Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems

Badrul Sarwar[†][*], George Karypis[‡], Joseph Konstan[†], and John Riedl[†]
{sarwar, karypis, konstan, riedl}@cs.umn.edu
[†]GroupLens Research Group / [‡]Army HPC Research Center
Department of Computer Science and Engineering
University of Minnesota, Minneapolis, MN 55455, USA

Abstract

We investigate the use of dimensionality reduction to improve the performance for a new class of data analysis software called "recommender systems". Recommender systems apply knowledge discovery techniques to the problem of making personalized product recommendations during a live customer interaction. The tremendous growth of customers and products in recent years poses some key challenges for recommender systems. These are: producing high quality recommendations and performing many recommendations per second for millions of customers and products. Singular Value Decomposition(SVD)-based recommendation algorithms can quickly produce high quality recommendations, but has to undergo very expensive matrix factorization steps. In this paper, we propose and experimentally validate a technique that has the potential to incrementally build SVD-based models and promises to make the recommender systems highly scalable.

1 Introduction

Recommender systems have evolved in the extremely interactive environment of the Web. They apply data analysis techniques to the problem of helping customers find which products they would like to purchase at E-commerce sites. These systems, especially the collaborative filtering based ones [3, 5, 7, 8, 11], are rapidly becoming a crucial tool in E-commerce on the Web. Nowadays, they are being stressed by the huge volume of customer data in existing corporate databases, and will be stressed even more in future by the increasing volume of customer data available on the Web. The tremendous growth of customers and products poses two key challenges for recommender systems. The first challenge is to improve the quality of the recommendations for the consumers. Consumers need recommendations they can trust to help them find products they will like. If a consumer trusts a recommender system, purchases a product, and finds out he does not like the product, the consumer will be unlikely to use the recommender system again. Another challenge is to improve the scalability of the collaborative filtering algorithms. These algorithms are able to search tens of thousands of potential neighbors in real-time, but the demands of modern E-commerce systems are to search tens of millions of potential neighbors.

In some ways these two challenges are in conflict, since the less time an algorithm spends searching for neighbors, the more scalable it will be, and the worse its quality. For this reason, it is important to treat the two challenges simultaneously so the solutions discovered are both useful and practical. New technologies are needed that can dramatically improve the scalability of recommender systems. Researchers [1, 4, 9, 10] suggest that Singular Value Decomposition (SVD) may be such a technology in some cases. SVD-based approach produced results that were better than a traditional collaborative filtering algorithm most of the time when applied to a Movie data set [9]. However, SVD-based recommender systems suffer one serious limitation that makes them less suitable for large-scale deployment in E-commerce. The matrix factorization step associated with these systems is computationally very expensive and is a major stumbling block towards achieving high scalability.

In this paper, we experiment with an incremental model-building technique for generating SVD-based recommendations that has the promise of being highly scalable while producing good predictive accuracy. In particular, we present an algorithm that builds upon a small pre-computed SVD model and provides larger SVD models using inexpensive techniques. Our experimental results suggest that the overall algorithm works twice as fast while producing similar prediction accuracy.

The rest of the paper is organized as follows. The next section gives a brief overview of the SVD-based prediction generation process and discusses its promises and challenges. Section 3 outlines our incremental SVD algorithm. Section 4 presents our experimental procedure, results and discussion. The final

[*]Currently with the Computer Science Department, San Jose State University, San Jose, CA 95112, USA. Email: sarwar@cs.sjsu.edu, Phone: +1 408-245-8202

section provides some concluding remarks and future research directions.

2 Dimensionality Reduction for Collaborative Filtering

In this section we briefly discuss how one dimensionality reduction technique can potentially be used for prediction generation. We then present some of the challenges of such algorithms and propose an incremental technique to make them highly scalable.

