \mathbf{S}_r, \mathbf{V}_r] = \text{SVD}(\mathbf{S}_b, r)$
  else
    update current factorization:
      $[\mathbf{U}_r, \mathbf{S}_r, \mathbf{V}_r] = \text{block-SVD}(\mathbf{U}_r, \mathbf{S}_r, \mathbf{V}_r, \mathbf{S}_b)$
  end if
end for
Output: $\mathbf{U}_r, \mathbf{S}_r, \mathbf{V}_r$

on ordering items at the top correctly and put less emphasis on relative orderings at the bottom.

This approach allows us to further reduce the space requirement for each block to $O(N_b\rho)$ since sparse format can now be used and zeros don't need to be explicitly stored. Sparse representation also allows for more efficient matrix multiplication and the complete block-factorization can now be computed in $O(T_{mult}r + (N+M)r^2)$ where T_{mult} is the cost of matrix-vector multiplication with sparse score matrix [11]. Given that the entire score matrix now has at most $N\rho$ non-zero entries, the matrix-vector product can be computed in time proportional to $N\rho$, and for $\rho \ll M$ we get that $T_{mult} \ll NM$ [25]. Note that $O(T_{mult}r + (N+M)r^2)$ is an order of magnitude more efficient than WRMF with complexity $O(r^2\mathcal{N} + (N+M)r^3)$. The full SVD algorithm with all the enhancements is outlined in Algorithm 1.

4.3 Strong Generalization

An important problem in CF is to efficiently provide accurate recommendations to new users that were not available during model optimization. This problem is often referred to as *strong generalization* and must be addressed by any successful recommender system. Neighbor-based models can generally be applied to new users without any difficulties since similarities can be straightforwardly re-calculated to incorporate new data. Model-based approaches on the other hand, often use complicated non-convex objectives and thus require expensive gradient updates to be conducted for every new user. In this section we address the strong generalization problem in our SVD model and derive a simple update equation to efficiently produce latent factors for new users.

One way of dealing with new users/items is to run a full block SVD update to generate new latent factors and also update the rest of the model. While in a real-world application we would want run such an update after enough new users/items have been added, it is infeasible to run it for every new user. To deal with this problem we propose a simple approach to approximate latent factors by utilizing the properties of SVD factorization. First, note that the score vector for every user u is approximated by the user-item factor product:

$$\mathbf{S}(u,:) \approx \mathbf{U}_r(u,:)\mathbf{V}_r^T$$

where $\mathbf{\Sigma}_r$ is absorbed into $\mathbf{U}_r$. From this it follows that:

$$\mathbf{U}_r(u,:) \approx \mathbf{S}(u,:)\mathbf{V}_r^{T(-1)}$$

but since $\mathbf{V}_r$ is approximately orthogonal we get that $\mathbf{V}_r^{-1} \approx \mathbf{V}_r^T$ and:

$$\mathbf{U}_r(u,:) \approx \mathbf{S}(u,:)\mathbf{V}_r \tag{9}$$

Similar approach can be used to derive an equation for item factors: $\mathbf{V}_r(v,:) \approx \mathbf{S}(:,v)^T\mathbf{U}_r$. Equation 9 gives us a simple and effective way to approximate user factors for a new user u by (1) calculating top-ρ neighbor scores $\mathbf{S}(u,:)$ and (2) multiplying $\mathbf{S}(u,:)$ with $\mathbf{V}_r$ to get $\mathbf{U}_r(u,:)$.

In production-level systems we can use Equation 9 to quickly generate recommendations for each new user, then run full block update once enough new users/items have been added. Note that unlike gradient-based models, block update doesn't require any iterative optimization or parameter tuning (initialization, learning rate, weight penalty etc.) and can efficiently update the entire model.

