Practical Linear Models for Large-Scale One-Class Collaborative Filtering

Anonymous

Abstract

Collaborative filtering has emerged as the de facto approach to personalized recommendation problems. However, a scenario that has proven difficult in practice is the implicit feedback or one-class collaborative filtering case (OC-CF), where one has examples of items that a user prefers, but no examples of items they do not prefer. In such cases, it is desirable to have recommendation algorithms that are personalized, learning-based, and highly scalable. Existing linear recommenders for OC-CF achieve good performance in benchmarking tasks, but they involve solving a large number of a regression subproblems, which limits their applicability to large-scale problems. We demonstrate that it is possible to scale up linear recommenders to big data by learning an OC-CF model in a randomized low-dimensional embedding of the user-item interaction matrix. Our algorithm, Linear-Flow, achieves state-of-the-art performance in a comprehensive set of experiments on standard benchmarks as well as real data.

1 Introduction

Personalised recommendation system is a core component in many modern e-commerce services. Collaborative filtering (CF) is the de-facto standard approach to making recommendation, based on information in a database of item preferences from a population of users. Most work on CF has considered the explicit feedback setting, where users express both positive and negative preferences for items in terms of ratings or likes/dislikes. In contrast, in the implicit feedback setting, we do not have explicit negative preference information. For example, consider recommending items to users of an e-commerce website based on their purchase history. One can assume that a purchase indicates a positive preference for an item. However, the lack of a purchase does not necessarily indicate a negative preference; it may just be that the user is unaware of an item. Such scenarios are also referred to as one-class collaborative filtering (OC-CF).

In designing a real world recommender systems there are various factors to be considered. First and foremost, a recommender system should produce good recommendation which can be quantified in terms of *relevance*. The relevance of a recommender system is measured using evaluation metrics such as precision@k, recall@k etc. Second, recommender systems should be highly *scalable*. In modern day applications, it is very common to have millions of users and items. A model should be able to handle the data as the number of user and item grows to this scale. *Similarity metric* is another important aspect of personalization. For instance, item-item similarity allows us to recommend similar items to the users. Recommending similar items is very prevalent in a real-world recommender systems. Hence, it is highly desirable to have a recommendation algorithm where the similarity is directly expressed and incorporated in the model. Also, *interpretability* of the recommendation is very critical in persuading users. By explaining the recommendations, the system becomes more transparent, build users' trust in the system and convince users in consuming the recommended items. Lack of interpretability weakens the ability to persuade users in decision making [15].

While there is a rich literature on OC-CF (discussed subsequently), to our knowledge, all existing methods lack one or more desiderata. Neighborhood-based methods are scalable, incorporates similarity metric in the model and gives explainable recommendations. However, the relevance of their recommendation is weaker compared to the linear counterparts. Matrix Factorization models are scalable but are also not competitive in terms of relevance. Also, the recommendations are not explainable and there is no notion of similarity in the model. On the other hand, Linear recommenders are state-of-the-art in terms of relevance. Furthermore, the model explicitly learns a similarity metric. Also, like neighborhood models, recommendations from linear model are easily explainable. However, current linear methods are computationally expensive which limits their applicability in large scale real world problems. Table 1 summarizes the strengths and weaknesses of ex-

Table 2: Commonly used symbols.

Symbol	Meaning	Symbol	Meaning
$\mathcal{U}$	Set of users	$\mathbf{R}$	Purchase matrix
$\mathcal{I}$	Set of items	$\hat{\mathbf{R}}$	Recommendation matrix
m	Number of users	$\mathcal{R}(u)$	Items purchased by u
n	Number of items	$\mathbf{S}$	Item similarity matrix

isting OC-CF methods.

In this paper we address the computational bottleneck of linear models, enabling them to scale up to large OC-CF problems without compromising performance. Our method, Linear-Flow (Fast Low Rank Linear Model), formulates OC-CF as a regularised linear regression problem with a randomized SVD for fast dimensionality reduction. Although regularized regression and randomized SVD are not new ideas, their combined use in the context of OC-CF is, to the best of our knowledge, novel. Through extensive experiments on known benchmarks and real-world datasets, we demonstrate that Linear-Flow achieves state-of-the-art performance as compared to other methods, with a significant reduction in computational cost. Due to its scalability and performance, Linear-Flow has all the desirable properties of a practical recommender system.

2 Background

We now introduce our notation, summarized in table 2. Let $\mathcal{U}$ denote a set of users, and $\mathcal{I}$ a set of items, with $m = |\mathcal{U}|$ and $n = |\mathcal{I}|$. In one-class collaborative filtering (OC-CF), we have a purchase[1] matrix $\mathbf{R} \in \{0,1\}^{m\times n}$. We use $\mathbf{R}_{ui}$ to refer to the purchase status for the user u and item i. We use $\mathbf{R}_{:i}$ to denote the indicator vector of each users' purchase of an item, and similarly $\mathbf{R}_{u:}$ to denote the vector of a user's preference for each item i. We denote by $\mathcal{R}(u)$ the set of the items purchased by user u. The goal in OC-CF is to learn a recommender, which for our purposes is simply a matrix $\hat{\mathbf{R}} \in \mathbb{R}^{m\times n}$. We call $\hat{\mathbf{R}}$ the *recommendation matrix*.

