2011 11th IEEE International Conference on Data Mining

SLIM: Sparse Linear Methods for Top-N Recommender Systems

Xia Ning and George Karypis
Computer Science & Engineering
University of Minnesota, Minneapolis, MN
Email: {xning,karypis@cs.umn.edu}

***Abstract*—This paper focuses on developing effective and efficient algorithms for *top-N* recommender systems. A novel Sparse LInear Method (SLIM) is proposed, which generates *top-N* recommendations by aggregating from user purchase/rating profiles. A sparse aggregation coefficient matrix W is learned from SLIM by solving an ℓ_1-norm and ℓ_2-norm regularized optimization problem. W is demonstrated to produce high-quality recommendations and its sparsity allows SLIM to generate recommendations very fast. A comprehensive set of experiments is conducted by comparing the SLIM method and other state-of-the-art *top-N* recommendation methods. The experiments show that SLIM achieves significant improvements both in run time performance and recommendation quality over the best existing methods.**

***Keywords*-Top-N Recommender Systems, Sparse Linear Methods, ℓ_1-norm Regularization**

I. Introduction

The emergence and fast growth of E-commerce have significantly changed people's traditional perspective on purchasing products by providing huge amounts of products and detailed product information, thus making online transactions much easier. However, as the number of products conforming to the customers' desires has dramatically increased, the problem has become how to effectively and efficiently help customers identify the products that best fit their personal tastes. In particular, given the user purchase/rating profiles, recommending a ranked list of items for the user so as to encourage additional purchases has the most application scenarios. This leads to the widely used *top-N* recommender systems.

In recent years, various algorithms for *top-N* recommendation have been developed [1]. These algorithms can be categorized into two classes: neighborhood-based collaborative filtering methods and model-based methods. Among the neighborhood-based methods, those based on item neighborhoods can generate recommendations very fast, but they achieve this with a sacrifice on recommendation quality. On the other hand, model-based methods, particularly those based on latent factor models incur a higher cost while generating recommendations, but the quality of these recommendations is higher, and they have been shown to achieve the best performance especially on large recommendation tasks.

In this paper, we propose a novel Sparse LInear Method (SLIM) for *top-N* recommendation that is able to make *high-quality* recommendations *fast*. SLIM learns a sparse coefficient matrix for the items in the system solely from the user purchase/rating profiles by solving a regularized optimization problem. Sparsity is introduced into the coefficient matrix which allows it to generate recommendations efficiently. Feature selection methods allow SLIM to substantially reduce the amount of time required to learn the coefficient matrix. Furthermore, SLIM can be used to do *top-N* recommendations from ratings, which is a less exploited direction in recommender system research.

The SLIM method addresses the demands for high quality and efficiency in *top-N* recommender systems concurrently, so it is better suitable for real-time applications. We conduct a comprehensive set of experiments on various datasets from different real applications. The results show that SLIM produces better recommendations than the state-of-the-art methods at a very high speed. In addition, it achieves good performance in using ratings to do *top-N* recommendation.

The rest of this paper is organized as follows. In Section II, a brief review on related work is provided. In Section III, definitions and notations are introduced. In Section IV, the methods are described. In Section V, the materials used for experiments are presented. In Section VI, the results are presented. Finally in Section VII are the discussions and conclusions.

II. Related Work

Top-N recommender systems are used in E-commerce applications to recommend size-N ranked lists of items that users may like the most, and they have been intensively studied during the last few years. The methods for *top-N* recommendation can be broadly classified into two categories. The first category is the neighborhood-based collaborative filtering methods [2]. For a certain user, *user-based k-nearest-neighbor* (userkNN) collaborative filtering methods first identify a set of similar users, and then recommend *top-N* items based on what items those similar users have purchased. Similarly, *item-based k-nearest-neighbor* (itemkNN) collaborative filtering methods first identify a set of similar items for each of the items that the user has purchased, and then recommend *top-N* items based on those similar items. The user/item similarity is calculated from user-item purchase/rating matrix in a collaborative filtering fashion with some similarity measures (e.g., Pearson correlation, cosine similarity) applied. One advantage of the item-based methods is that they are efficient to generate recommendations due to the fact that the item neighborhood

1550-4786/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDM.2011.134

is sparse. However, they suffer from low accuracy since there is essentially no knowledge learned about item characteristics so as to produce accurate *top-N* recommendations.

The second category is model-based methods, particularly latent factor models as they have achieved the state-of-the-art performance on large-scale recommendation tasks. The key idea of latent factor models is to factorize the user-item matrix into (low-rank) user factors and item factors that represent user tastes and item characteristics in a common latent space, respectively. The prediction for a user on an item can be calculated as the dot product of the corresponding user factor and item factor. There are various Matrix Factorization (`MF`)-based methods proposed in recent years for building such latent factor models. Cremonesi *et al* [3] proposed a simple Pure Singular-Value-Decomposition-based (`PureSVD`) matrix factorization method, which describes users and items by the most principle singular vectors of the user-item matrix. Pan *et al* [4] and Hu *et al* [5] proposed a Weighted Regularized Matrix Factorization (`WRMF`) method formulated as a regularized Least-Squares (`LS`) problem, in which a weighting matrix is used to differentiate the contributions from observed purchase/rating activities and unobserved ones. Rennie [6] and Srebro [7] proposed a Max-Margin Matrix Factorization (`MMMF`) method, which requires a low-norm factorization of the user-item matrix and allows unbounded dimensionality for the latent space. This is implemented by minimizing the trace-norm of the reconstructed user-item matrix from the factors. Sindhwani *et al* [8] proposed a Weighted Non-Negative Matrix Factorization (`WNNMF`) method, in which they enforce non-negativity on the user and item factors so as to lend "part-based" interpretability to the model. Hofmann [9] applied Probabilistic Latent Semantic Analysis (`PLSA`) technique for collaborative filtering, which has been shown equivalent to non-negative matrix factorization. `PLSA` introduces a latent space such that the co-occurrence of users and items (i.e., a certain user has purchased a certain item) can be rendered conditionally independent. Koren [10] proposed an intersecting approach between neighborhood-based method and `MF`. In his approach, item similarity is learned simultaneously with a matrix factorization so as to take advantages of both methods.

Top-N recommendation has also been formulated as a ranking problem. Rendle *et al* [11] proposed a Bayesian Personalized Ranking (`BPR`) criterion, which is the maximum posterior estimator from a Bayesian analysis and measures the difference between the rankings of user-purchased items and the rest items. `BPR` can be well adopted for item $k$nn method (`BPRkNN`) and MF methods (`BPRMF`) as a general objective function.

III. Definitions and Notations

In this paper, the symbols u and t will be used to denote the users and items, respectively. Individual users and items will be denoted using different subscripts (i.e., u_i, t_j). The set of all users and items in the system will be denoted by $\mathcal{U}$ ($|\mathcal{U}| = m$) and $\mathcal{T}$ ($|\mathcal{T}| = n$), respectively. The entire set of user-item purchases/ratings will be represented by a user-item purchase/rating matrix A of size $m \times n$, in which the (i, j) entry (denoted by a_{ij}) is 1 or a positive value if user u_i has ever purchased/rated item t_j, otherwise the entry is marked as 0. The i-th row of A represents the purchase/rating history of user u_i on all items $\mathcal{T}$, and this row is denoted by $\mathbf{a}_i^\mathsf{T}$. The j-th column of A represents the purchase/rating history of all users $\mathcal{U}$ on item t_j and this column is denoted by $\mathbf{a}_j$.