In summary, our SVD-based model provides the following advantages:

- Low complexity and storage requirement block update that can efficiently update the entire model.
- Only two parameters to tune: rank r and sparsity factor ρ (block size N_b is typically selected to maximize RAM usage).
- Existing distributed implementations (e.g., Mahout's stochastic SVD [17]) allow this model to easily scale to very large CF problems.
- Approximate procedure to quickly incorporate new users/items into existing model with one matrix multiplication.

4.4 Relation to Existing Work

The idea of using SVD in CF is not new and a number of SVD-based methods have been proposed. One of the first uses of SVD for CF can be traced back to [6]. A number of extensions have since been proposed that incorporate additional information and control for user privacy [22, 19]. Incremental SVD updates in the context of CF have been investigated by Sarwar et al., [24], their approach however only produced latent factors for new users and did not update the existing model. This method is thus unsuitable for incrementally learning/updating the full model.

To the best of our knowledge all existing SVD methods apply factorization to the raw preference matrix $\mathbf{R}$ and are thus not suitable for binary feedback. Moreover, since SVD can't handle missing data, applying it directly to $\mathbf{R}$ is equivalent to setting all missing values to 0. The factorization then minimizes the Frobenius norm between this "filled in" matrix and the SVD approximation. Setting missing values to 0 is not ideal in most CF applications since it assumes that *all* unobserved items should have a low score. For this reason SVD has now largely been replaced by other matrix factorization approaches such as PMF [21] that only optimize the reconstruction error on observed data. Our approach avoids this problem by applying SVD to the sparse neighbor similarity score matrix instead of $\mathbf{R}$. In our model setting small scores to 0 is valid since these items are to be ranked at the bottom of the list and should have low reconstruction scores

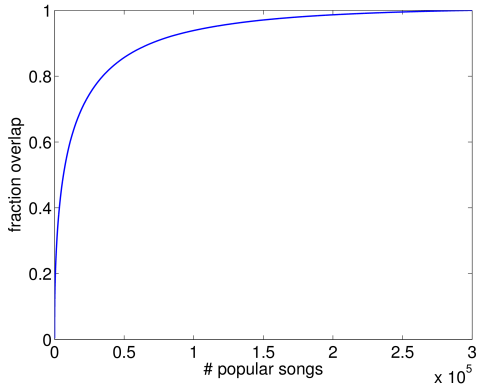

Figure 1: Fraction overlap between training data for the target 110K MSD users and popular songs sorted in the order of popularity.

from SVD. To the best of our knowledge this approach is the first successful application of SVD to the binary CF setting.

4.5 Alternative Formulations

The idea of using neighbor similarity information to train factor models is not limited to the SVD model. By replacing the original binary matrix $\mathbf{R}$ with $\mathbf{S}$ (Equations 1 and 2) we can apply most existing model-based approaches. Below we give some examples of the objective functions that can be used to learn $\mathbf{U}$ and $\mathbf{V}$. We begin with the more commonly used regression objectives such as the squared error and weighted squared error (used in WRMF):

$$\textbf{sq. error:} \quad \sum_{n=1}^{N}\sum_{m=1}^{M}(\mathbf{S}(u_n,v_m)-\mathbf{U}_r(u_n,:)\mathbf{V}_r(v_m,:)^T)^2$$

$$\textbf{wrmf:} \quad \sum_{n=1}^{N}\sum_{m=1}^{M}c_{nm}(\mathbf{R}(u_n,v_m)-\mathbf{U}_r(u_n,:)\mathbf{V}_r(v_m,:)^T)^2$$
$$c_{nm}=1+\alpha\mathbf{S}(u_n,v_m)$$