In the rest of this section we review some existing approaches to OC-CF.

2.1 Neighbourhood methods

In (item-based) neighbourhood methods, we produce a recommendation matrix of the form

$$\hat{\mathbf{R}} = \mathbf{RS} \tag{1}$$

where $\mathbf{S} \in \mathbb{R}^{n\times n}$ is some *item-similarity matrix*. Typically, one uses a predefined matrix $\mathbf{S}$ that relies on $\mathbf{R}$. A popular example is cosine similarity [11, 8],

$$\mathbf{S}_{i'i} = \frac{\mathbf{R}_{:i}^T \mathbf{R}_{:i'}}{||\mathbf{R}_{:i}||_2 ||\mathbf{R}_{:i'}||_2}.$$

[1]We use the word "purchase" simply for the purposes of exposition. The method described in this paper is applicable in other implicit feedback settings, such as modelling users' play counts of songs.

It is typical to sparsify $\mathbf{S}$ so that its columns only keeps top-k similar items. Neighbourhood methods are attractive for several reasons. They are simple to implement, efficient and interpretable. However, they are unable to adapt to the characteristics of the data as they rely on a fixed $\mathbf{S}$ that is not learned from data [6]. Furthermore, recommendation performance can be quite sensitive to the choice of $\mathbf{S}$.

2.2 Matrix Factorization

Matrix Factorization methods are the *de facto* approach to collaborative filtering with explicit feedback. The basic idea is to embed users and items into some shared latent space, with the aim of inferring complex preference profiles for both users and items. Weighted matrix factorization (WRMF) for one class collaborative filtering [5, 10] optimises

$$\min_{\mathbf{A},\mathbf{B}} \sum_{u\in\mathcal{U}, i\in\mathcal{I}} \mathbf{J}_{ui}(\mathbf{R}_{ui} - \mathbf{A}_u^T \mathbf{B}_i)^2 + \frac{\lambda}{2}(||\mathbf{A}||_F^2 + ||\mathbf{B}||_F^2) \tag{2}$$

where $\mathbf{J} \in \mathbb{R}_+^{m\times n}$ is some pre-defined weighting matrix and

$$\mathbf{J}_{ui} = [\![\mathbf{R}_{ui} = 0]\!] + \alpha[\![\mathbf{R}_{ui} > 0]\!] \tag{3}$$

where α assigns an importance weight to the observed interaction.

2.3 Linear Recommenders

Linear methods [9, 12] learn the similarity metric from the data. SLIM [9] views the recommendation as learning item-item similarity and learns an item-similarity matrix $\mathbf{W} \in \mathbb{R}^{n\times n}$ via

$$\min_{\mathbf{W}\in\mathcal{C}} ||\mathbf{R} - \mathbf{RW}||_F^2 + \frac{\lambda}{2}||\mathbf{W}||_F^2 + \mu||\mathbf{W}||_1, \tag{4}$$

where $\lambda, \mu > 0$ are appropriate constants, and

$$\mathcal{C} = \{\mathbf{W} \in \mathbb{R}^{n\times n} : \mathrm{diag}(\mathbf{W}) = 0, \mathbf{W} \geq 0\}. \tag{5}$$

Here, $||\cdot||_1$ denotes the elementwise ℓ_1 norm of $\mathbf{W}$ so as to encourage sparsity, and the constraint $\mathrm{diag}(\mathbf{W}) = 0$ prevents a trivial solution of $\mathbf{W} = \mathbf{I}_{n\times n}$. SLIM is equivalent to an item-based neighbourhood approach where the similarity matrix $\mathbf{S} = \mathbf{W}$ is *learned* from the data. In a related linear model for OC-CF [12], the authors report very good performance, but the proposed method is computationallly expensive, which restricts its applicability in real world problems.

Linear methods are attractive for several reasons. They have superior performance, and unlike neighborhood methods, they adapt with the data as the parameters are learned from data itself. Furthermore, the recommendations are easily interpretable. However, linear methods can be computationally expensive as they require solving a large number of regression subproblems with a big design matrix $\mathbf{R}$. This can scale quadratically with the order of $\mathbf{R}$, or worse.

Method	Relevance	Scalability	Similarity	Interpretability
Neighborhood	×	✓	✓	✓
MF	×	✓*	×	×
Linear	✓	×	✓	✓
Linear-Flow	✓	✓	✓	✓

Table 1: Comparison of recommendation methods for OC-CF. The * for MF is added because weighted MF, WRMF, is relatively expensive.

2.4 Randomized SVD

SVD is the archetypal matrix factorization algorithm and has been widely used in machine learning for dimensionality reduction. However, SVD is computationally expensive and not scalable to large scale datasets. It has been recently shown that SVD can be significantly scaled up, at a negligible cost in performance, by randomization [4]. We will describe the randomized SVD algorithm in the next section, in the context of the OC-CF problem.

