Scaling Linear Models for One-Class Collaborative Filtering via Dimensionality Reduction

Anonymous

Abstract

In recent years, collaborative filtering has emerged as the de facto approach to personalized recommendation problems. However, a practically pervasive scenario that proves difficult for most collaborative filtering techniques is where one has only examples of items a user prefers, but no examples of items they do not prefer. In such implicit feedback or one-class collaborative filtering(OC-CF) problems, it is desirable to have recommendation methods that are personalized, learning based and highly scalable. Linear recommenders are proven to be very effective in OC-CF task and have several desirable properties. Although they have superior performance, linear models involves solving a large number of a regression problem making them computationally expensive and limiting their applicability in a large-scale recommendation task. This paper focuses on developing a pipeline and fast algorithm for fast large scale linear OC-CF models to large datasets. We propose a fast low dimensional regularized linear model, Linear-Flow , that learns OC-CF model on low dimensional projection of large scale user-item interaction matrix. A comprehensive set of experiments illustrates that Linear-Flow is computationally superior and yields competitive performance compared to the state-of-the-art methods.

1 Introduction

Personalised recommendation systems are a core component of many modern e-commerce services. Collaborative filtering (CF) is the de-facto approach to an algorithmic recommendation, based on extracted information from a database of item preferences for a collection of users. Most work on CF has considered the explicit feedback setting, where users express positive and negative preferences for items in terms of ratings or likes/dislikes. By 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)

While there is a rich literature on OC-CF (discussed subsequently), to our knowledge, all existing methods lack one or more desiderata. For example, methods that do not employ learning (such as neighbourhood approaches) cannot ensure optimal exploitation of available data. On the other hand, amongst methods that employ learning (such as matrix factorisation), it is common to employ learning objectives that are non-convex, which limits the range of methods for efficient global optimisation (if even possible) for OC-CF problems. While linear models have been shown to perform well on OC-CF problems, they are computationally expensive as they require solving large number of regression problem.

Furthermore, while 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 **performance**. The performance 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** 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 [17].

Neighborhood based methods are scalable, incorporates similarity metric in the model and gives explainable recommendations. However, they do not perform well compared to the linear counterparts. Matrix Factorization models are scalable but are not competitive in terms of performance. 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 performance. Furthermore, the model explicitly learns the similarity metric. Also, like neighborhood models, recommendations from linear model are easily explainable. This makes linear methods an ideal choice for OC-CF. However, the linear methods are computationally expensive which limits its applicability in real world scenario. Table 1 summarises the strengths and weaknesses of existing OC-CF methods.

In this paper, we address the computational limitation of linear models by proposing an algorithm that uses fast randomized SVD dimensionality reduction to scale linear models on large-scale datasets. Experimentation on a range of real-world datasets reveals that Linear-Flow provides competitive performance with a significant reduction of the computational cost compared to the state-of-the-art Linear models. Also, as shown in Table 1, Linear-Flow has all desirable properties of a practical recommender system compared to other state-of-the-art methods.

2 Background

We now introduce our notation, summarised 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*. As the entries are real-valued, for each user u, we can sort the entries of $\hat{\mathbf{R}}_{u:}$ to obtain a predicted ranking over items. We evaluate $\hat{\mathbf{R}}$ assuming we are in the *top-N recommendation* scenario [4]. Here, the interest is in finding $\hat{\mathbf{R}}$ such that the head of the ranked list for each user comprises items she will enjoy.

The challenge in OC-CF is the lack of examples of explicit negative preference. While we may assume that a user purchasing an item is an indication of positive preference, a user not purchasing an item does not necessarily indicate a negative preference: the user may simply be unaware of the item. Henceforth, we say that (u, i) pairs with $\mathbf{R}_{ui} = 1$ denote "known positive preference", while pairs with $\mathbf{R}_{ui} = 0$ denote "unknown preference".

We now review some existing approaches to OC-CF. Broadly, the OC-CF algorithms can be categorised into three categories, namely, Neighborhood, Linear and Bilinear methods.

[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.