Note that in both cases $\mathbf{S}$ is a sparse matrix containing only the top-ρ scores for each user. Unlike the traditional CF where most effort is concentrated on rating prediction, for binary preference CF the goal is to optimize the ranking accuracy. Ranking-based objectives might thus be better suited for this problem. Since $\mathbf{S}$ provides rich ordering information over the items, we can apply virtually any objective from learning-to-rank [16] to this domain. Here we give examples of two such objectives RankNet [8] (also used in BPR-MF) and ListNet [9]:

$$\textbf{ranknet:} \quad \sum_{(u_n,v_m,v_l)}\log(1+e^{-\mathbf{U}_r(u_n,:)(\mathbf{V}_r(v_m,:)-\mathbf{V}_r(v_l,:))^T})$$
$$\text{for} \quad \mathbf{S}(u_n,v_m)>\mathbf{S}(u_n,v_l)$$

listnet:

$$-\sum_{n=1}^{N}\sum_{m=1}^{M}\frac{\phi(\mathbf{S}(u_n,v_m))}{\sum_{l=1}^{M}\phi(\mathbf{S}(u_n,v_l))}\log\left(\frac{\phi(\mathbf{U}_r(u_n,:)\mathbf{V}_r(v_m,:)^T)}{\sum_{l=1}^{M}\phi(\mathbf{U}_r(u_n,:)\mathbf{V}_r(v_l,:)^T)}\right)$$

All of the above objectives require gradient optimization and thus lack the advantages of the SVD approach outlined in Section 4.3. In this work we thus chose to concentrate on the SVD model and leave gradient optimization for future work.

Table 1: MSD MAP@500 private leaderboard results. SVD results are reported for four different rank sizes 1K, 5K, 10K, and 20K; all models were trained on top 50,000 and 100,000 of the most popular songs. The winning team achieved a score of 0.1791 [3] using an optimized blend of user-user and item-item approaches. The best model-based approach was reported to get 0.1095.

	1K	5K	10K	20K
50,000 most popular songs				
item-item (0.1498)				
ii-SVD-1K	0.1249	0.1466	0.1492	0.1493
ii-SVD-5K	**0.1261**	**0.1472**	**0.1497**	**0.1494**
ii-SVD-full	0.1258	0.1470	0.1495	**0.1494**
100,000 most popular songs				
item-item (0.1628)				
ii-SVD-1K	0.1258	0.1539	0.1598	0.1622
ii-SVD-5K	**0.1274**	**0.1550**	**0.1606**	0.1621
ii-SVD-full	0.1272	0.1548	0.1604	**0.1623**

5. EXPERIMENTS

To validate the proposed approach we conducted extensive experiments on three large publicly available datasets: song dataset from the Kaggle's MSD challenge and two movie datasets MovieLens and Netflix. We implemented Algorithm 1 in Matlab, employing stochastic SVD library by Mark Tygert [5] and block SVD library by David Wingate [6]. For block SVD we modified the provided code by replacing the call to Matlab's `svds` routine with a much faster stochastic SVD call. This implementation was then used in all experiments. We used MyMediaLite [10] library to run the WRMF and BPR-MF baselines and extensively tuned both models on each dataset.

Across all datasets we consistently found that the accuracy of user-user and item-item approaches can be significantly improved by applying both row and column normalizations (Equations 3 and 4). We also found that the order in which these normalizations are applied is important and used validation set to determine the best combination for each dataset. Generally, we observed that applying row followed by column normalization worked best across most datasets and folds. For all SVD experiments we first validated normalization for each neighbor approach and then applied SVD to the best performing combination.

5.1 MSD Experiments

The MSD dataset was used in the Kaggle Million Song Dataset Challenge [18] and consists of listening histories for 1.2M users and 380K songs. The goal of the challenge was to use these listening histories to recommend 500 songs for a subset of 110K test users. The data for the 110K test users was partitioned into training and test sets, and only training portion was made available to the challenge participants (in addition to full listening history for the other 1.1M users). The test set was further split into two subsets with 10K and 100K users respectively. The results on the smaller 10K subset were made visible throughout the challenge ("public leaderboard"), while results on the 100K

[5] `pca.m` from `cims.nyu.edu/~tygert/software.html`
[6] `addblock_svd_update.m` from `www.mit.edu/~wingated/resources.html`

Table 2: MovieLens weak generalization results. uu-SVD and ii-SVD are SVD factorizations computed using scores from user-user and item-item neighbor approaches respectively. Number of non-zero scores ρ used to calculate each SVD factorization is shown on the right, for instance ii-SVD-100 uses 100 scores for each user and ii-SVD-full uses all M scores.