In this paper, all vectors (e.g., $\mathbf{a}_i^\mathsf{T}$ and $\mathbf{a}_j$) are represented by bold lower-case letters and all matrices (e.g., A) are represented by upper-case letters. Row vectors are represented by having the transpose supscript$^\mathsf{T}$, otherwise by default they are column vectors. A predicted/approximated value is denoted by having a $\sim$ head. We will use corresponding matrix/vector notations instead of user/item purchase/rating profiles if no ambiguity is raised.

IV. Sparse Linear Methods for *Top-N* Recommendation

A. `SLIM` for Top-N Recommendation

In this paper, we propose a Sparse LInear Method (`SLIM`) to do *top-N* recommendation. In the `SLIM` method, the recommendation score on an un-purchased/-rated item t_j of a user u_i is calculated as a *sparse* aggregation of items that have been purchased/rated by u_i, that is,

$$\tilde{a}_{ij} = \mathbf{a}_i^\mathsf{T} \mathbf{w}_j, \tag{1}$$

where $a_{ij} = 0$ and $\mathbf{w}_j$ is a sparse size-n column vector of aggregation coefficients. Thus, the model utilized by `SLIM` can be presented as

$$\tilde{A} = AW, \tag{2}$$

where A is the binary user-item purchase matrix or the user-item rating matrix, W is an $n \times n$ sparse matrix of aggregation coefficients, whose j-th column corresponds to $\mathbf{w}_j$ as in Equation 1, and each row $\tilde{\mathbf{a}}_i^\mathsf{T}$ of $\tilde{A}$ ($\tilde{\mathbf{a}}_i^\mathsf{T} = \mathbf{a}_i^\mathsf{T} W$) represents the recommendation scores on all items for user u_i. *Top-N* recommendation for u_i is done by sorting u_i's non-purchased/-rated items based on their recommendation scores in $\tilde{\mathbf{a}}_i^\mathsf{T}$ in decreasing order and recommending the *top-N* items.

B. Learning W for `SLIM`

We view the purchase/rating activity of user u_i on item t_j in A (i.e., a_{ij}) as the ground-truth item recommendation score. Given a user-item purchase/rating matrix A of size $m \times n$, we learn the sparse $n \times n$ matrix W in Equation 2 as the minimizer for the following regularized optimization problem:

$$\begin{aligned} \underset{W}{\text{minimize}} \quad & \frac{1}{2}\|A - AW\|_F^2 + \frac{\beta}{2}\|W\|_F^2 + \lambda\|W\|_1 \\ \text{subject to} \quad & W \geq 0 \\ & \text{diag}(W) = 0, \end{aligned} \tag{3}$$

where $\|W\|_1 = \sum_{i=1}^{n}\sum_{j=1}^{n}|w_{ij}|$ is the entry-wise ℓ_1-norm of W, and $\|\cdot\|_F$ is the matrix Frobenius norm. In Equation 3, AW is the estimated matrix of recommendation scores (i.e., $\tilde{A}$) by the sparse linear model as in Equation 2. The first term $\frac{1}{2}\|A - AW\|_F^2$ (i.e., the residual sum of squares) measures how well the linear model fits the training data, and $\|W\|_F^2$ and $\|W\|_1^2$ are ℓ_F-norm and ℓ_1-norm regularization

terms, respectively. The constants β and λ are regularization parameters. The larger the parameters are, the more severe the regularizations are. The non-negativity constraint is applied on W such that the learned W represents positive relations between items, if any. The constraint $\text{diag}(W) = 0$ is also applied so as to avoid trivial solutions (i.e., the optimal W is an identical matrix such that an item always recommends itself so as to minimize $\frac{1}{2}\|A - AW\|_F^2$). In addition, the constraint $\text{diag}(W) = 0$ ensures that a_{ij} is not used to compute $\tilde{a}_{ij}$.

1) ℓ_1-norm and ℓ_F-norm Regularizations for SLIM *:* In order to learn a sparse W, we introduce the ℓ_1-norm of W as a regularizer in Equation 3. It is well known that ℓ_1-norm regularization introduces sparsity into the solutions [12].

Besides the ℓ_1-norm, we have the ℓ_F-norm of W as another regularizer, which leads the optimization problem to an elastic net problem [13]. The ℓ_F-norm measures model complexity and prevents overfitting (as in ridge regression). Moreover, ℓ_1-norm and ℓ_F-norm regularization together implicitly group correlated items in the solutions [13].

2) Computing W: Since the columns of W are independent, the optimization problem in Equation 3 can be decoupled into a set of optimization problems of the form

$$\begin{aligned} \underset{\mathbf{w}_j}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{a}_j - A\mathbf{w}_j\|_2^2 + \frac{\beta}{2}\|\mathbf{w}_j\|_2^2 + \lambda\|\mathbf{w}_j\|_1 \\ \text{subject to} \quad & \mathbf{w}_j \geq \mathbf{0} \\ & w_{j,j} = 0, \end{aligned} \tag{4}$$

which allows each column of W to be solved independently. In Equation 4, $\mathbf{w}_j$ is the j-th column of W and $\mathbf{a}_j$ is the j-th column of A, $\|\cdot\|_2$ is ℓ_2-norm of vectors, and $\|\mathbf{w}_j\|_1 = \sum_{i=1}^{n} |w_{ij}|$ is the entry-wise ℓ_1-norm of vector $\mathbf{w}_j$. Due to the column-wise independence property of W, learning W can be easily parallelized. The optimization problem of Equation 4 can be solved using coordinate descent and soft thresholding [14].

3) SLIM *with Feature Selection:* The estimation of $\mathbf{w}_j$ in Equation 4 can be considered as the solution to a regularized regression problem in which the j-th column of A is the dependent variable to be estimated as a function of the remaining $n-1$ columns of A (independent variables). This view suggests that feature selection methods can potentially be used to reduce the number of independent variables prior to computing $\mathbf{w}_j$. The advantage of such feature selection methods is that they reduce the number of columns in A, which can substantially decrease the overall amount of time required for SLIM learning.

Motivated by these observations, we extended the SLIM method to incorporate feature selection. We will refer to these methods as fsSLIM. Even though many feature selection approaches can be used, in this paper we only investigated one approach, inspired by the itemkNN *top-N* recommendation algorithms. Specifically, since the goal is to learn a linear model to estimate the j-th column of A (i.e., $\mathbf{a}_j$), then the columns of A that are the most similar to $\mathbf{a}_j$ can be used as the selected features. As our experiments will later show, using the cosine similarity and this feature selection approach, leads to a method that has considerably lower computational requirements with minimal quality degradation.

C. Efficient Top-N Recommendation from SLIM

The SLIM method in Equation 2 and the sparsity of W enable significantly faster *top-N* recommendation. In Equation 2, $\mathbf{a}_i^\mathsf{T}$ is always very sparse (i.e., the user usually purchased/rated only a small fraction of all items), and when W is also sparse, the calculation of $\tilde{\mathbf{a}}_i^\mathsf{T}$ can be very fast by exploiting W's sparse structure (i.e., applying a "gather" operation along W columns on its non-zero values in the rows corresponding to non-zero values in $\mathbf{a}_i^\mathsf{T}$). Thus, the computational complexity to do recommendations for user u_i is $O(n_{a_i} \times n_w + N\log(N))$, where n_{a_i} is the number of non-zero values in $\mathbf{a}_i^\mathsf{T}$, and n_w is the average number of non-zero values in the rows of W. The $N\log(N)$ term is for sorting the highest scoring N items, which can be selected from the potentially $n_{ai} \times n_w$ non-zeros entries in $\tilde{\mathbf{a}}_i^\mathsf{T}$ in linear time using linear selection.