2.1 Neighbourhood methods

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

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

where $\mathbf{S} \in \mathbb{R}^{n \times n}$ is some *item-similarity matrix*. That is, for a user u, one scores an item i as the *sum of i's similarity to all other items that u enjoys*. Typically, one uses a predefined matrix $\mathbf{S}$ that relies on $\mathbf{R}$. A popular example is cosine similarity [13, 10],

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

Other examples are the Pearson correlation, Jaccard coefficient and conditional probability [4]. It is typical to sparsify $\mathbf{S}$ so that its rows and columns only keep the largest entries, i.e. to compute the k-neighbourhood graph based on $\mathbf{S}$.

Neighbourhood methods are attractive for several reasons. They are simple to implement, efficient (as the computation of (1) requires $O(|\mathcal{R}(u)|)$ time), and interpretable. However, their biggest drawback is that they rely on a fixed $\mathbf{S}$ that is not learned from data [7]. Recommendation performance can be quite sensitive to the choice of $\mathbf{S}$, and the choice generally depends on the problem domain. Therefore, neighbourhood methods are unable to adapt to the characteristics of the data at hand.

2.2 Linear Recommenders

An alternative to neighborhood models, Linear methods [11, 14] learns the similarity metric from the data. SLIM [11] views the recommendation as learning item-item similarity and learns an item-similarity matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ via

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

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

$$\mathcal{C} = \{\mathbf{W} \in \mathbb{R}^{n \times n} \colon \text{diag}(\mathbf{W}) = 0, \mathbf{W} \geq 0\}. \tag{3}$$

Here, $|| \cdot ||_1$ denotes the elementwise ℓ_1 norm of $\mathbf{W}$ so as to encourage sparsity, and the constraint $\text{diag}(\mathbf{W}) = 0$ 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.

Similarly, [14] decomposed OO-CF as learning a linear model per user for personalized recommendation. Despite its superior performance, proposed method is computationally expensive and memory exhaustive restricting its applicability in real world problem.

Linear methods are attractive for several reasons. They have superior performance [11, 14] and unlike neighborhood methods, they adapt with the data as the parameters are learned from data itself. Furthermore, the recommendations are easily interpretable. However, the linear methods are computationally expensive as they require solving large number of regression problem for a huge design matrix R.

Method	Performance	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.

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

2.3 Bilinear Models

Matrix factorisation

Matrix factorisation 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 [15, 8].

Formally, let $\mathbf{J} \in \mathbb{R}_+^{m \times n}$ be some pre-defined weighting matrix to be defined shortly. Let $\ell \colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$ be some loss function, typically squared loss $\ell(y, \hat{y}) = (y - \hat{y})^2$. Then, matrix factorisation methods optimise

$$\min_{\theta} \sum_{u \in \mathcal{U}, i \in \mathcal{I}} \mathbf{J}_{ui} \cdot \ell(\mathbf{R}_{ui}, \hat{\mathbf{R}}_{ui}(\theta)) + \Omega(\theta), \tag{4}$$

where the recommendation matrix is[2]

$$\hat{\mathbf{R}}(\theta) = \mathbf{A}^T \mathbf{B} \tag{5}$$

for $\theta = \{\mathbf{A}, \mathbf{B}\}$, and $\Omega(\theta)$ is the *regulariser*, which is typically

$$\Omega(\theta) = \frac{\lambda}{2} \cdot (||\mathbf{A}||_F^2 + ||\mathbf{B}||_F^2)$$

for some $\lambda > 0$. The matrices $\mathbf{A} \in \mathbb{R}^{K \times m}, \mathbf{B} \in \mathbb{R}^{K \times n}$ are the *latent representations* of users and items respectively, with $K \in \mathbb{N}_+$ being the *latent dimension* of the factorisation.

In standard collaborative filtering applications with explicit negative feedback, one typically sets [8] $\mathbf{J}_{ui} = [\![\mathbf{R}_{ui} > 0]\!]$, so that one only considers (user, item) pairs with known preference information. In implicit feedback problems, this is not appropriate, as one will simply learn on the known positive preferences, and thus predict $\hat{\mathbf{R}}_{ui} = 1$ uninformatively.