	NDCG@10 binary				NDCG@10 rating			
	10%	30%	50%	70%	10%	30%	50%	70%
user-user	0.5416	0.5324	0.4660	0.3710	**0.6354**	0.6512	0.6765	0.7227
uu-SVD-100	0.5503	0.5468	0.4690	0.3700	0.6288	0.6508	0.6751	0.7205
uu-SVD-500	0.5545	0.5434	0.4743	0.3742	0.6309	**0.6515**	**0.6769**	**0.7230**
uu-SVD-full	0.5513	0.5452	0.4744	0.3757	0.6316	0.6513	0.6768	0.7229
item-item	0.4910	0.5957	0.5330	0.4321	0.5903	0.6223	0.6564	0.7108
ii-SVD-100	0.5504	0.5910	0.5287	0.4271	0.6092	0.6243	0.6537	0.7076
ii-SVD-500	**0.5672**	**0.6113**	0.5514	0.4434	0.6107	0.6293	0.6604	0.7124
ii-SVD-full	0.5484	0.5982	0.5523	0.4479	0.6121	0.6259	0.6562	0.7115
WRMF	0.5570	0.5969	**0.5557**	**0.4857**	0.6046	0.6280	0.6511	0.7071
BPR-MF	0.4428	0.5238	0.4976	0.4213	0.6190	0.6223	0.6585	0.7131

subset were only revealed at the end of the challenge ("private leaderboard"). All submissions had to provide rankings of top-500 songs for each of the 110K test users and were evaluated using MAP@500. At the end of the challenge organizers released all the challenge data (including test data) into public domain.

Note that the MSD dataset has more than 20 times more items than both MovieLens and Netflix, and is over 100 times more sparse. This makes the problem very challenging since many user/items have very little data to build accurate representations. The sparsity problem is further illustrated in Figure 1, which shows fraction overlap between the training data for the 110K target users and most popular songs sorted in the order of popularity. From the figure we see that more that 93% of all the training data is contained within the first 100K most popular songs, leaving less than 7% of data for the remaining 280K songs.

We mentioned above that model-based approaches were found to perform poorly in this challenge. The best model-based submission was reported to get MAP@500 of 0.1095, while the winning solution achieved 0.1791 using a blend of user-user and item-item approaches [3]. These results indicate that neighbor methods produce over 60% relative improvement in accuracy on this data compared to model-based approaches. In the following section we show that our approach eliminates this performance gap achieving results that are comparable to neighbor methods.

In all MSD experiments we follow the same set-up that was used during the competition and tune models on the 10K public leaderboard set, then evaluate on the 100K private leaderboard set.

5.1.1 Results

MAP@500 private leaderboard results are shown in Table 1. For this dataset we found that using larger rank generally improved performance, and we report results for four different rank sizes: 1K, 5K, 10K and 20K. To reduce experiment complexity we downsampled the item space to only include most popular songs and experimented with top 50,000 and 100,000 songs. From Figure 1 we see that 100,000 (50,000) most popular songs contain over 93% (80%) of all training data. Consequently, selecting only the popular songs allows us to reduce the item space by a factor of 4 while keeping most of the training data.

Table 3: MSD training runtimes in hours for the SVD model with four different rank sizes 1K, 5K, 10K and 20K. For comparison, on the same hardware WRMF took 10.9 hours to complete for rank 250 and would take over a month for rank 10K.