D. SLIM vs Existing Linear Methods

Linear methods have already been used for *top-N* recommendation. For example, the itemkNN method in [2] has a linear model similar to that of SLIM. The model of itemkNN is a $k$nn item-item cosine similarity matrix S, that is, each row $\mathbf{s}_i^\mathsf{T}$ has exactly k nonzero values representing the cosine similarities between item t_j and its k most similar neighbors. The fundamental difference between itemkNN and SLIM's linear models is that the former is highly dependent on the pre-specified item-item similarity measure used to identify the neighbors, whereas the later generates W by solving the optimization problem of Equation 3. In this way, W can potentially encode rich and subtle relations across items that may not be easily captured by conventional item-item similarity metrics. This is validated by the experimental results in Section VI that show the W substantially outperforms S.

Rendle *et al* [11] discussed an adaptive k-Nearest-Neighbor method, which used the same model as in itemkNN in [2] but adaptively learn the item-item similarity matrix. However, the item-item similarity matrix in [11] is fully dense, symmetric and has negative values. W is different from Rendle *et al*'s item-item similarity matrix in that, in addition to its sparsity which leads to fast recommendation and low requirement for storage, W is not necessarily symmetric due to the optimization process and thus allows more flexibility for recommendation.

Paterek [15] introduced a linear model for each item for rating prediction, in which the rating of a user u_i on an item t_j is calculated as the aggregation of the ratings of u_i on all other items. They learned the aggregation coefficients (equivalent to W) by solving an ℓ_2-norm regularized least squares problem for each item. The learned coefficients are fully dense. The advantage of SLIM over Paterek's method is that ℓ_1-norm regularization is incorporated during learning which enforces W to be sparse, and thus, the most informative signals are captured in W while noises are discarded. In addition, SLIM learns W from all purchase/rating activities so as to better fuse information, compared to Paterek's method, which only uses a certain set of purchase/rating activities.

TABLE I: The Datasets Used in Evaluation

dataset	#users	#items	#trns	rsize	csize	density	ratings
ccard	42,067	18,004	308,420	7.33	17.13	0.04%	-
ctlg2	22,505	17,096	1,814,072	80.61	106.11	0.47%	-
ctlg3	58,565	37,841	453,219	7.74	11.98	0.02%	-
ecmrc	6,594	3,972	50,372	7.64	12.68	0.19%	-
BX	3,586	7,602	84,981	23.70	11.18	0.31%	1-10
ML10M	69,878	10,677	10,000,054	143.11	936.60	1.34%	1-10
Netflix	39,884	8,478	1,256,115	31.49	148.16	0.37%	1-5
Yahoo	85,325	55,371	3,973,104	46.56	71.75	0.08%	1-5

Columns corresponding to #users, #items and #trns show the number of users, number of items and number of transactions, respectively, in each dataset. Columns corresponding to rsize and csize show the average number of transactions for each user and on each item (i.e., row density and column density of the user-item matrix), respectively, in each dataset. Column corresponding to density shows the density of each dataset (i.e., density = #trns/(#users $\times$ #items)). Column corresponding to ratings shows the rating range of each dataset with granularity 1.

E. Relations between SLIM and MF Methods

MF methods for *top-N* recommendation have a model

$$\tilde{A} = UV^{\mathsf{T}}, \tag{5}$$

where U and V^{T} are the user and item factors, respectively. Comparing MF model in Equation 5 and the SLIM method in Equation 2, we can see that SLIM's model can be considered as a special case of MF model (i.e., A is equivalent to U and W is equivalent to V^{T})

U and V^{T} in Equation 5 are in a latent space, whose dimensionality is usually specified as a parameter. The "latent" space becomes exactly the item space in Equation 2, and therefore, in SLIM there is no need to learn user representation in the "latent" space and thus the learning process is simplified. On the other hand, U and V^{T} are typically of low dimensionality, and thus useful information may potentially get lost during the low-rank approximation of A from U and V^{T}. On the contrary, in SLIM, since information on users are fully preserved in A and the counterpart on items is optimized via the learning, SLIM can potentially give better recommendations than MF methods.

In addition, since both U and V^{T} in Equation 5 are typically dense, the computation of $\mathbf{a}_i^{\mathsf{T}}$ requires the calculation of each $\tilde{a}_{ij}$ from its corresponding dense vectors in U and V^{T}. This results in a high computational complexity for MF methods to do recommendations, that is, $O(k^2 \times n)$ for each user, where k is the number of latent factors, and n is the number of items. The computational complexity can be potentially reduced by utilizing the sparse matrix factorization algorithms developed in [16], [17], [18]. However, none of these sparse matrix factorization algorithms have been applied to solve *top-N* recommendation problem due to their high computational costs.

V. Materials

A. Datasets

We evaluated the performance of SLIM methods on eight different real datasets whose characteristics are shown in Table I. These datasets can be broadly classified into two categories.

The first category (containing ccard, ctlg2, ctlg3 and ecmrc [2]) was derived from customer purchasing transactions. Specifically, the ccard dataset corresponds to credit card purchasing transactions of a major department store, n which each card has at least 5 transactions. The ctlg2 and ctlg3 datasets correspond to the catalog purchasing transactions of two major mail-order catalog retailers. The ecmrc dataset corresponds to web-based purchasing transactions of an e-commerce site. These four datasets have only binary purchase information.

The second category (containing BX, ML10M, Netflix and Yahoo) contains multi-value ratings. All the ratings are converted to binary indications if needed. In particular, the BX dataset is a subset from the Book-Crossing dataset[1], in which each user has rated at least 20 items and each item has been rated by at least 5 users and at most 300 users. The ML10M dataset corresponds to movie ratings and was obtained from the MovieLens[2] research projects. The Netflix dataset is a subset extracted from the Netflix Prize dataset[3] such that each user has rated 20 – 250 movies, and each movie is rated by 20 – 50 users. Finally, the Yahoo dataset is a subset extracted from Yahoo! Music user ratings of songs, provided as part of the Yahoo! Research Alliance Webscope program[4]. In Yahoo dataset, each user has rated 20 – 200 songs, and each music has been rated by at least 10 users and at most 5000 users.

B. Evaluation Methodology & Metrics

We applied 5-time Leave-One-Out cross validation (LOOCV) to evaluate the performance of SLIM methods. In each run, each of the datasets is split into a training set and a testing set by randomly selecting one of the non-zero entries of each user and placing it into the testing set. The training set is used to train a model, then for each user a size-N ranked list of recommended items is generated by the model. The evaluation is conducted by comparing the recommendation list of each user and the item of that user in the testing set. In the majority of the results reported in Section VI, N is equal to 10. However, we also report some limited results for different values of N.