Randomized SVD has been applied to matrix factorization [14] (but not in the OC-CF setting that we are considering). The authors compute the rank-k randomized SVD of the matrix $\mathbf{R}$,

$$\mathbf{R} \approx \mathbf{P}_k \mathbf{\Sigma}_k \mathbf{Q}_k^T$$

where, $\mathbf{P}_k \in \mathbb{R}^{m\times k}$, $\mathbf{Q}_k \in \mathbb{R}^{n\times k}$ and $\mathbf{\Sigma}_k \in \mathbb{R}^{k\times k}$. Given the truncated SVD solution, they initialize the item latent factor with the SVD solution and solve

$$\underset{\mathbf{A}}{\operatorname{argmin}} \left\|\mathbf{R} - \mathbf{A}^T\mathbf{B}\right\|_F^2 + \lambda \left\|\mathbf{A}\right\|_F^2 \tag{6}$$

$$s.t.\ \mathbf{B} = \mathbf{\Sigma}_k^{\frac{1}{2}} \mathbf{Q}_k^T \tag{7}$$

Similarly, if the matrix $\mathbf{A}$ is fixed instead of the matrix $\mathbf{B}$, the objective becomes

$$\underset{\mathbf{B}}{\operatorname{argmin}} \left\|\mathbf{R} - \mathbf{A}^T\mathbf{B}\right\|_F^2 + \lambda \left\|\mathbf{B}\right\|_F^2 \tag{8}$$

$$s.t.\ \mathbf{A} = \mathbf{P}_k \mathbf{\Sigma}_k^{\frac{1}{2}} \tag{9}$$

We refer to (8) and (6) as U-MF-RSVD and I-MF-RSVD respectively. As we will see in the results, the performance of these two models can vary significantly.

3 Large Scale Linear Methods for One Class Collaborative Filtering

Despite their superior performance, the applicability of Linear methods on real world large scale dataset are constrained by their computational cost. Linear methods involve solving a large number of regression subproblems on a huge design matrix $\mathbf{R}$ making it extremely challenging on real world applications where the number of users and items is in millions.

Here we propose a model and an algorithm for scaling up linear methods to large OC-CF problems. In particular, we are seeking an approximation $\mathbf{R} \approx \mathbf{RW}$ that attempts to capture most of the row space of the matrix $\mathbf{R}$ through a matrix $\mathbf{W}$ that is low-rank *and* has small Frobenius norm. The motivation for such a double regularization is to better control the generalization error of the model, an insight that was proven correct by our experiments. Moreover, it turns out that there is a very natural and efficient way to compute such an approximate decomposition of the matrix $\mathbf{R}$ by randomization [4], which allows scaling to large problems.

Our model amounts to solving the following optimization problem

$$\underset{\text{rank}(\mathbf{W})\leq k}{\operatorname{argmin}} \left\|\mathbf{R} - \mathbf{RW}\right\|_F^2 + \lambda \left\|\mathbf{W}\right\|_F^2 \tag{10}$$

where, typically, $k \ll n$. For $\lambda = 0$, the optimal solution is given by the Eckart-Young theorem (see, e.g., [4])

$$\mathbf{W} = \mathbf{Q}_k \mathbf{Q}_k^T \tag{11}$$

where $\mathbf{Q}_k$ is an orthogonal matrix computed by a truncated SVD

$$\mathbf{R} \approx \mathbf{P}_k \mathbf{\Sigma}_k \mathbf{Q}_k^T$$

Similarly, if we drop the low-rank constraint in (10), the optimal matrix $\mathbf{W}$ is given by the solution of a standard regression problem

$$\mathbf{W} = (\mathbf{R}^T\mathbf{R} + \lambda \mathbf{I})^{-1}\mathbf{R}^T\mathbf{R} \tag{12}$$

which involves the inverse of the original matrix $\mathbf{R}$ and therefore does not scale to large problems.

However, under both a low-rank constraint and $\lambda > 0$ in (10), finding the optimal $\mathbf{W}$ involves solving a hard nonconvex problem with no analytical solution in general. Nonetheless, an analytical solution is possible for a certain parametrization of $\mathbf{W}$ as we explain next. We first compute an approximate orthogonal basis $\mathbf{Q}_k$ of the row space of $\mathbf{R}$, i.e.,

$$\mathbf{R} \approx \mathbf{RQ}_k \mathbf{Q}_k^T \tag{13}$$

using randomized SVD. The randomized SVD algorithm is shown in Algorithm 1. (We refer to Halko et. al [4] for more details.) Then we reparametrize the matrix $\mathbf{W}$ as

$$\mathbf{W} = \mathbf{Q}_k \mathbf{Y} \tag{14}$$

for some matrix $\mathbf{Y}$. Note that through this parametrization the rank of $\mathbf{W}$ is automatically controlled, and the optimization problem (10) reads

$$\underset{\mathbf{Y}}{\operatorname{argmin}} \left\|\mathbf{R} - \mathbf{RQ}_k\mathbf{Y}\right\|_F^2 + \lambda \left\|\mathbf{Q}_k\mathbf{Y}\right\|_F^2 \tag{15}$$

Table 3: Summary of datasets used in evaluation.