	1K	5K	10K	20K
50,000 most popular songs				
ii-SVD-1K	0.9	1.9	4.6	17.9
100,000 most popular songs				
ii-SVD-1K	1.8	2.8	5.6	24.6

From results in Table 1 we see that the SVD model with large enough rank is able to match the performance of the corresponding item-item approach. The results for the best SVD model place it in top-3 (out of 150 teams) on Kaggle's leaderboard with a score of 0.1623. To the best of our knowledge this is by far the highest score for model-based approach in this competition. All of the top-10 teams used combinations of user-user and/or item-item approaches, and the best model-based approach was reported to only get 0.1095. We also see that SVD-1K performs comparably to SVD-full on both 50,000 and 100,000 songs suggesting that scores for only 1,000 songs per user are required to produce accurate factorizations.

5.1.2 Runtime

MSD is the largest of the three datasets that we selected with, and we use it to benchmark the runtimes for the SVD approach and the best existing model-based approach WRMF. To ensure accurate comparison all experiments were conducted on the same server with 32 Intel Xeon E5-2690 2.90GHz cores and 64GB of RAM. Runtimes in hours for the SVD model are shown in Table 3. From this table we see that full factorization with 100,000 songs can be completed in under 3 hours with rank 5,000 and under 6 hours with rank 10,000. For comparison, WRMF took over 10 hours to complete with rank 250, and using complexity bounds we estimate that for rank 10,000 it would take over a month. These results demonstrate that our approach is considerably more efficient than the existing state-of-the-art and scales well to large datasets.

Table 4: MovieLens strong generalization results. 1000 randomly chosen users were withheld during model optimization and then added into the model either via full model block update (SVD-block) or via inference procedure outlined in Section 4.3 (SVD-infer); ρ was set to 500 across all training splits. Item-item baseline was computed using all users and item-item\1000 was computed with 1000 test users removed from the training data and excluded for item similarity calculation.

	NDCG@10 binary				NDCG@10 rating			
	10%	30%	50%	70%	10%	30%	50%	70%
item-item	0.5276	0.5985	0.5320	0.4298	0.6076	0.6352	0.6730	0.7199
item-item\1000	0.5101	0.5802	0.5112	0.4119	0.6044	0.6327	0.6688	0.7167
ii-SVD-block	**0.5837**	**0.6144**	**0.5539**	**0.4414**	**0.6273**	**0.6443**	**0.6754**	**0.7217**
ii-SVD-infer	0.5533	0.6043	0.5387	0.4264	0.6214	0.6411	0.6722	0.7200

5.2 MovieLens

For MovieLens experiments we use the largest of the available datasets MovieLens-10M with 10 million ratings from 72,000 users on 10,000 movies. This dataset contains ratings on the scale of 1 to 5 with half point increments, and each users has rated at least 20 movies. To simulate binary feedback we binarized the data setting all ratings to 1. Binarization simulates implicit feedback that we would get from actions like movie purchases, plays or clicks.

To get an estimate of how each model's performance is affected by training set size we repeat all experiments 4 times with different percentages of training ratings: 10%, 30%, 50%, 70%. For each percentage θ, θ percent of ratings for each user are randomly selected for training and the rest are kept for testing. Partitioning data in this way allows us to gradually increase the training set size from 10% to 70% while keeping distributions of ratings across users consistent. All parameter selection was done using 5-fold cross validation and the best models were re-trained on the full training set and evaluated on the test set. Rank size was validated in the range 5 to 250 for each model.

We evaluate both weak and strong generalization. For weak generalization training data from all users is used to learn the models, and average NDCG is computed on the withheld test data from same users (see Section 2 for details on NDCG calculation). On the other hand, for strong generalization a subset of users is removed from training data and all model estimation is done on the remaining users. Once the models are estimated, the goal is to use the training data from withheld users to incorporate them into the model (without full re-training) and produce accurate recommendations. In SVD model we proposed two ways of dealing with new users: block update and approximate inference (see Section 4.3). We evaluate both approaches and compare their performance.