The recommendation quality is measured by the Hit Rate (HR) and the Average Reciprocal Hit-Rank (ARHR) [2]. HR is defined as follows,

$$\text{HR} = \frac{\text{\#hits}}{\text{\#users}}, \tag{6}$$

where #users is the total number of users, and #hits is the number of users whose item in the testing set is recommended (i.e., hit) in the size-N recommendation list. A second measure for evaluation is ARHR, which is defined as follows:

$$\text{ARHR} = \frac{1}{\text{\#users}} \sum_{i=1}^{\text{\#hits}} \frac{1}{p_i}, \tag{7}$$

where if an item of a user is hit, p is the position of the item in the ranked recommendation list. ARHR is a weighted version

[1] http://www.informatik.uni-freiburg.de/~cziegler/BX/
[2] http://www.grouplens.org/node/12
[3] http://www.netflixprize.com/
[4] http://research.yahoo.com/Academic_Relations

of HR and it measures how strongly an item is recommended, in which the weight is the reciprocal of the hit position in the recommendation list.

For the experiments utilizing ratings, we evaluate the performance of the methods by looking at how well they can recommend the items that have a particular rating value. For this purpose, we define the per-rating Hit Rate (rHR) and cumulative Hit Rate (cHR). The rHR is calculated as the hit rate on items that have a certain rating value. cHR is calculated as the hit rate on items that have a rating value which is no lower than a certain rating threshold.

Note that in the *top-N* recommendation literature, there exist other metrics for evaluation. Such metrics include area under the ROC curve (AUC), which measures relative positions of true positives and false positives in an entire ranked list. Variances of AUC can measure the positions in top part of a ranked list. Another popular metric is recall. However, in *top-N* recommendation scenario, we believe HR and ARHR are the most direct and meaningful measures, since the users only care if a short recommendation list has the items of interest or not rather than a very long recommendation list. Due to this we use HR and ARHR in our evaluation.

All the algorithms compared in Section VI are implemented in C. All the experiments are done on a Linux cluster with 6-core Intel Xeon X7542 "Westmere" processors at 2.66 GHz.

VI. Experimental Results

In this section, we present the performance of SLIM methods and compare them with other popular *top-N* recommendation methods. We present the results from two sets of experiments. In the first set of experiments, all the *top-N* recommendation methods use binary user-item purchase information during learning, and thus all the methods are appended by -*b* to indicate binary data used (e.g., SLIM-*b*) if there is confusion. In the second set of experiments, all the *top-N* recommendation methods use user-item rating information during learning, and correspondingly they are appended by -r if there is confusion.

We optimized all the C implementations of the algorithms to make sure that any time difference in performance is due to the algorithms themselves, and not due to the implementation. For all the methods, we conducted an exhaustive grid search to identify the best parameters to use. We only report the performance corresponding to the parameters that lead to the best results in this section.

A. SLIM *Performance on Binary data*

1) Comparison Algorithms: We compare the performance of SLIM with another three categories of *top-N* recommendation algorithms. The first category of algorithms is the item/user neighborhood-based collaborative filtering methods itemkNN, itemprob and userkNN. Methods itemkNN and userkNN are as discussed in Section II and Section IV-E. Method itemprob is similarity to itemkNN except that instead of item-item cosine similarity, it uses modified item-item transition probabilities. These methods are engineered with various heuristics for better performance[5].

[5]http://glaros.dtc.umn.edu/gkhome/suggest/overview

The second category of algorithms is the MF methods, including PureSVD and WRMF as discussed in Section II. Note that both PureSVD and WRMF use 0 values in the user-item matrix during learning. PureSVD is demonstrated to outperform other MF methods in *top-N* recommendation using ratings [3], including the MF methods which treat 0s as missing data. WRMF represents the state-of-the-art matrix factorization methods for *top-N* recommendation using binary information.

The third category of algorithms is the methods which rely on ranking/retrieval criteria, including BPRMF and BPRkNN as discussed in Section II and Section IV-E. It is demonstrated in [11] that BPRMF outperforms other methods in *top-N* recommendation in terms of AUC measure using binary information.

2) Top-N Recommendation Performance: Table II shows the overall performance of different *top-N* recommendation algorithms. These results show that SLIM produces recommendations that are consistently better (in terms of HR and ARHR) than other methods over all the datasets except ML10M (SLIM has HR 0.311 on ML10M, which is only worse than BPRkNN with HR 0.327). In term of HR, SLIM is on average 19.67%, 12.91%, 22.41%, 50.80%, 13.42%, 14.32% and 12.95% better than itemkNN, itemprob, userkNN, PureSVD, WRMF, BPRMF and BPRkNN, respectively, over all the eight datasets. Similar performance gains can also be observed with respect to ARHR. Among the three MF-based models, WRMF and BPRMF have similar performance, which is substantially better than PureSVD on all datasets except ML10M and Netflix. BPRkNN has better performance than MF methods on large datasets (i.e., ML10M, Netflix and Yahoo) but worse than MF methods on small datasets.

In terms of recommendation efficiency, SLIM is comparable to itemkNN and itemprob (i.e., the required times range in seconds), but considerably faster than the other methods (i.e., the required times range in minutes). The somewhat worse efficiency of SLIM compared to itemkNN is due to the fact that the resulting best W matrix is denser than the best performing item-item cosine similarity matrix from itemkNN. PureSVD, WRMF and BPRMF have worse computational complexities (i.e., linear to the product of the number of items and the dimensionality of the latent space), which is validated by their high recommendation run time. BPRkNN produces a fully dense item-item similarity matrix, which is responsible for its high recommendation time.

In terms of the amount of time required to learn the models, we see that the time required by itemkNN/itemprob is substantially smaller than the rest of the methods. The amount of time required by SLIM to learn its model, relative to PureSVD, WRMF, BPRMF and BPRkNN, varies depending on the datasets. However, even though SLIM is slower on some datasets (e.g., ML10M and Yahoo), this situation can be easily remediated by the feature-selection-based fsSLIM as will be discussed later in Section VI-A3.

One thing that is surprising with the results shown in Table II is that the MF-based methods are sometimes even worse than the simple itemkNN, itemprob and userkNN in terms of HR. For example, BPRMF performs worse for BX, ML10M, Netflix and Yahoo. This may because that in the BPRMF

TABLE II: Comparison of *Top-N* Recommendation Algorithms

method	ccard						ctlg2					
	params		HR	ARHR	mt	tt	params		HR	ARHR	mt	tt
itemkNN	50	-	0.195	0.145	0.54(s)	1.34(s)	10	-	0.222	0.108	33.78(s)	1.19(s)
itemprob	50	0.2	0.226	0.154	0.97(s)	1.24(s)	10	0.5	0.222	0.105	47.86(s)	0.99(s)
userkNN	150	-	0.189	0.122	0.06(s)	14.84(s)	50	-	0.204	0.106	0.37(s)	48.45(s)
PureSVD	3500	10	0.101	0.058	42.89(m)	2.65(h)	1300	10	0.196	0.099	3.95(m)	18.46(m)
WRMF	250	15	0.230	0.150	4.01(h)	9.14(m)	300	10	0.235	0.114	20.42(h)	5.49(s)
BPRMF	350	0.3	0.238	0.157	1.29(h)	6.64(m)	400	0.1	0.249	0.123	9.72(h)	3.14(m)
BPRkNN	1e-4	0.01	0.208	0.145	2.38(m)	8.15(m)	0.001	0.001	0.224	0.104	1.28(h)	22.03(m)
SLIM	5	0.5	**0.246**	0.170	17.24(m)	13.57(s)	5	2.0	**0.272**	0.140	7.24(h)	26.98(s)
fsSLIM	100	0.5	0.243	0.168	4.97(m)	4.45(s)	100	1.0	**0.282**	0.149	10.92(m)	5.00(s)
fsSLIM	50	0.5	0.244	0.169	2.40(m)	3.34(s)	10	0.5	0.262	0.138	4.21(m)	2.01(s)