Dataset	m	n	$\lvert\mathbf{R}_{ui} > 0\rvert$
ML10M	69,613	9,405	5,004,150
LASTFM	992	88,428	800,274
PROPRIETARY-1	26,928	14,399	120,268
PROPRIETARY-2	264,054	57,214	1,398,332

Since $\mathbf{Q}_k$ is orthogonal, we have $\|\mathbf{Q}_k\mathbf{Y}\|_F = \|\mathbf{Y}\|_F$, and (15) becomes

$$\underset{\mathbf{Y}}{\operatorname{argmin}} \|\mathbf{R} - \mathbf{R}\mathbf{Q}_k\mathbf{Y}\|_F^2 + \lambda \|\mathbf{Y}\|_F^2 \tag{16}$$

The latter can be solved analytically to give

$$\mathbf{Y} = (\mathbf{Q}_k^T\mathbf{R}^T\mathbf{R}\mathbf{Q}_k + \lambda I)^{-1}\mathbf{Q}_k^T\mathbf{R}^T\mathbf{R}$$

Note that this inversion involves a $k \times k$ matrix, and hence it is tractable.

We refer to (15) as I-Linear-Flow as it corresponds to item-item model. Similarly, we can define a user-user model, U-Linear-Flow

$$\underset{\mathbf{Y}}{\operatorname{argmin}} \|\mathbf{R} - \mathbf{R}\mathbf{P}_k\mathbf{Y}\|_F^2 + \lambda \|\mathbf{Y}\|_F^2 \tag{17}$$

As we discussed earlier, recommending similar items is very prevalent in real-world recommender systems. In I-Linear-Flow model, the item-item similarity is explicitly given by the matrix $\mathbf{W} = \mathbf{Q}_k\mathbf{Y}$.

Algorithm 1 Given $R \in \mathbb{R}^{m\times n}$, compute approximate rank-k SVD; R $\approx P_k\mathbf{\Sigma}_k Q_k$

1: **procedure** RSVD(R, k)
2: Draw $n \times k$ Gaussian random matrix Ω
3: Construct $n \times k$ sample matrix $A = R\Omega$
4: Construct $m \times k$ orthonormal matrix Z, such that $A = ZX$
5: Constuct $k \times n$ matrix $B = Z^T R$
6: Compute the SVD of B, B $= \hat{P}_k\mathbf{\Sigma}_k Q_k$
7: $R \Rightarrow ZB \Rightarrow Z\hat{P}_k\mathbf{\Sigma}_k Q_k \Rightarrow P_k\mathbf{\Sigma}_k Q_k, where\ P_k = Z\hat{P}_k$
8: return $P_k\mathbf{\Sigma}_k Q_k$
9: **end procedure**

4 Experiment and Evaluation

4.1 Dataset

We now report experimental results and compare the recommendation performance of the proposed method to competing methods on a number of datasets. We used two proprietary and two publicly available dataset for evaluating the performance. In all of our dataset, we remove the users and items with less than 3 purchases. Table 3 summarizes statistics of the datasets.

ML10M . The MovieLens 10M dataset[2] is a standard benchmark for collaborative filtering tasks. Following the "Who Rated What" KDD Cup 2007 challenge [1], we created a binarised version of the dataset suitable for evaluating implicit feedback methods. From the original rating matrix $\tilde{\mathbf{R}} \in \{0, 1, \ldots, 5\}^{m\times n}$, we created a preference matrix $\mathbf{R}$ with $R_{ui} = [\![\tilde{\mathbf{R}}_{ui} \geq 4]\!]$.

LASTFM . The LastFM dataset[3] [2] contains the play counts of ∼1000 users on ∼170,000 artists. As per ML10M , we binarised the raw play counts.

PROPRIETARY-2 & PROPRIETARY-1 are anonymized dataset provided by XXX[4], a major provider of third party recommendation services. PROPRIETARY-1 dataset consists of ∼27,000 users, ∼14,000 items and ∼120,000 item purchases. Similarity, PROPRIETARY-2 dataset consists of ∼264,000 users, ∼57,000 items and ∼1 billion item purchases.

For a qualitative analysis of the item-item similarities learned by I-Linear-Flow model, we use PROPRIETARY-3 dataset[5]. This dataset consists clicks of users on various image categories. We choose this dataset mainly because the items are easy to comprehend.

4.2 Evaluation Protocol

We split the datasets into random 90%-10% train-test set and hold out 10% of the training set for hyperparamater tuning. We report the mean test split performance, along with standard errors corresponding to 95% confidence intervals. To evaluate the performance of the various recommenders, we report precision@k and recall@k for $k \in \{3, 5, 10, 20\}$ (averaged across all test fold users), and mean average precision (mAP@20).

4.3 Methods compared

We compared the proposed method to a number of baselines:

- User- and item-based nearest neighbour (U-KNN and I-KNN), as per §2.1, using cosine similarity and Jaccard coefficient to define the similarity matrix $\mathbf{S}$. For each dataset, we picked the best performing of the two metrics.
- PureSVD of [3]. Instead of exact SVD, we use randomized SVD.
- Weighted matrix Factorization (WRMF) as defined in (2).
- MF-RSVD of [14]. We ran this method with user and item based initialization, U-MF-RSVD and I-MF-RSVD, as discussed in (8) and (6) respectively.
- SLIM, as per (4). For computational convenience, we used the SGDReg variant [7], which is identical to SLIM except that the nonnegativity constraint is removed. We did not evaluate SLIM with nonnegativity directly as [7] reports superior performance to SLIM, and is considerably faster to train.