2.1 Promises of Dimensionality Reduction

The goal of CF-based recommendation algorithms [7, 8, 11] is to suggest new products or to predict the utility of a certain product for a particular customer, based on the customer's previous liking and the opinions of other like-minded customers. These systems have been successful in several domains. However, in our earlier papers [9, 10], we mentioned some limitations of these systems, namely *sparsity, scalability,* and *synonymy.* The weakness of CF algorithms for large, sparse databases led us to explore alternative recommender system algorithms. After reviewing several techniques, we decided to try applying Latent Semantic Indexing (LSI) to reduce the dimensionality of our customer-product ratings matrix. LSI is a dimensionality reduction technique that has been widely used in information retrieval (IR) to solve the problems of synonymy and polysemy [2]. LSI, which uses *singular value decomposition (SVD)* as its underlying dimensionality reduction algorithm, maps nicely into the collaborative filtering recommender algorithm challenge.

Singular Value Decomposition (SVD). SVD is a matrix factorization technique commonly used for producing *low-rank* approximations. Given an $m \times n$ matrix A, with rank r, the singular value decomposition, $SVD(A)$, is defined as

$$SVD(A) = U \times S \times V^T \tag{1}$$

Where U, S and V are of dimensions $m \times m, m \times n$, and $n \times n$, respectively. Matrix S is a diagonal matrix having only r nonzero entries, which makes the effective dimensions of these three matrices $m \times r, r \times r$, and $r \times n$, respectively. U and V are two orthogonal matrices and S is a diagonal matrix, called the *singular matrix.* The diagonal entries $(s_1, s_2, \ldots, s_r)$ of S have the property that $s_i > 0$ and $s_1 \geq s_2 \geq \ldots \geq s_r$. The first r columns of U and V represent the orthogonal eigenvectors associated with the r nonzero eigenvalues of AA^T and A^TA, respectively. In other words, the r columns of U corresponding to the nonzero singular values span the *column space*, and the r columns of V span the *row space* of the matrix A. U and V are called the *left* and the *right* singular vectors, respectively.

SVD has an important property that makes it particularly interesting for our application. SVD provides the best *low-rank* linear approximation of the original matrix A. It is possible to retain only $k \ll r$ singular values by discarding other entries. We term this reduced matrix S_k. Since the entries in S are sorted i.e., $s_1 \geq s_2 \geq \ldots \geq s_r$, the reduction process is performed by retianing the first k singular values. The matrices U and V are also reduced to produce matrices U_k and V_k, respectively. The matrix U_k is produced by removing $(r-k)$ columns from the matrix U and matrix V_k is produced by removing $(r-k)$ rows from the matrix V. When we multiply these three reduced matrices, we obtain a matrix A_k. The reconstructed matrix $A_k = U_k.S_k.V_k^T$ is a *rank-k* matrix that is the closest approximation to the original matrix A. More specifically, A_k minimizes the *Frobenius norm* $||A - A_k||_F$ over all rank-k matrices. Researchers [1, 2] pointed out that the *low-rank* approximation of the original space is better than the original space itself due to the filtering out of the small singular values that introduce "noise" in the customer-product relatioship.

The dimensionality reduction approach in SVD can be very useful for the collaborative filtering process. SVD produces a set of uncorrelated eigenvectors. Each customer and product is represented by its corresponding eigenvector. The process of dimensionality reduction may help customers who rated similar products (but not exactly the same products) to be mapped into the space spanned by the same eigenvectors. We now present an outline of the prediction generation algorithm using SVD (see [9] for details).

Prediction generation using SVD. Once the $m \times n$ ratings matrix R is decomposed and reduced into three SVD component matrices with k features U_k, S_k, and V_k, prediction can be generated from it by computing the cosine similarities (dot products) between m pseudo-customers $U_k.\sqrt{S_k}^T$ and n pseudo-products $\sqrt{S_k}.V_k^T$ [1]. In particular, the prediction score $P_{i,j}$ for the i-th customer on the j-th product by adding the row average $\overline{r_i}$ to the similarity. Formally, $P_{i,j} = \overline{r_i} + U_k.\sqrt{S_k}^T(i).\sqrt{S_k}.V_k^T(j)$. Once the SVD decomposition is done, the prediction generation process involves only a dot product computation, which takes $\mathcal{O}(1)$ time, since k is a constant.