method	ctlg3						ecmrc					
	params		HR	ARHR	mt	tt	params		HR	ARHR	mt	tt
itemkNN	300	-	0.544	0.313	0.55(s)	6.66(s)	300	-	0.218	0.125	0.06(s)	0.54(s)
itemprob	400	0.3	0.558	0.322	0.87(s)	7.62(s)	30	0.2	0.245	0.138	0.09(s)	0.12(s)
userkNN	350	-	0.492	0.285	0.11(s)	19.18(s)	400	-	0.212	0.119	0.01(s)	0.78(s)
PureSVD	3000	10	0.373	0.210	1.11(h)	4.28(h)	1900	10	0.186	0.110	3.67(m)	3.22(m)
WRMF	420	20	0.543	0.308	14.42(h)	50.67(m)	270	15	0.242	0.133	3.22(h)	13.60(s)
BPRMF	300	0.5	0.541	0.283	1.49(h)	13.66(m)	350	0.1	0.249	0.128	4.00(m)	12.76(s)
BPRkNN	0.001	1e-4	0.542	0.304	6.20(m)	20.28(m)	1e-5	0.010	0.242	0.130	1.02(m)	13.53(s)
SLIM	3	0.5	**0.579**	0.347	1.02(h)	16.23(s)	5	0.5	**0.255**	0.149	11.10(s)	0.51(s)
fsSLIM	100	0.0	0.546	0.292	12.57(m)	9.62(s)	100	0.5	0.252	0.147	16.89(s)	0.32(s)
fsSLIM	400	0.5	0.570	0.339	14.27(m)	12.52(s)	30	0.5	0.252	0.147	5.41(s)	0.16(s)

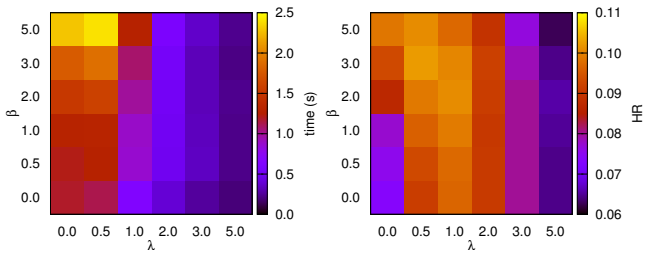
method	BX						ML10M					
	params		HR	ARHR	mt	tt	params		HR	ARHR	mt	tt
itemkNN	10	-	0.085	0.044	1.34(s)	0.08(s)	20	-	0.238	0.106	1.97(m)	8.93(s)
itemprob	30	0.3	0.103	0.050	2.11(s)	0.22(s)	20	0.5	0.237	0.106	1.88(m)	7.49(s)
userkNN	100	-	0.083	0.039	0.01(s)	1.49(s)	50	-	0.303	0.146	2.26(s)	34.42(m)
PureSVD	1500	10	0.072	0.037	1.91(m)	2.57(m)	170	10	0.294	0.139	1.68(m)	1.72(m)
WRMF	400	5	0.086	0.040	12.01(h)	29.77(s)	100	2	0.306	0.139	16.27(h)	1.59(m)
BPRMF	350	0.1	0.089	0.040	8.95(m)	12.44(s)	350	0.1	0.281	0.123	4.77(h)	5.20(m)
BPRkNN	1e-4	0.010	0.082	0.035	5.16(m)	42.23(s)	0.001	1e-4	**0.327**	0.156	15.78(h)	1.08(h)
SLIM	3	0.5	**0.109**	0.055	5.51(m)	1.39(s)	1	2.0	0.311	0.153	50.98(h)	41.59(s)
fsSLIM	100	0.5	0.109	0.053	36.26(s)	0.63(s)	100	0.5	0.311	0.152	37.12(m)	17.97(s)
fsSLIM	30	1.0	0.105	0.055	16.07(s)	0.18(s)	20	1.0	0.298	0.145	14.26(m)	8.87(s)

method	Netflix						Yahoo					
	params		HR	ARHR	mt	tt	params		HR	ARHR	mt	tt
itemkNN	150	-	0.178	0.088	24.53(s)	13.17(s)	400	-	0.107	0.041	21.54(s)	2.25(m)
itemprob	10	0.5	0.177	0.083	30.36(s)	1.01(s)	350	0.5	0.107	0.041	34.23(s)	1.90(m)
userkNN	200	-	0.154	0.077	0.33(s)	1.04(m)	50	-	0.107	0.041	18.46(s)	3.26(m)
PureSVD	3500	10	0.182	0.092	29.86(m)	21.29(m)	170	10	0.074	0.027	53.05(s)	11.18(m)
WRMF	350	10	0.184	0.085	22.47(h)	2.63(m)	200	8	0.090	0.032	16.23(h)	50.05(m)
BPRMF	400	0.1	0.156	0.071	43.55(m)	3.56(m)	400	0.1	0.093	0.033	10.36(h)	47.28(m)
BPRkNN	0.01	0.01	0.188	0.092	10.91(m)	6.12(m)	0.01	0.001	0.104	0.038	2.60(h)	4.11(h)
SLIM	5	1.0	**0.200**	0.102	7.85(h)	9.84(s)	5	0.5	**0.122**	0.047	21.30(h)	5.69(m)
fsSLIM	100	0.5	**0.202**	0.104	6.43(m)	5.73(s)	100	0.5	**0.124**	0.048	1.39(m)	41.24(s)
fsSLIM	150	0.5	**0.202**	0.104	9.09(m)	7.47(s)	400	0.5	**0.123**	0.048	2.41(m)	1.72(m)

Columns corresponding to params present the parameters for the corresponding method. For methods `itemkNN` and `userkNN`, the paramesters are number of neighbors, respectively. For method `itemprob`, the parameters are the number of neighbors and transition parameter α. For method `PureSVD`, the parameters are the number of singular values and the number of iterations during SVD. For method `WRMF`, the parameters are the dimension of the latent space and the weight on purchases. For method `BPRMF`, the parameters are the dimension of the latent space and learning rate, respectively. For method `BPRkNN`, the parameters are the learning rate and regularization parameter λ. For method `SLIM`, the parameters are ℓ_2-norm regularization parameter β and ℓ_1-norm regularization parameter λ. For method `fsSLIM`, the parameters are the number of neighbors and ℓ_1-norm regularization parameter λ. Columns corresponding to HR and ARHR present the hit rate and average reciprocal hit-rank, respectively. Columns corresponding to mt and tt present the time used by model learning and recommendation, respectively. The mt/tt numbers with (s), (m) and (h) are time used in seconds, minutes and hours, respectively. **Bold** numbers are the best performance in terms of HR for each dataset.

paper [11], the authors evaluated the entire AUC curve to measure if the interested items are ranked higher than the rest. However, a good AUC value does not necessarily lead to good performance on *top-N* of the ranked list. In addition, in the case of `PureSVD`, the best performance is achieved when a rather larger number of singular values is used (e.g., ccard, ctlg3, BX and Netflix).