An alternative is to set $\mathbf{J}_{ui} = 1$ uniformly. This treats all absent purchases as indications of a negative preference. While this appears problematic, the resulting approach been shown to perform well in recommendation (as opposed to rating prediction) tasks, and is termed PureSVD in [3]. One appealing property of this choice of $\mathbf{J}$ is that the parameters $\mathbf{A}, \mathbf{B}$ can be found using the singular value decomposition (SVD) of $\mathbf{R}$.

[2]Typically, one also includes user- and item- bias terms in the recommendation matrix. We omit these for brevity.

As an intermediate between the two extreme weighting schemes above, the WRMF method [12] sets $\mathbf{J}_{ui}$ to be

$$\mathbf{J}_{ui} = [\![\mathbf{R}_{ui} = 0]\!] + \alpha \cdot [\![\mathbf{R}_{ui} > 0]\!] \tag{6}$$

where α assigns an importance weight to the observed ratings. More sophisticated weighting schemes have also been explored. In scenarios where there are multiple observations for each (user, item) pair, [6] considered a logarithmic weighting of these counts.

[16] proposed a two step randomized SVD based algorithm to scale matrix factorization. First, they compute the rank-k 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 and fix the item latent factor with SVD solution and solve

$$\begin{gathered} \underset{\mathbf{A}}{\operatorname{argmin}} \left\|\mathbf{R} - \mathbf{A}^T \mathbf{B}\right\|_F^2 + \lambda \left\|\mathbf{A}\right\|_F^2 \\ where,\ \mathbf{B} = \mathbf{\Sigma}^{\frac{1}{2}} \mathbf{Q}_k^T \end{gathered} \tag{7}$$

In the (7), the item latent factors, $\mathbf{B}$, is initialized with SVD solution and fixed. Similarly, instead of fixing $\mathbf{B}$, if we fix $\mathbf{A}$, we get the following objective

$$\begin{gathered} \underset{\mathbf{B}}{\operatorname{argmin}} \left\|\mathbf{R} - \mathbf{A}^T \mathbf{B}\right\|_F^2 + \lambda \left\|\mathbf{B}\right\|_F^2 f \\ where,\ \mathbf{A} = \mathbf{P}_k \mathbf{\Sigma}^{\frac{1}{2}} \end{gathered} \tag{8}$$

We refer (8) and (7) as U-MF-RSVD and I-MF-RSVD respectively. Furthermore, we emperically show that the performance varies for these variants .

3 Large Scale Linear Methods

Despite the superior performance, applicability of Linear methods are constrained by its computational cost. Linear methods involve solving a large number of regression problem on a huge design matrix $\mathbf{R}$ making it extremely challenging on real world applications where the order of user and items ranges in millions.

In this section, we propose an algorithm to scale linear methods on large scale datasets. First, we define L_2 regularized linear model

$$\underset{\mathbf{W}}{\operatorname{argmin}} ||\mathbf{R} - \mathbf{R}\mathbf{W}||_F^2 + \frac{\lambda}{2} ||\mathbf{W}||_F^2 \tag{9}$$

We observe that (9) is a standard multiple linear regression problem with analytical solution as

$$\mathbf{W} = (\mathbf{R}^T\mathbf{R} + \lambda\mathbf{I})^{-1}\mathbf{R}^T\mathbf{R} \tag{10}$$

However, the memory requirement, computational cost and numerical instability in computing the inverse, $(\mathbf{R}^T\mathbf{R} + \lambda\mathbf{I})^{-1}$, makes the analytical solution infeasible. Similarly, if we apply gradient based optimization, it becomes computationally expensive as it involves solving n linear regression.