2.2 Challenges of Dimensionality Reduction

In a typical recommender system, the entire algorithm works in two separate steps. The first step is the *off-line* or *model-building* step and the second step is the *on-line* or the *execution* step. The user-user similarity computation and neighborhood formation [10] can be thought of as the off-line step of a CF system, whereas

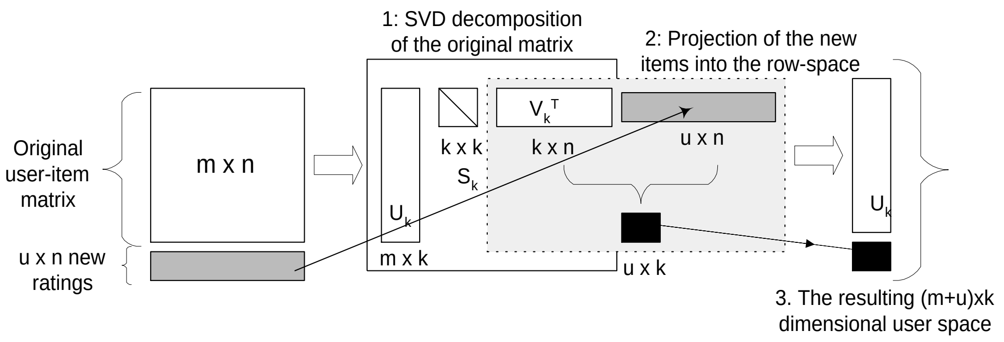
1: SVD decomposition of the original matrix
2: Projection of the new items into the row-space
Original user-item matrix
m x n
u x n new ratings
U_k
k x k
S_k
V_k^T
k x n
u x n
m x k
u x k
U_k
3. The resulting (m+u)xk dimensional user space

Figure 1: Schematic diagram of the SVD folding-in technique.

the actual prediction generation is the online step. Usually, the off-line component is very time-consuming and is computed relatively infrequently. For instance, an e-commerce site may compute the user-user similarity only once a day or even once a week. This works well if the ratings database is static and if the user behavior does not change significantly over a short period of time.

Researchers have demonstrated that the SVD-based algorithms can make the neighborhood formation process of CF systems highly scalable while producing better results in most of the cases [4, 9, 10]. Despite the good quality and excellent on-line performance SVD based algorithms suffer a serious drawback—the off-line SVD decomposition step is computationally very expensive. For an $m \times n$ user-item matrix, the SVD decomposition requires a run-time of $\mathcal{O}(m)^3$ [1, 2]. Our focus is to develop algorithms that ensure highly scalable overall performance. In order to achieve that goal, we must ensure that both the online and the off-line algorithms become more scalable. In the following section, we devise an algorithm that makes the off-line model building of SVD more scalable while achieving prediction quality comparable to the original SVD scheme.

3 Incremental SVD Algorithms

SVD has a property that allows the model to be incrementally computed. This method was used by the LSI researchers [1, 2] to handle dynamic databases, where new terms and documents may arrive once the model is built. It was shown that a projection of additional terms and documents can potentially provide a fairly good approximation of the model. We extend this idea to build a system where we first compute a suitably sized model and then use the projection method to build incrementally upon that. The resulting model is not a perfect SVD model as the space is not orthogonal, but the quality is expected to be good with potentially high performance gain.

The Algorithm. The projection technique is known as *folding-in* in SVD literature [1, 2]. To fold-in new users into the space of already reduced user-item matrix A_k, we compute the coordinates for that vector in the basis U_k. Let the size of the new user vector N_u be $t \times 1$. The first step in folding-in is to compute a projection P that projects N_u onto the space. Such a projection P of N_u is computed as:

$$P = U_k \times U_k^T \times N_u. \qquad (2)$$

This user set is then folded-in by appending the k dimensional vector $U_k^T.N_u$ as a new column of the $k \times d$ matrix $S_k.V_k^T$. Figure 1 shows a schematic diagram of the folding-in approach.