3) `fsSLIM` *Performance:* Table II also presents the results for the `SLIM` version that utilizes feature selection (rows labeled as `fsSLIM`). In the first set of experiments we used the item-item cosine similarity (as in `itemkNN`) and for each column of A selected its 100 other most similar columns and used the $m \times 100$ matrix to estimate the coefficient matrix in Equation 3. The results are shown in the first `fsSLIM` row in Table II. In the second set of experiments we selected the most similar columns of A based on item-item cosine similarity or item-item probability similarity (as in `itemprob`), whichever performs best, and corresponding number of columns. The results of these experiments are shown in the second `fsSLIM` row.



(a) Recommendation time (b) HR

Fig. 1: ℓ_1-Norm and ℓ_2-Norm Regularization Effects on BX

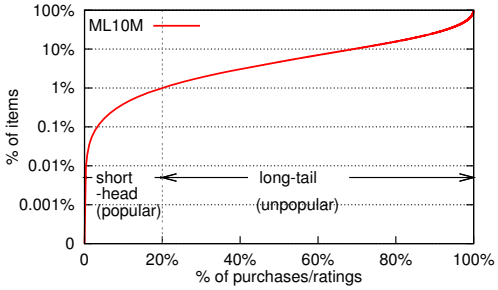


Fig. 2: Purchase/Rating Distribution in ML10M

There are three important observations that can be made from the `fsSLIM` results. First, the performance of `fsSLIM` is comparable to that of `SLIM` for nearly all the datasets. Second, the amount of time required by `fsSLIM` to learn the model is much smaller than that required by `SLIM`. Third, using `fsSLIM` to estimate W, whose sparsity structure is constrained by the `itemkNN`/`itemprob` neighbors, leads to significantly better recommendation performance than `itemkNN`/`itemprob` itself. This suggests that we can utilize feature selection to improve the learning time without decreasing the performance.

4) Regularization Effects in `SLIM` *:* Figure 1 shows the effects of ℓ_1-norm and ℓ_2-norm regularizations in terms of recommendation time (which directly depends on how sparse W is.) and HR on the dataset BX(similar results are observed from all the other datasets). Figure 1 demonstrates that as greater ℓ_1-norm regularization (i.e., larger λ in Equation 3) is applied, lower recommendation time is achieved, indicating that the learned W is sparser. Figure 1 also shows the effects of ℓ_1-norm and ℓ_2-norm regularizations together for recommendation quality. The best recommendation quality is achieved when both of the regularization parameters β and λ are non-zero. In addition, the recommendation quality changes smoothly as the regularization parameters β and λ change.

5) `SLIM` *for the Long-Tail Distribution:* The long-tail effect, which refers to the fact that a disproportionally large number of purchases/ratings are condensed in a small number of items (popular items), has been a concern for recommender systems. Popular items tend to dominate the recommendations, making it difficult to produce novel and diverse recommendations.

TABLE III: Performance on the Long Tail of ML10M

method	**ML10M** long tail					
	params		HR	ARHR	mt	tt
`itemkNN`	10	-	0.130	0.052	1.59(m)	4.62(s)
`itemprob`	10	0.5	0.126	0.051	1.65(m)	4.04(s)
`userkNN`	50	-	0.162	0.069	2.10(s)	20.43(m)
`PureSVD`	350	70	0.224	0.096	2.98(m)	10.45(m)
`WRMF`	100	2	0.232	0.097	23.15(h)	1.74(m)
`BPRMF`	300	0.01	0.240	0.102	22.63(h)	8.56(m)
`BPRkNN`	0.001	1e-4	0.239	0.098	15.72(h)	36.42(m)
`SLIM`	1	5.0	**0.256**	0.106	57.55(h)	47.69(s)
`fsSLIM`	10	5.0	0.255	0.105	25.37(m)	9.57(s)
`fsSLIM`	100	4.0	0.255	0.105	58.32(m)	19.32(s)

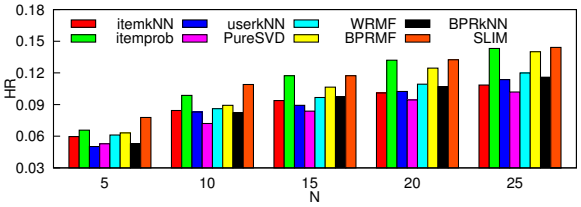
1% most popular items are eliminated from ML10M. Params have same meanings as those in Table II.

Since while constructing the BX, Netflix and Yahoo datasets, we eliminated the items that are purchased/rated by many users, these datasets do not suffer from long-tail effects. The results presented in Table II as they relate to these datasets demonstrate that `SLIM` is superior to other methods in producing non-trivial *top-N* recommendations when no significantly popular items are present.

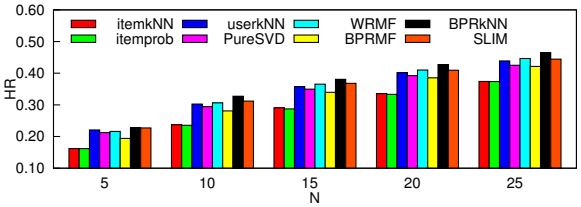
The plot in Figure 2 demonstrates the long-tail distribution of the items in ML10M dataset, in which only 1% of the items contribute 20% of the ratings. We eliminate these 1% most popular items and use the remaining ratings for all the *top-N* methods during learning. The results are presented in Table III. These results show that the performance of all methods is notably worse than the corresponding performance in Table II in which the "short head" (i.e., corresponding to the most popular items) is present. However, `SLIM` outperforms the rest methods. In particular, `SLIM` outperforms `BPRkNN` even though `BPRkNN` does better than `SLIM` as in Table II when the popular items are presented in ML10M. This conforms to the observations based on BX, Netflix and Yahoo results, that `SLIM` is resistant to the long-tail effects.

6) Recommendation for Different Top-N: Figure 3 shows the performance of the methods for different values of N (i.e., 5, 10, 15, 20 and 25) for BX, ML10M, Netflix and Yahoo datasets. Table IV presents the performance difference between `SLIM` and the best of the other methods in terms of HR on the four datasets. For example, 0.012 in Table IV for BX when $N = 5$ is calculated as the difference between `SLIM`'s HR and the best HR from all the other methods on BX when top-5 items are recommended. The performance difference between `SLIM` and the best of the other methods are higher for smaller values of N across the datasets BX, ML10M and Netflix. Figure 3 and Table IV demonstrate that `SLIM` produces better than the other methods when smaller number of items are recommended. This indicates that `SLIM` tends to rank most relevant items higher than the other methods.