Taking advantage of the fact that we have access to explicit preferences in the form of ratings we conduct two types of evaluation. After training each model using only binary data, we evaluate the recommended rankings on both binary relevance and full ratings. For binary relevance we use the procedure outlined in Section 2 and compute NDCG using full rankings of all the items that are not in the training set for each user. For rating evaluation we only rank the rated test set items for each user and calculate NDCG using rating values (1-5) as relevance for those items. Assuming that ratings provide the ground truth preference information, rating NDCG estimates the degree to which models trained on weaker binary data generalize to ground truth. While this assumption might not be completely valid due to inherent noise in ratings, rating NDCG still gives us a more accurate metric to compare the models.

5.2.1 Weak Generalization

Table 2 shows MovieLens weak generalization results for four training set sizes 10%, 30%, 50% and 70%. For each training set size, Table 2 shows binary and rating NDCG@10 results. We apply our SVD method to both user-user (uu-SVD-ρ) and item-item (ii-SVD-ρ) scores and vary the sparsity factor ρ of the similarity score matrix. In addition to SVD, we also evaluate WRMF and BPR-MF approaches, all models were learned and tuned using binary data only.

We can observe several patterns from Table 2. First, item-item approach significantly outperforms user-user on binary NDCG for all training set sizes except 10%, while user-user significantly outperforms item-item on rating NDCG. The difference in performance is somewhat puzzling and implies that improvement on binary feedback doesn't necessarily translate to improvement on explicit preferences. This suggests an optimization approach where explicit feedback is first solicited from a subset of users and the best binary model is then chosen based on the accuracy on this explicit data.

Second, SVD model always performs at least as well as the underlying similarity approach (user-user or item-item) and in many cases beats it. This leads to the conclusion that applying low-rank compression to similarity scores is beneficial and can improve accuracy by removing/shrinking noisy dimensions. The gains from SVD model are especially significant on very sparse training sets with 10% and 30% of data where it outperforms neighbor approaches on both binary and rating NDCG. We also see that SVD-500 perform almost as well as SVD-full indicating that only 500 top scores (out of nearly 10,000 items) are required for each user to produce accurate factorizations. This reduces storage requirement for each block update by a factor of 20 and significantly speeds up runtime. Consistent performance on both binary and rating NDCG suggests that the SVD model can readily be used if explicit data approach to model selection is chosen. In this setting, explicit preferences from a subset of users would first be used to select the best neighbor method. SVD model would then be applied to the chosen neighbor method to generate latent factors for all users and items.

Third, similarly to neighbor approaches, WRMF outperforms BPR-MF on binary NDCG while BPR-MF significantly outperforms WRMF on rating NDCG. We also see that best SVD model consistently outperforms both approaches on binary and rating NDCG.

Table 5: Netflix weak generalization results. uu-SVD and ii-SVD are SVD factorizations computed using scores form user-user and item-item neighbor approaches respectively. Number of non-zero scores ρ used to calculate each SVD factorization is shown on the right, for instance SVD-500 uses 500 scores for each user and SVD-full uses all M scores.

	NDCG@10 binary				NDCG@10 rating			
	10%	30%	50%	70%	10%	30%	50%	70%
user-user	0.4717	0.4456	0.3844	0.2995	0.6226	0.6287	0.6489	0.6846
uu-SVD-500	0.4888	0.4602	0.3889	0.3005	0.6226	0.6299	0.6491	0.6845
uu-SVD-1000	0.4889	0.4609	0.3924	0.3043	0.6212	0.6303	0.6493	0.6850
uu-SVD-full	0.4846	0.4608	0.3939	0.3044	0.6196	0.6299	0.6498	0.6847
item-item	0.4980	0.4840	0.4297	0.3394	0.6287	0.6518	0.6698	0.7009
ii-SVD-500	**0.5172**	0.5164	0.4502	0.3487	0.6275	**0.6544**	0.6703	0.7008
ii-SVD-1000	0.5108	0.5200	0.4521	0.3500	**0.6299**	0.6519	**0.6715**	**0.7017**
ii-SVD-full	0.4994	0.5127	0.4521	0.3541	0.6203	0.6501	0.6708	0.7014
WRMF	0.5060	**0.5485**	**0.5072**	**0.4522**	0.6077	0.6376	0.6632	0.7000
BPR-MF	0.4273	0.4744	0.4447	0.3660	0.5985	0.6309	0.6606	0.6930