The folding-in technique allows us to devise a model-based approach for SVD-based prediction algorithms. Folding-in is based on the existing model (U_k, S_k, and V_k) and hence, new users or items do not affect existing user and items. In practice, it is possible to pre-compute the SVD decomposition using m existing users. For a user-item ratings matrix A, the three decomposed matrix U_k, S_k and V_k are computed at first. As described in the previous section, these matrices can be used for prediction generation. However, when a new set of ratings is added to the database, it is not necessary to re-compute the low-dimensional model from the scratch. We can take advantage of the folding-in technique to build an incremental system that has the potential to be highly scalable.

4 Experimental Evaluation

This section describes our experimental verification of the incremental SVD algorithm. We first present our experimental platform—the data set, the evaluation metric, and the computational environment. Then we present our experimental procedure followed by the results and discussion.

4.1 Experimental platform

Data set. We used data from MovieLens (`www.movielens.umn.edu`), which is our web-based research recommender system that debuted in Fall 1997. We randomly selected enough users to obtain $100,000$ ratings from the database (we only considered

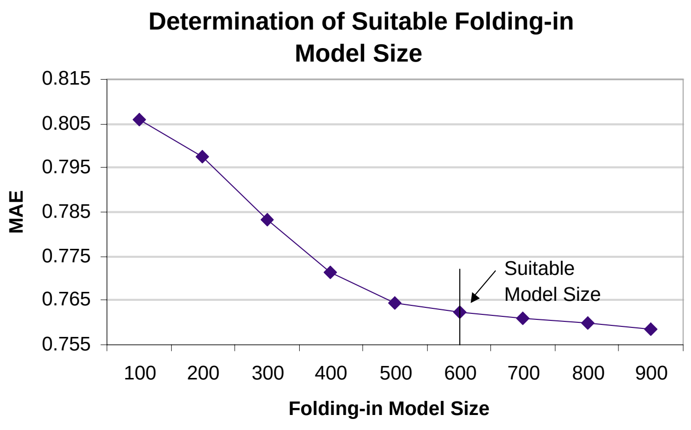

Figure 2: Determination of the threshold basis size for folding-in based SVD system

users that had rated 20 or more movies). The data set was converted into a user-movie matrix R that had 943 rows(users) and 1682 columns (movies). For our experiments, we divided the data set into a *training* and a *test* portion. We varied the training and test data ratio by using a parameter x, where $x = 0.8$ means that 80% data was used for training the algorithm and 20% was used as test.

Evaluation metric. For our experiments, we use a widely popular statistical accuracy metric named *Mean Absolute Error (MAE)*, which is a measure of the deviation of recommendations from their true user-specified values [5, 11]. For each ratings-prediction pair $< p_i, q_i >$, this metric treats the absolute error between them i.e., $|p_i - q_i|$ equally. The MAE is computed by first summing these absolute errors of the N corresponding ratings-prediction pairs and then computing the average. Formally, $MAE = \frac{\sum_{i=1}^{N} |p_i - q_i|}{N}$. The lower the MAE, the more accurately the recommendation engine predicts user ratings.

Environment. All our experiments were done using a combination of MATLAB and C, running on a Linux platform. The machine had 650 MHz Intel Pentium III CPU with 256 MB of RAM, and 512 KB of cache memory.

4.2 Experimental Procedure

For this experiment, we use the prediction generation algorithm using SVD described in [9], but instead of computing the SVD model (decomposition of matrix A into matrices U, S, and V) for all users, we use a *threshold* size to build an initial model and then use the folding-in technique to incrementally compute the SVD model for additional users. Before performing the prediction experiments, we first determine the optimal values of our two experimental parameters—*i)* the number of dimensions k, and *ii)* the threshold model size (basis size). We then perform the folding-in step and generate predictions using the incremental model. Finally, we investigate the performance implications of the folding-in technique. We used $10-$fold cross validation by selecting random training and test data for all our experiments.

4.2.1 Values of experimental parameters

Optimal value of k. To determine the optimal value of the number of dimension k, we performed an experiment where we generated predictions using different dimensions each time. We plotted our results and from there obtained that 14 is the suitable value of k. For the rest our experiments we use $k = 14$.