To address the computational and numerical issues, we reformulate the linear model with additional rank constraints

$$\underset{rank(\mathbf{W})\leq k}{\text{argmin}} \ \|\mathbf{R} - \mathbf{R}\mathbf{W}\|_F^2 + \lambda \|\mathbf{W}\|_F^2 \qquad where,\ k << n \tag{11}$$

For $\lambda = 0$, the optimal solution is given by SVD as

$$\mathbf{W} = \mathbf{Q}_k\mathbf{Q}_k^T \tag{12}$$

where $\mathbf{Q}_k$ is given by truncated SVD

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

However, the key limitation of the SVD solution is that it corresponds to an unregularized model, hence highly susceptible to overfitting. To incorporate model complexity control via regularization, we constrain (9) further by factoring $W \in span(Q_k)$ by writing $W = Q_kY$ which gives the following optimization objective

$$\underset{\mathbf{Y}}{\text{argmin}} \|\mathbf{R} - \mathbf{R}\mathbf{Q}_k\mathbf{Y}\|_F^2 + \lambda \|\mathbf{Q}_k\mathbf{Y}\|_F^2 \tag{13}$$

Since $\mathbf{Q}_k$ is orthonormal, we get $\|\mathbf{Q}_k\mathbf{Y}\|_F = \|\mathbf{Y}\|_F$. Hence 13 is equivalent to

$$\underset{\mathbf{Y}}{\text{argmin}} \|\mathbf{R} - \mathbf{R}\mathbf{Q}_k\mathbf{Y}\|_F^2 + \lambda \|\mathbf{Y}\|_F^2 \tag{14}$$

For a resonable rank k, we can efficiently solve (13) with the analytical solution

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

In other words, the model corresponding to 13 projects the user-item matrix rank-k orthogonal basis and then optimises the regularised squared loss objective on the projected representation.

We refer to (13) as I-Linear-Flow as it corresponds to item-item model. Similarly, we can define a user-user model, U-Linear-Flow , in (15)

$$\underset{\mathbf{Y}}{\text{argmin}} \|\mathbf{R} - \mathbf{R}\mathbf{P}_k\mathbf{Y}\|_F^2 + \lambda \|\mathbf{Y}\|_F^2 \tag{15}$$

Although low rank multiple-regression can be solved efficiently, it requires computationally expensive dimensionality reduction. However, the recent advances in randomized algorithms for approximate dimensionality reduction [5] have made dimensionality reduction very efficient on large scale matrices. In this work, we use randomized SVD [5] for low dimensional projection which is summarized in Algorithm 1.

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

```
1: procedure RSVD(R, k)
2:     Draw $n \times k$ Gaussian random matrix $\Omega$
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 $B = Z^TR$
6:     Compute the SVD of B, B = $\hat{P}_k\mathbf{\Sigma}_kQ_k$
7:     $R \Rightarrow ZB \Rightarrow Z\hat{P}_k\mathbf{\Sigma}_kQ_k \Rightarrow P_k\mathbf{\Sigma}_kQ_k, where\ P_k = Z\hat{P}_k$
8:     return $P_k\mathbf{\Sigma}_kQ_k$
9: end procedure
```

Table 3: Summary of datasets used in evaluation.

Dataset	m	n	$\lvert\mathbf{R}_{ui} > 0\rvert$
ML10M	69,557	9,309	2,909,119
LASTFM	992	88,428	800,274
PROPRIETARY-1	26,928	14,399	120,268
PROPRIETARY-2	264,054	57,214	1,398,332

As discussed earlier, in a real world recommendation item-item similarities are widely used to recommend similar items. As a by-product of the I-Linear-Flow model item-item similarity can be recovered as Q_kY. Unlike matrix factorization model, notion of the similarity is directly embedded in the model definition.

4 Experiment and Evaluation

4.1 Dataset

We now report experimental results that 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 all methods. In all of our dataset, we remove the users and items with less than 3 purchases. Table 3 summarises statistics of the datasets.

ML10M. The MovieLens 10M dataset[3] 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 $\mathbf{R} \in \{0, 1, \ldots, 5\}^{m\times n}$, we created a preference matrix $\tilde{\mathbf{R}}$ with $\tilde{\mathbf{R}}_{ui} = [\![\mathbf{R}_{ui} \geq 4]\!]$.