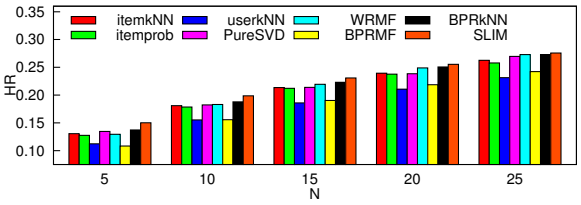
7) Sparsity Pattern of W : We use ML10M as an example to illustrate what `SLIM` is learning. The item-item similarity matrix S constructed from `itemkNN` and the W from `SLIM` are shown in Figure 4. Note that in Figure 4, the S matrix is obtained using 100 nearest neighbors. The density of the matrices produced by `itemkNN` and `SLIM` is 0.936% and 0.935%, respectively. However, their sparsity patterns are



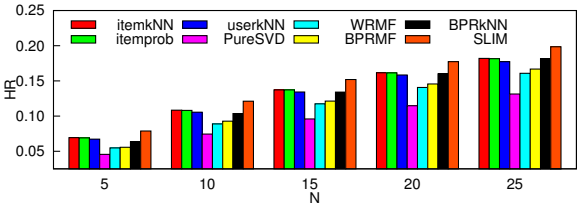
(a) BX



(b) ML10M



(c) Netflix



(d) Yahoo

Fig. 3: Recommendations for Different Values of N

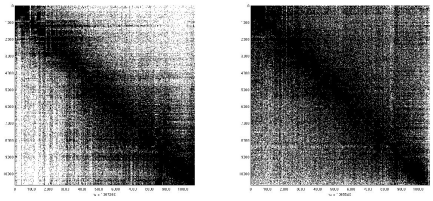
TABLE IV: Performance Difference on *Top-N* Recommendations

dataset	N				
	5	10	15	20	25
BX	0.012	0.006	0.000	0.000	0.001
ML10M	0.000	-0.016	-0.013	-0.018	-0.021
Netflix	0.013	0.012	0.008	0.005	0.003
Yahoo	0.009	0.015	0.015	0.016	0.017

Columns corresponding to N shows the performance (in terms of HR) difference between SLIM and the best of the rest methods on corresponding *top-N* recommendatons.

different. First, the S matrix has non-zero item-item similarity values that are clustered towards the diagonal, while W has non-zero values that are more evenly distributed. Second, during recommendation, on average 53.60 non-zero values in S contribute to the recommendation score calculation on one item for one user, whereas in case of W, on average 14.79 non-zero values make contributions, which is 1/3 as that in S.

W recovers 31.8% of the non-zero entries in S (those entries have larger values than average) and it also discovers new non-zero entries that are not in S. The newly discovered item-item similarities contribute to 37.1% of the hits from W. This



(a) S from itemkNN (b) W from SLIM

Fig. 4: Sparsity Patterns for ML10M (dark for non-zeros)

suggests that W, even though it is also very sparse, recovers some subtle relations that are not captured by item-item cosine similarity measure, which brings performance improvement. In SLIM, item t_k that are co-purchased with item t_j also contributes to the similarity between item t_j and another item t_i, even t_k has never been co-purchased with t_i. Furthermore, treating missing values as 0s helps to generalize. Including all missing values as 0s in w_j vector in Equation 4 help smooth out item similarities and help incorporate the impacts from dissimilar/un-co-purchased items. The above can be shown theoretically by the coordinate descent updates (proof omitted here).

8) Matrix Reconstruction: We compare SLIM with MF methods by looking at how it reconstructs the user/item purchase matrix. We use BPRMF as an example of MF methods since it has the typical properties that most of the state-of-the-art MF methods have. We focused on ML10M, whose matrix A has a density of 1.3%. The reconstructed matrix $\tilde{A}_{\texttt{SLIM}} = AW$ from SLIM has a density 25.1%, whose non-zero values have a mean of 0.0593. For those 1.3% non-zero entries in A, $\tilde{A}_{\texttt{SLIM}}$ recovered 99.1% of them and their mean value is 0.4489 (i.e., 7.57 times of the non-zero mean). The reconstructed matrix $\tilde{A}_{\texttt{BPRMF}} = UV^{\mathsf{T}}$ is fully dense, with 13.1% of its values greater than 0 with mean of 1.8636, and 86.9% of its values less than 0 with a mean of -2.4718. For those 1.3% non-zero entries in A, $\tilde{A}_{\texttt{BPRMF}}$ has 97.3% of them as positive values with a mean of 4.7623 (i.e., 2.56 times of positive mean). This suggests that SLIM recovers A better than BPRMF since SLIM recovers more non-zero entries with relatively much larger values.

B. SLIM *Performance on Ratings*

1) Comparison Algorithms: We compare the performance of SLIM with PureSVD, WRMF and BPRkNN. In SLIM, the W matrix is learned by using the user-item rating matrix A as in Equation 2. PureSVD also uses the user-item rating matrix for the SVD calculation. In WRMF, the ratings are used as weights following the approach suggested in [5]. We modified BPRkNN such that in addition to raking rated items higher than non-rated items, they also rank high-rated items higher than low-rated items. We will use the suffix *-r* after each method to explicitly denote that a method utilizes rating information during the model construction. Similarly, we will use the suffix

-b in this section to denote that a method utilizes binary information as in Section VI-A for comparison purpose.

2) Top-N Recommendation Performance on Ratings: We compare SLIM-r with PureSVD-r, WRMF-r and BPRkNN-r on the BX, ML10M, Netflix and Yahoo datasets for which rating information is available. In addition, we also evaluated SLIM-b, PureSVD-b, WRMF-b and BPRkNN-b on the four datasets, for which the models are still learned from binary user-item purchase matrix but the recommendations are evaluated based on ratings.

Figure 5 presents the results of these experiments. The first column of figures show the rating distribution of the four datasets. The second column of figures show the per-rating hit rates (rHR) for the four datasets. Finally, the third column of figures show the cumulative hit rates (cHR) for the four datasets. In Figure 5, the binary *top-N* models correspond to the best performing models of each method as in Table II. The results from *top-N* models using ratings are selected based on the cHR performance on rating 6, 6, 3 and 3 for the datasets, respectively.

The results in Figure 5 show that all the -r methods tend to produce higher hit rates on items with higher ratings. However, the per-rating hit rates of the -b methods have smaller dynamics across different ratings. This is because during learning, high-rated items and low-rated items are not differentiated in the -b methods. In addition, the -r methods outperform -b methods in terms of rHR on high-rated items. In particular, -r methods consistently outperform -b methods on items with ratings above the average across all the datasets.

Figure 5 also shows that the SLIM-r consistently outperforms the other methods in terms of both rHR and cHR on items with higher ratings over all the four datasets. In particular, it outperforms PureSVD-r in terms of cHR, which is demonstrated in [3] to be the best performing methods for *top-N* recommendation using ratings. This indicates that incorporating rating information during learning allows the SLIM methods to identify more highly-rated items.

VII. Discussion & Conclusions

A. Observed Data vs Missing Data

In the user-item purchase/rating matrix A, the non-zero entries represent purchase/rating activities. However, the entries with "0" value can be ambiguous. They may either represent that the users will never purchase the items, the users may purchase the items but have not done so, or we do not know if the users have purchased the items or not or if they will. This is the typical "missing data" setting and it has been well studied in recommender systems [4], [8].

In SLIM, we treated all missing data in $\mathbf{a}_j$ and A in Equation 4 as true negative(i.e., the users will never purchase the items). Differentiation of observed data and missing data in Equation 4 is under development.

B. Conclusions

In this paper, we proposed a sparse linear method for *top-N* recommendation, which is able to generate high-quality *top-N* recommendations fast. SLIM employs a sparse linear model in which the recommendation score for a new item can be calculated as an aggregation of other items. A sparse aggregation coefficient matrix W is learned for SLIM to make the aggregation very fast. W is learned by solving an ℓ_1-norm and ℓ_2-norm regularized optimization problem such that sparsity is introduced into W.

We conducted a comprehensive set of experiments and compared SLIM with other state-of-the-art *top-N* recommendation algorithms. The results showed that SLIM achieves prediction quality better that the state-of-the-art MF-based methods. Moreover, SLIM generates recommendations very fast. The experimental results also demonstrated the good properties of SLIM compared to other methods. Such properties include that SLIM is able to have significant speedup if feature selection is applied prior to learning. SLIM is also resistant to long-tail effects in *top-N* recommendation problems. In addition, when trained using ratings, SLIM tends to produce recommendations that are also potentially highly rated. Due to these properties, SLIM is very suitable for real-time *top-N* recommendation tasks.