Optimal value of the basis size. Our goal is to select a basis size that is small enough to produce fast model building yet large enough to produce good prediction quality. If we start with a very small basis size, the entire model computation can be very fast, but due to non-orthogonal spaces the prediction quality may not be good. On the other hand a large basis size will defeat the purpose of incremental model building. We determined the optimal basis size through experiments, where we first fix a basis size and compute the SVD model by projecting the rest of $(total - basis)$ users using the folding-in technique. We start with a model size of 100 and go up to 900 with an increment of 100 at each step. Finally, we apply MAE to evaluate the prediction quality. We observe from Figure 2 that the quality of predictions improved as we increased the basis size. We further noticed that after the basis size crossed 600, the improvement in MAE values became relatively small. From this observation, we select 600 to be our threshold basis size.

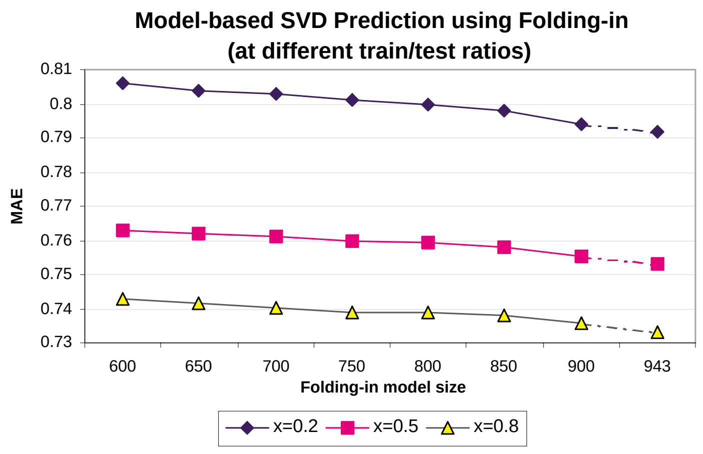

Figure 3: Prediction quality of folding-in based algorithms

4.2.2 Results and Discussion

Prediction quality experiments. Figure 4(a) plots the prediction quality results with the incremental model building using folding-in technique. We report our results at three different training-test ratios—at $x = 0.2, 0.5$, and 0.8. For each experiment, we start with a model size equal to the threshold basis size for folding-in. Then we use the projection method to fold-in the rest of $(943 - \text{basis})$ users onto the SVD space. This process gives us an approximate SVD model for 943 users, where unlike the original SVD model the component matrices U, S, and V are not orthogonal. We then use the component matrices to generate prediction for a test set of ratings. We start with a model size of 600 and go up to 900 with an increment of 50.

The plots of Figure 3 show that even with a small basis size it is possible to obtain a good quality. The MAE value at $x = 0.8$, for example, is 0.733 for the full model size and 0.742 for a model size of 600 (only 1.22% quality drop!). Similar numbers can also be found at other x values. This suggests that the inexpensive projection technique provides good quality even with a small basis size.

Performance implications The observation from Figure 4 that the quality does not change dramatically with varying model size suggests that the SVD prediction generation system can be made more scalable by using the folding-in method. To investigate these scalability impacts, we record the run-time in seconds for each run and from there, we compute the throughput performance metric. The throughput plot, presented in Figure 4(b), shows the number of predictions generated per second at different basis sizes. From the plot corresponding to $x = 0.8$ and the basis size of 600, we have a test case size of $(100,000 * (1 - 0.8))$. This means that our algorithm generates 20,000 ratings in 408.27 seconds, from there we obtain a throughput rate of 88.82 recommendations per second. Accordingly, at a full model size of 943 the throughput becomes 48.9 (81.63% performance gain!). This difference is even more prominent at lower values of x, where the workload size is larger.

Overall, the folding-in technique shows the potential to be very useful in addressing the scalability challenge of SVD-based prediction generation systems. We have demonstrated that although folding-in results in slight quality loss due to the non-orthogonality of the resultant space, it shows substantial performance gain.