[2] http://grouplens.org/datasets/movielens/

[3] http://ocelma.net/MusicRecommendationDataset/index.html

[4] anonymised for the blind review and will be revealed later

[5] Dataset sharing agreement with the provider restricts us from reporting the statistics and quantitative results. Hence, we wont be reporting the quantitative results on this dataset

Note that we do not compare against [12] due to its memory complexity on a large scale dataset. For instance, on PROPRIETARY-2 dataset LRec requires ∼260 GB of memory.

4.4 Performance Evaluation

Tables 4 – 7 summarize the results of the various methods. We make the following observations:

- Linear-Flow is generally competitive with SLIM and outperforms the other baselines on all four datasets. Unlike other methods, performance of Linear-Flow is not very sensitive to the choice of user or item based method. In figure 1, we observe that the performance of the Linear-Flow method improves with the number of the projection dimensions.
- Amongst the matrix Factorization methods, Randomized SVD method [14] consistently performs the best in all dataset. This is possibly due to the fact that SVD provides a good initialization and the optimization finds optimal solution for the given initialization. Whereas, matrix factorization optimizes nonconvex bilinear objective with random initialization which may lead to the inferior solution. Furthermore, we observe that the performance of MF-RSVD varies significantly based on whether we initialize user or item latent factors.
- We observe that the neighborhood models yield inferior results compared to the Linear models. Also, the performance varies with the user and item based models. While they perform comparatively competitive in PROPRIETARY-2 and PROPRIETARY-1 dataset, they perform poorly in LASTFM and ML10M dataset. Hence, we conclude that neighborhood methods are not consistent in their performance.

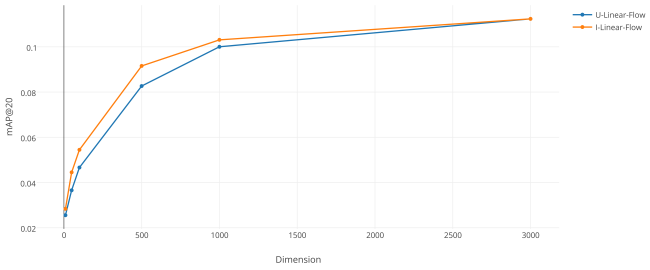

Figure 1: Performance of the U-Linear-Flow and I-Linear-Flow with the number of dimensions.

Since similarity metric is an important aspect of personalization, we evaluate item-item similarities learned by I-Linear-Flow model on the PROPRIETARY-3 dataset. In Table 8, we show top-3 similar items for a small subset of items learned by I-Linear-Flow model. We observe that I-Linear-Flow learns meaningful and explainable similarities making it very applicable in recommending similar items.

Table 9: Training time on the PROPRIETARY-2 and ML10M Dataset.

	PROPRIETARY-1	ML10M
I-KNN	2.5 sec	10.7 sec
U-KNN	46.9 sec	-
PureSVD	3 min	1 min 27 sec
WRMF	27 min 3 sec	TBA
U-MF-SVD	3 min 10 sec	1 min 38 sec
I-MF-SVD	3 min 8 sec	1 min 39 sec
U-LRec-Low	3 min 27 sec	1 min 44 sec
I-Lrec-Low	3 min 32 sec	1 min 42 sec
SLIM	32 min 37 sec	7 min 40 sec

4.5 Runtime Evaluation

To compare the training efficiency of various algorithm, we choose PROPRIETARY-2 and ML10M dataset for analysis. We benchmarked the training time of the algorithms by training the model on workstation with 128 GB of main memory and Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz with 32 cores. All of the method exploits multi-core enabled via linear algebra library, whereas SLIM and WRMF attains parallelism via multiprocessing. For a fair comparision, we ran SLIM and WRMF parallelly to use all available cores. In table 9 we compare the runtime of proposed method with the baseline methods on PROPRIETARY-2 dataset. We made following observation

- SLIM is computationally expensive and is slowest among the baselines. Whereas, Linear-Flow is 10 fold faster than SLIM. Furthermore, the computational bottleneck for Linear-Flow is mainly in the SVD computation. Since the method heavily depends upon the linear algebra library, we expect significant speedup by using the GPU backend linear algebra library [16].
- Neighborhood methods are blazingly fast as they involve only sparse linear algebra.
- While WRMF is computationally expensive, as it involves computing inverse for individual users and items in each optimization step, other matrix factorization based methods have similar computational footprints as Low-Linear method.

B: Table 9 is one of the most important results. We need more than one dataset.

S: Suvash: Since other datasets are not large in terms of number of items, I think there wont be significant difference in the training time between proposed method and SLIM which may give a wrong impression.