Acknowledgement

This work was supported in part by NSF (IIS-0905220, OCI-1048018, and IOS-0820730), the DOE grant USDOE/DE-SC0005013 and the Digital Technology Center at the University of Minnesota. Access to research and computing facilities was provided by the Digital Technology Center and the Minnesota Supercomputing Institute.

References

[1] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds., *Recommender Systems Handbook*. Springer, 2011.

[2] M. Deshpande and G. Karypis, "Item-based top-n recommendation algorithms," *ACM Transactions on Information Systems*, vol. 22, pp. 143–177, January 2004.

[3] P. Cremonesi, Y. Koren, and R. Turrin, "Performance of recommender algorithms on top-n recommendation tasks," in *Proceedings of the fourth ACM conference on Recommender systems*, ser. RecSys '10. New York, NY, USA: ACM, 2010, pp. 39–46.

[4] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang, "One-class collaborative filtering," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 502–511.

[5] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 263–272.

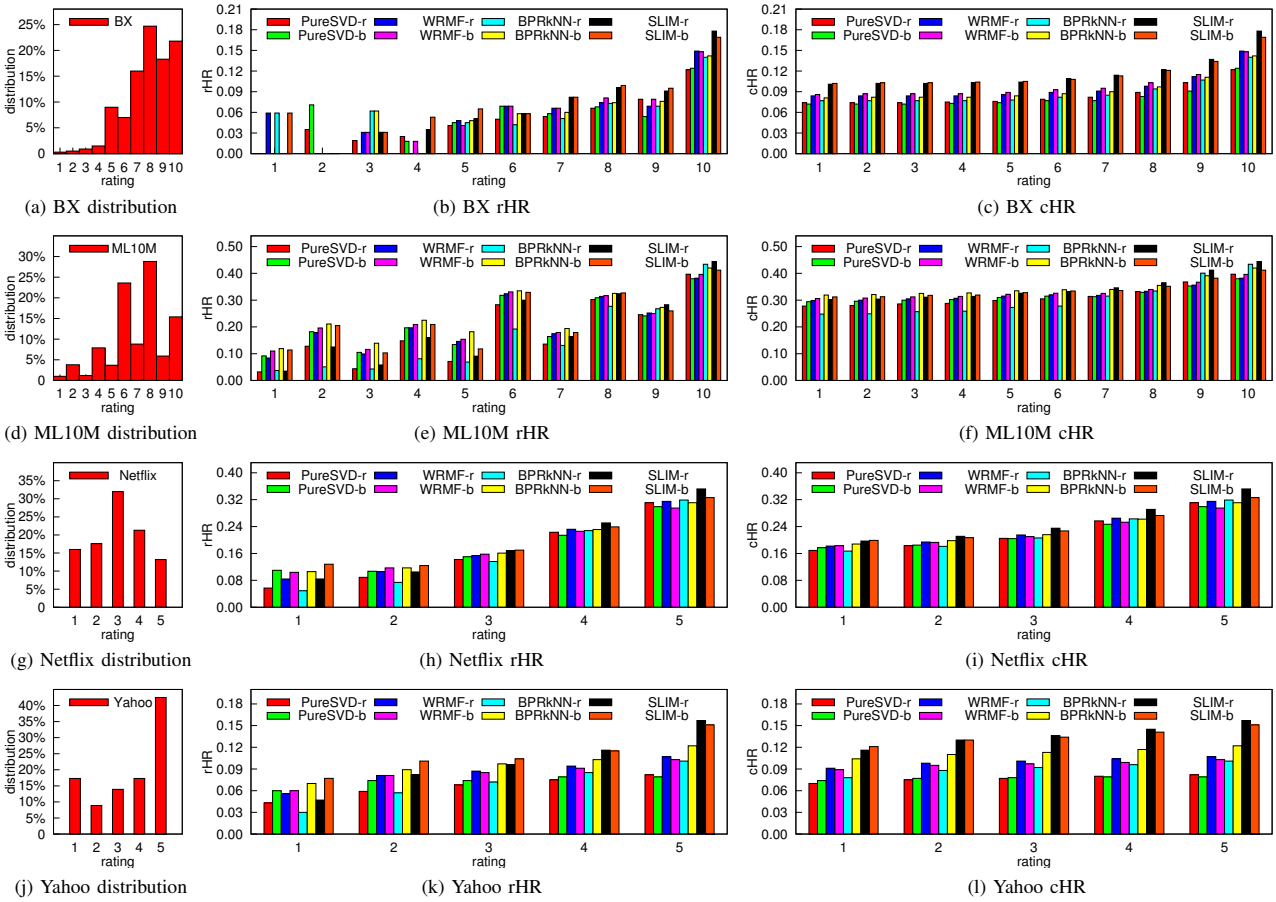
[6] J. D. M. Rennie and N. Srebro, "Fast maximum margin matrix factorization for collaborative prediction," in *Proceedings of the 22nd international conference on Machine learning*, ser. ICML '05. New York, NY, USA: ACM, 2005, pp. 713–719.

[7] N. Srebro, J. D. M. Rennie, and T. S. Jaakkola, "Maximum-margin matrix factorization," in *Advances in Neural Information Processing Systems 17*. MIT Press, 2005, pp. 1329–1336.

[8] V. Sindhwani, S. S. Bucak, J. Hu, and A. Mojsilovic, "One-class matrix completion with low-density factorizations," in *Proceedings of the 2010 IEEE International Conference on Data Mining*, ser. ICDM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1055–1060.

[9] T. Hofmann, "Latent semantic models for collaborative filtering," *ACM Trans. Inf. Syst.*, vol. 22, pp. 89–115, January 2004.

[10] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '08. New York, NY, USA: ACM, 2008, pp. 426–434.



(a) BX distribution (b) BX rHR (c) BX cHR

(d) ML10M distribution (e) ML10M rHR (f) ML10M cHR

(g) Netflix distribution (h) Netflix rHR (i) Netflix cHR

(j) Yahoo distribution (k) Yahoo rHR (l) Yahoo cHR

Fig. 5: *Top-N* Recommendation Performance on Ratings

[11] S. Rendle, C. Freudenthaler, Z. Gantner, and S.-T. Lars, "Bpr: Bayesian personalized ranking from implicit feedback," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, ser. UAI '09. Arlington, Virginia, United States: AUAI Press, 2009, pp. 452–461.

[12] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society (Series B)*, vol. 58, pp. 267–288, 1996.

[13] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal Of The Royal Statistical Society Series B*, vol. 67, no. 2, pp. 301–320, 2005.

[14] J. H. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of Statistical Software*, vol. 33, no. 1, pp. 1–22, 2 2010.

[15] A. Paterek, "Improving regularized singular value decomposition for collaborative filtering," *Statistics*, pp. 2–5, 2007.

[16] F. Bach, J. Mairal, and J. Ponce, "Convex sparse matrix factorizations," *CoRR*, vol. abs/0812.1869, 2008.

[17] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online learning for matrix factorization and sparse coding," *J. Mach. Learn. Res.*, vol. 11, pp. 19–60, March 2010.

[18] P. O. Hoyer, "Non-negative matrix factorization with sparseness constraints," *Journal of Machine Learning Research*, vol. 5, pp. 1457–1469, December 2004.