5.2.2 Strong Generalization

Strong generalization results are shown in Table 4. For strong generalization 1000 randomly chosen users were withheld during model optimization and then added into the model either by doing a full model block update (SVD-block) or via inference procedure outlined in Section 4.3 (SVD-infer). For all SVD experiments ρ was set to 500 so top-500 scores were used for each user. We only show results for item-item similarity, results with user-user similarity were similar and are omitted from this table.

From the table we see that SVD-infer generalizes well and suffers only a small drop in performance relative to block-SVD which updates the entire model. Both SVD-block and SVD-infer significantly outperform the item-item approach on binary and rating NDCG. These results support the conclusion that the approximate inference procedure can be used to quickly generate recommendations for new users with acceptable accuracy.

5.3 Netflix Experiments

For Netflix experiments we used the full Netflix dataset with 100M ratings from 480K users on 17.7K movies. Similar to the MovieLens experiments, we binarized all ratings and evaluated performance using both binary and rating NDCG@10. We partitioned the dataset using 10%, 30%, 50% and 70% of ratings from each user for training and the rest for testing. All parameter selection was done using 5-fold cross validation and the best models were re-trained on the full training set and evaluated on the test set. Rank size was validated in the range 5 to 250 for each model.

5.3.1 Weak Generalization

Weak generalization results are shown in Table 5. From this table we see that, unlike the results for the MovieLens data, item-item approach significantly outperforms user-user on both binary and rating NDCG. This could be attributed to the fact that Netflix dataset is considerably larger and more sparse. A number of previous studies have shown that item-item similarity tends to work better than user-user in sparse settings [12].

From the table we also see that, first, our SVD approach either performs comparably or outperforms both user-user and item-item methods on binary and rating NDCG. Second, WRMF achieves very strong performance on binary NDCG beating our approach on all splits except 10% but under-performs on rating NDCG losing by as much as 2 points. This difference in performance could be attributed to overfitting. We discussed in Section 4 that WRMF and BPR-MF treat all items with observed preferences ($\mathbf{R}(u, v) = 1$) as equally relevant. Both methods are thus susceptible to outliers that are often present in noisy CF data and can produce suboptimal user representations. In this experiment outliers correspond to items with low ratings (≤ 2). Placing such items at the top of the ranking improves binary NDCG while significantly hurting rating NDCG. Neighbor methods are able to detect such outliers and thus tend to do better on rating NDCG.

Lastly, both SVD-500 and SVD-1000 fully match the performance SVD-full leading to a conclusion that only 500 items (out of over 17,000) are needed to build accurate models for each user.

5.3.2 Strong Generalization

Strong generalization results for the Netflix dataset are shown in Table 6. Similarly to MovieLens, we withheld a subset of 1000 randomly chosen users during main model optimization, and then added these users to the model via block update and approximate inference procedures. Binary and rating NDCG results for both procedures are shown in Table 6.

From the table we see that SVD-infer suffers only a minor drop in performance relative to SVD-block on rating NDCG. Also, surprisingly, SVD-infer outperforms SVD-block on binary NDCG for 50% and 70% training splits. These results further validate the approximate inference procedure as a useful way to quickly generate recommendations for new users.

6. CONCLUSION AND FUTURE WORK

We presented a general approach to deal with binary preferences in CF. In our approach, observed binary matrix is first transformed into a score matrix by applying neighborhood similarity rescaling. The score matrix is then factorized to produce accurate user and item representations. We demonstrated that with this approach even simple factorization techniques like SVD produce accurate representations using only a small subset of the highest scores for each user.