5 Conclusion

In this paper, we proposed a fast low dimensional regularized linear model, Linear-Flow , to addresses the computational bottleneck of linear model, enabling it to scale up to large OC-CF problem while retaining essentially the same performance. In our comprehensive experimentation, we illustrated that the proposed method is computationally superior to the state-of-the-art, more specifically 10 folds faster, and yields competitive performance. Future work may explore user and item side-information as well as further improving the computational and memory footprints by exploring more efficient dimensionality reduction techniques.

References

[1] J. Bennett, C. Elkan, B. Liu, P. Smyth, and D. Tikk. KDD cup and workshop 2007. *SIGKDD Explo-*

Table 4: Results on the PROPRIETARY-1 Dataset.Reported numbers are the mean and standard errors across test folds.

	prec@3	prec@5	prec@10	prec@20	recall@3	recall@5	recall@10	recall@20	mAP@20
I-KNN	0.0393 ± 0.0007	0.0291 ± 0.0005	0.0186 ± 0.0002	0.0115 ± 0.0002	0.0797 ± 0.0011	0.0973 ± 0.0015	0.1232 ± 0.0021	0.1521 ± 0.0028	0.0725 ± 0.0008
U-KNN	0.0514 ± 0.0008	0.0386 ± 0.0005	0.0249 ± 0.0003	0.0153 ± 0.0001	0.1068 ± 0.0035	0.1321 ± 0.0037	0.1679 ± 0.0029	0.2052 ± 0.0036	0.0969 ± 0.0033
PureSVD	0.0376 ± 0.0013	0.0267 ± 0.0009	0.0160 ± 0.0005	0.0094 ± 0.0003	0.0776 ± 0.0018	0.0906 ± 0.0018	0.1073 ± 0.0026	0.1247 ± 0.0030	0.0692 ± 0.0022
WRMF	0.0397 ± 0.0008	0.0293 ± 0.0013	0.0183 ± 0.0008	0.0113 ± 0.0004	0.0787 ± 0.0021	0.0955 ± 0.0047	0.1186 ± 0.0045	0.1467 ± 0.0037	0.0707 ± 0.0025
U-MF-RSVD	0.0503 ± 0.0007	0.0381 ± 0.0004	0.0247 ± 0.0003	0.0155 ± 0.0001	0.1003 ± 0.0024	0.1301 ± 0.0017	0.1693 ± 0.0032	0.2069 ± 0.0039	0.0961 ± 0.0026
I-MF-RSVD	0.0480 ± 0.0010	0.0360 ± 0.0008	0.0234 ± 0.0004	0.0144 ± 0.0002	0.0976 ± 0.0030	0.1211 ± 0.0033	0.1550 ± 0.0038	0.1886 ± 0.0037	0.0889 ± 0.0030
SLIM	**0.0519 ± 0.0010**	0.0395 ± 0.0004	0.0258 ± 0.0004	0.0160 ± 0.0002	**0.1069 ± 0.0034**	0.1341 ± 0.0026	0.1729 ± 0.0039	0.2117 ± 0.0033	**0.0976 ± 0.0029**
U-Linear-Flow	0.0518 ± 0.0013	0.0391 ± 0.0005	0.0259 ± 0.0004	0.0160 ± 0.0002	0.1060 ± 0.0035	0.1350 ± 0.0032	0.1711 ± 0.0026	0.2118 ± 0.0029	0.0970 ± 0.0030
I-Linear-Flow	0.0520 ± 0.0012	**0.0398 ± 0.0004**	**0.0262 ± 0.0003**	**0.0162 ± 0.0002**	0.1062 ± 0.0032	**0.1362 ± 0.0026**	**0.1758 ± 0.0026**	**0.2145 ± 0.0033**	0.0971 ± 0.0028

Table 5: Results on the PROPRIETARY-2 Dataset. Reported numbers are the mean and standard errors across test folds.