LASTFM. The LastFM dataset[4] [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[5], a major provider of third party

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

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

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

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.

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].
- Weighted matrix factorisation (WRMF) as defined in (4).
- MF-RSVD of [16]. We ran this method both on user and item based initialization, U-MF-RSVD and I-MF-RSVD, as discussed in (8) and (7) respectively.
- SLIM, as per (2). For computational convenience, we used the SGDReg variant [9], which is identical to SLIM except that the nonnegativity constraint is removed. We did not evaluate SLIM with nonnegativity directly as [9] reports superior performance to SLIM, and is considerably faster to train.

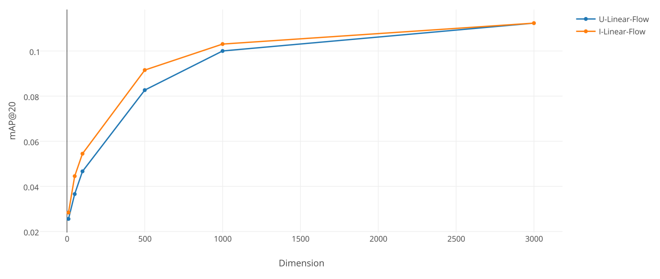
Note that, We didn't compare against [14] due to its memory complexity on a large scale dataset. For instance, on PROPRIETARY-2 dataset LRec requires ∼260 GB of memory.

4.4 Performance Evaluation

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

- Linear-Flow generally is competitive with SLIM and outperforms the other methods on all four datasets. Unlike other methods, Linear-Flow performance 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 dimensions of the projection.
- Amongst the matrix factorisation methods, Randomized SVD method [16] consistently performs the best in all dataset. This is possibly due to the fact that SVD provides a good initialization and the linear regression finds the optimal solution for the given initialization which could have been missed by optimizing non-convex bilinear objective initialized with random parameters. Furthermore, we observe that the performance of MF-RSVD varies depending upon the choice of latent factors we choose to initialize.
- Neighborhood based methods are computationally inexpensive. However, they are consistently outperformed by the linear methods in all of our dataset.

Figure 1: Performance of the user and item Linear-Flow with the number of dimensions. [NOTE: This graph is incomplete and will be updated as soon as I get the results of the ongoing experiment.]



4.5 Runtime Evaluation

To compare the efficiency of various algorithm, we choose our largest dataset i.e. PROPRIETARY-2 in terms of users and items. 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 8 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 computation of the orthonormal basis via randomized SVD. Since the method heavily depends upon the linear algebra library, we expect significant speedup by using the GPU backend linear algebra library [18].
- 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 iteration of ALS, other matrix factorization based methods have similar computational footprints as Low-Linear method.

5 Conclusion

In this paper, we defined a novel fast low dimensional regularized linear model, $Linear - Flow$, for one-class collaborative filtering. The proposed method exploits fast randomized SVD algorithm for dimensionality reduction followed by an

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
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
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**

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
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
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**

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
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**
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**

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
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
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

Table 8: Model training time on the PROPRIETARY-1 Dataset.

	Training time
I-KNN	2.5 sec
U-KNN	46.9 sec
PureSVD	3 min
WRMF	27 min 3 sec
U-MF-SVD	3 min 10 sec
I-MF-SVD	3 min 8 sec
U-LRec-Low	3 min 27 sec
I-Lrec-Low	3 min 32 sec
SLIM	32 min 37 sec

analytical solution to the low dimensional multi-regression problem for scalability. 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 Explorations 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] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems*, 22(1):143–177, Jan. 2004.

[5] 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.

[6] 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.

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

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

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

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

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

[12] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *IEEE International Conference on Data Mining (ICDM)*, pages 502–511, Washington, DC, USA, 2008. IEEE Computer Society.

[13] 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.

[14] 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.

[15] N. Srebro and T. Jaakkola. Weighted low-rank approximations. In *ICML*, pages 720–727, 2003.

[16] 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.

[17] 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.

[18] 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.