Table 6: Netflix strong generalization results. 1000 randomly chosen users were withheld during model optimization and then added to the model either via full model block update (SVD-block) or via inference procedure outlined in Section 4.3 (SVD-infer); ρ was set to 500 across all training splits. Item-item baseline was computed using all users and item-item\1000 was computed with 1000 test users removed from the training data and exclude for item similarity calculation.

	NDCG@10 binary				NDCG@10 rating			
	10%	30%	50%	70%	10%	30%	50%	70%
item-item	0.4441	0.4754	0.4227	0.3364	**0.6380**	**0.6583**	0.6734	**0.7074**
item-item\1000	0.4484	0.4922	0.4391	0.3492	0.6224	0.6492	0.6688	0.7045
ii-SVD-block	**0.4897**	0.5113	0.4441	0.3449	0.6277	0.6568	**0.6741**	**0.7074**
ii-SVD-infer	0.4736	**0.5117**	**0.4624**	**0.3645**	0.6190	0.6498	0.6718	0.7044

In the future work we plan to explore more complex factorization procedures that can potentially lead to better latent representations. We also plan to investigate ways to deal with discrepancy between model performance on implicit/binary and explicit data. We have shown that improvement on implicit/binary data doesn't always lead to improvement on explicit preferences. We believe that this discrepancy must be considered in every approach designed for binary data. Finally, we plan to extend our approach beyond the binary setting and apply it to graded preferences (ratings).

Acknowledgments

We would like to thank Richard Zemel and Tomi Poutanen for many useful ideas and discussions. We also thank the anonymous reviewers for their comments and suggestions.

7. REFERENCES

[1] P.-A. Absil, C. G. Baker, and K. A. Gallivan. Trust-region methods on riemannian manifolds. *Foundations of Computational Mathematics*, 7(3), 2007.

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of- the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6), 2005.

[3] F. Aiolli. Efficient top-n recommendation for very large scale binary rated datasets. In *ACM Recommender Systems*, 2013.

[4] R. Baeza-Yates and B. Ribeiro-Neto. *Information Retrieval*. Addison-Wesley, 1999.

[5] J. Bennet and S. Lanning. The Netflix prize. `www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf`.

[6] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *International Conference on Machine Learning*, 1998.

[7] M. Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1), 2006.

[8] C. J. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *International Conference on Machine Learning*, 2005.

[9] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: From pairwise approach to listwise approach. In *International Conference on Machine Learning*, 2007.

[10] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. MyMediaLite: A free recommender system library. In *ACM Recommender Systems*, 2011.

[11] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2), 2011.

[12] J. Herlocker, J. A. Konstan, and J. Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Information Retrieval*, 5(4), 2002.

[13] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *International Conference on Data Engineering*, 2008.

[14] Z. Huang, D. Zeng, and H. Chen. A comparison of collaborative-filtering recommendation algorithms for e-commerce. *IEEE Intelligent Systems*, 22(5), 2007.

[15] K. Jarvelin and J. Kekalainen. IR evaluation methods for retrieving highly relevant documents. In *ACM Special Interest Group on Information Retrieval*, 2000.

[16] T.-Y. Liu. *Learning to rank for information retrieval*. Springer, 2011.

[17] Apache Software Foundation. Mahout stochastic singular value decomposition. `https://mahout.apache.org/users/dim-reduction/dimensional-reduction.html`.

[18] B. McFee, T. Bertin-Mahieux, D. P. Ellis, and G. R. Lanckriet. The Million Song Dataset Challenge. `http://www.kaggle.com/c/msdchallenge`. In *International World Wide Web Conference*, 2012.

[19] H. Polat and W. Du. SVD-based collaborative filtering with privacy. In *ACM Symposium On Applied Computing*, 2005.

[20] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Conference on Uncertainty in Artificial Intelligence*, 2009.

[21] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Neural Information Processing Systems*, 2008.

[22] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system - a case study. Technical report, DTIC Document, 2000.

[23] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *International World Wide Web Conference*, 2001.

[24] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *International Conference on Information Systems*, 2002.

[25] R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Transactions on Algorithms*, 1(1), 2005.