	prec@3	prec@5	prec@10	prec@20	recall@3	recall@5	recall@10	recall@20	mAP@20
I-KNN	0.0873 ± 0.0003	0.0641 ± 0.0002	$0.0400 \pm 2.4970 \times 10^{-5}$	$0.0236 \pm 2.2155 \times 10^{-5}$	0.1743 ± 0.0005	0.2097 ± 0.0007	0.2566 ± 0.0007	0.2982 ± 0.0008	0.1591 ± 0.0007
U-KNN	0.0836 ± 0.0005	0.0605 ± 0.0003	0.0365 ± 0.0002	$0.0207 \pm 7.6396 \times 10^{-5}$	0.1711 ± 0.0011	0.2029 ± 0.0013	0.2407 ± 0.0013	0.2695 ± 0.0014	0.1532 ± 0.0007
WRMF	0.0579 ± 0.0005	0.0437 ± 0.0004	0.0283 ± 0.0001	$0.0175 \pm 8.6682 \times 10^{-5}$	0.1145 ± 0.0013	0.1417 ± 0.0015	0.1801 ± 0.0012	0.2184 ± 0.0012	0.1053 ± 0.0011
PureSVD	0.0335 ± 0.0004	0.0244 ± 0.0002	0.0153 ± 0.0002	$0.0094 \pm 9.7863 \times 10^{-5}$	0.0686 ± 0.0006	0.0823 ± 0.0005	0.1018 ± 0.0009	0.1233 ± 0.0010	0.0624 ± 0.0004
U-MF-RSVD	0.0699 ± 0.0003	0.0502 ± 0.0002	$0.0305 \pm 6.9209 \times 10^{-5}$	$0.0178 \pm 2.4425 \times 10^{-5}$	0.1408 ± 0.0007	0.1660 ± 0.0008	0.1985 ± 0.0005	0.2282 ± 0.0004	0.1277 ± 0.0006
I-MF-RSVD	0.0880 ± 0.0003	0.0652 ± 0.0001	$0.0408 \pm 6.5965 \times 10^{-5}$	$0.0242 \pm 3.4697 \times 10^{-5}$	0.1720 ± 0.0008	0.2199 ± 0.0008	0.2685 ± 0.0010	0.3203 ± 0.0011	0.1582 ± 0.0005
SLIM	**0.0893 ± 0.0004**	**0.0671 ± 0.0003**	$\mathbf{0.0433 \pm 9.5600 \times 10^{-5}}$	$\mathbf{0.0264 \pm 4.3581 \times 10^{-5}}$	**0.1796 ± 0.0010**	**0.2213 ± 0.0013**	**0.2790 ± 0.0012**	**0.3336 ± 0.0009**	**0.1654 ± 0.0006**
U-Linear-Flow	0.0887 ± 0.0004	0.0666 ± 0.0002	$0.0430 \pm 8.6598 \times 10^{-5}$	$0.0258 \pm 4.0894 \times 10^{-5}$	0.1763 ± 0.0012	0.2214 ± 0.0010	0.2739 ± 0.0013	0.3289 ± 0.0010	0.1611 ± 0.0008
I-Linear-Flow	0.0885 ± 0.0003	0.0656 ± 0.0002	$0.0429 \pm 8.6598 \times 10^{-5}$	$0.0256 \pm 4.0894 \times 10^{-5}$	0.1783 ± 0.0012	0.2202 ± 0.0012	0.2719 ± 0.0010	0.3259 ± 0.0010	0.1598 ± 0.0006

Table 6: Results on the LASTFM dataset. Reported numbers are the mean and standard errors across test folds.

	prec@3	prec@5	prec@10	prec@20	recall@3	recall@5	recall@10	recall@20	mAP@20
I-KNN	0.5904 ± 0.0068	0.5600 ± 0.0067	0.5068 ± 0.0053	0.4390 ± 0.0054	0.0187 ± 0.0004	0.0284 ± 0.0002	0.0497 ± 0.0007	0.0824 ± 0.0016	0.3280 ± 0.0055
U-KNN	0.5434 ± 0.0083	0.5127 ± 0.0038	0.4619 ± 0.0044	0.4009 ± 0.0028	0.0169 ± 0.0004	0.0260 ± 0.0011	0.0448 ± 0.0011	0.0747 ± 0.0014	0.2894 ± 0.0029
WRMF	0.6277 ± 0.0071	0.5899 ± 0.0049	0.5308 ± 0.0049	0.4577 ± 0.0032	0.0198 ± 0.0006	0.0300 ± 0.0009	0.0516 ± 0.0015	0.0842 ± 0.0013	0.3562 ± 0.0036
PureSVD	0.3993 ± 0.0121	0.3690 ± 0.0082	0.3321 ± 0.0069	0.2880 ± 0.0069	0.0128 ± 0.0003	0.0194 ± 0.0004	0.0338 ± 0.0006	0.0567 ± 0.0018	0.1723 ± 0.0065
U-MF-RSVD	0.5176 ± 0.0042	0.4865 ± 0.0068	0.4423 ± 0.0028	0.3859 ± 0.0038	0.0141 ± 0.0005	0.0217 ± 0.0008	0.0387 ± 0.0005	0.0653 ± 0.0013	0.2810 ± 0.0023
I-MF-RSVD	0.5780 ± 0.0063	0.5447 ± 0.0050	0.4901 ± 0.0011	0.4236 ± 0.0029	0.0190 ± 0.0008	0.0292 ± 0.0006	0.0496 ± 0.0011	0.0811 ± 0.0009	0.3146 ± 0.0020
SLIM	**0.6319 ± 0.0069**	**0.6002 ± 0.0055**	**0.5384 ± 0.0048**	0.4639 ± 0.0056	**0.0223 ± 0.0006**	**0.0346 ± 0.0004**	**0.0592 ± 0.0007**	**0.0972 ± 0.0017**	**0.3587 ± 0.0065**
U-Linear-Flow	0.6229 ± 0.0081	0.5912 ± 0.0067	0.5337 ± 0.0027	0.4627 ± 0.0020	0.0205 ± 0.0009	0.0315 ± 0.0014	0.0540 ± 0.0004	0.0881 ± 0.0010	0.3563 ± 0.0025
I-Linear-Flow	0.6229 ± 0.0103	0.5913 ± 0.0046	0.5337 ± 0.0021	**0.4653 ± 0.0023**	0.0203 ± 0.0011	0.0310 ± 0.0010	0.0529 ± 0.0006	0.0874 ± 0.0015	**0.3615 ± 0.0014**

Table 7: Results on the ML10M dataset. Reported numbers are the mean and standard errors across test folds.

	prec@3	prec@5	prec@10	prec@20	recall@3	recall@5	recall@10	recall@20	mAP@20
I-KNN	0.1753 ± 0.0011	0.1506 ± 0.0009	0.1179 ± 0.0006	0.0883 ± 0.0003	0.0994 ± 0.0004	0.1393 ± 0.0009	0.2121 ± 0.0012	0.3070 ± 0.0012	0.1216 ± 0.0003
U-KNN					Out of Memory				
WRMF	0.1832 ± 0.0007	0.1561 ± 0.0006	0.1205 ± 0.0002	0.0889 ± 0.0002	0.1024 ± 0.0003	0.1428 ± 0.0008	0.2139 ± 0.0009	0.3061 ± 0.0010	0.1255 ± 0.0003
PureSVD	0.1216 ± 0.0013	0.1054 ± 0.0006	0.0836 ± 0.0005	0.0634 ± 0.0002	0.0730 ± 0.0014	0.1030 ± 0.0011	0.1572 ± 0.0009	0.2264 ± 0.0011	0.0836 ± 0.0010
U-MF-RSVD	0.2229 ± 0.0007	0.1895 ± 0.0007	0.1456 ± 0.0008	0.1069 ± 0.0004	0.1273 ± 0.0003	0.1755 ± 0.0005	0.2592 ± 0.0014	0.3656 ± 0.0018	0.1586 ± 0.0005
I-MF-RSVD	0.2232 ± 0.0009	0.1902 ± 0.0010	0.1461 ± 0.0005	0.1027 ± 0.0004	0.1246 ± 0.0007	0.1745 ± 0.0010	0.2602 ± 0.0008	0.3675 ± 0.0017	0.1590 ± 0.0007
SLIM	0.2208 ± 0.0011	0.1888 ± 0.0011	0.1464 ± 0.0004	0.1080 ± 0.0004	0.1258 ± 0.0003	0.1748 ± 0.0012	0.2611 ± 0.0009	0.3690 ± 0.0016	0.1579 ± 0.0007
U-Linear-Flow	**0.2270 ± 0.0012**	**0.1927 ± 0.0009**	**0.1477 ± 0.0006**	**0.1083 ± 0.0004**	**0.1290 ± 0.0004**	**0.1777 ± 0.0008**	**0.2624 ± 0.0013**	**0.3701 ± 0.0016**	**0.1601 ± 0.0006**
I-Linear-Flow	0.2242 ± 0.0009	0.1909 ± 0.0010	0.1468 ± 0.0005	0.1077 ± 0.0004	0.1276 ± 0.0007	0.1765 ± 0.0010	0.2609 ± 0.0008	0.3676 ± 0.0017	0.1592 ± 0.0007

Table 8: Top-3 similar items learned by I-Linear-Flow model.

Item	Chemistry	Chilling out	Workers	Unemployment	Divorce and Conflict
Similar items	Test and Analysis	Beach Holidays	Construction	Job Search	Depression
	Drug and Pills	Tourism	Teamwork	Tax and Accounting	Getting upset
	Health Care	Relaxing	Manufacturing	Breaking the law and Illegal Activity	Crying

rations Newsletter, 9(2):51–52, Dec. 2007.

[2] Ò. Celma. *Music Recommendation and Discovery in the Long Tail*. PhD thesis, Universitat Pompeu Fabra, Barcelona, 2008.

[3] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *ACM Conference on Recommender Systems (RecSys)*, RecSys '10, pages 39–46, New York, NY, USA, 2010. ACM.

[4] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[5] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM '08*, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[6] Y. Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *KDD '08*, pages 426–434, New York, NY, USA, 2008. ACM.

[7] M. Levy and K. Jack. Efficient top-n recommendation by linear regression. In *RecSys Large Scale Recommender Systems Workshop*, 2013.

[8] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, Jan. 2003.

[9] X. Ning and G. Karypis. SLIM: Sparse linear methods for top-n recommender systems. In *IEEE International Conference on Data Mining (ICDM)*. IEEE, 2011.

[10] R. Pan and M. Scholz. Mind the gaps: Weighting the unknown in large-scale one-class collaborative filtering. In *KDD '09*, pages 667–676, New York, NY, USA, 2009. ACM.

[11] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW '01*, pages 285–295, New York, NY, USA, 2001. ACM.

[12] S. Sedhain, A. Menon, S. Sanner, and D. Braziunas. On the effectiveness of linear models for one-class collaborative filtering. *Proceedings of the 30th Conference on Artificial Intelligence (AAAI-16)*, 2016.

[13] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, 2015.

[14] L. Tang and P. Harrington. Scaling matrix factorization for recommendation with randomness. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 39–40. International World Wide Web Conferences Steering Committee, 2013.

[15] J. Vig, S. Sen, and J. Riedl. Tagsplanations: Explaining recommendations using tags. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*, IUI '09, pages 47–56, New York, NY, USA, 2009. ACM.

[16] S. Voronin and P.-G. Martinsson. Rsvdpack: Subroutines for computing partial singular value decompositions via randomized sampling on single core, multi core, and gpu architectures. *arXiv preprint arXiv:1502.05366*, 2015.