5 Conclusion and Future Work

SVD-based recommendation generation technique leads to very fast online performance, requiring just a few simple arithmetic operations for each recommendation but computing the SVD is very expensive. Use of incremental SVD algorithms such as folding-in [1] can significantly speed up the SVD computation cost while providing comparable prediction quality. In this paper, we have demonstrated that incremental SVD algorithms, based on folding-in, can help recommender systems achieve high scalability while providing good predictive accuracy.

The folding-in technique requires less time and storage space but can result in the loss of quality due to the non-orthogonality of the incremental SVD space. Researchers [1, 2] have pointed out techniques to incrementally update the space by retaining the orthogonality. This method is known as the *SVD-update* and requires more time and memory than the folding-in technique. Zha *et al.* [12] points out that the updating technique described in [1] is, in fact, inaccurate and they provide modified and very complex mathematical technique to implement the updating technique they claim to be more accurate. Implementation of this technique remains as a future work.

Future work is required to understand exactly why SVD works well for some recommender applications,

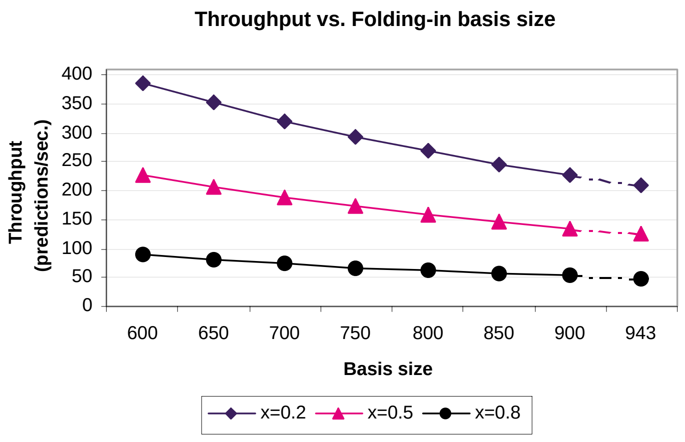

Figure 4: Throughput of the SVD folding-in algorithm

and less well for others. Also, there are many other ways in which SVD could be applied to recommender systems problems, including using SVD to create low-dimensional visualizations of the ratings space or using SVD to identify significant products that would help bootstrapping the recommender systems.

6 Acknowledgments

Funding for this research was provided in part by the National Science Foundation under grants IIS 9613960, IIS 9734442, and IIS 9978717 with additional funding by Net Perceptions Inc. This work was also supported by NSF CCR-9972519, by Army Research Office contract DA/DAAG55-98-1-0441, by the DOE ASCI program and by Army High Performance Computing Research Center contract number DAAH04-95-C-0008. Access to computing facilities was provided by AHPCRC and Minnesota Supercomputer Institute. We also thank anonymous reviewers for their valuable comments.

References

[1] Berry, M. W., Dumais, S. T., and O'Brian, G. W. (1995). Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review, 37(4).*

[2] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science.* 41(6).

[3] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM.* December.

[4] Gupta, D., and Goldberg, K. (1999). Jester 2.0: A Linear Time Collaborative Filtering Algorithm Applied to Jokes. In *Proc. of the ACM SIGIR '99.*

[5] Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J. (1999). An Algorithmic Framework for Performing Collaborative Filtering. In *Proc. of ACM SIGIR'99.*

[6] Hill, W., Stead, L., Rosenstein, M., and Furnas, G. (1995). Recommending and Evaluating Choices in a Virtual Community of Use. In *Proc. of CHI '95.*

[7] Resnick, P. and Varian, H. R. (1997). Recommender Systems. *Communications of the ACM.* 40(3), pp. 56-58.

[8] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proc. of CSCW '94*, Chapel Hill, NC.

[9] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. (2000). Application of Dimensionality Reduction in Recommender System—A Case Study. In *ACM WebKDD'00 (Web-mining for E-Commerce Workshop).*

[10] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. (2000). Analysis of Recommendation Algorithms for E-Commerce. In *Proc. of the ACM EC'00 Conference.* Minneapolis, MN, pp. 158-167.

[11] Shardanand, U., and Maes, P. (1995). Social Information Filtering: Algorithms for Automating 'Word of Mouth'. In *Proc. of CHI '95.* Denver, CO.

[12] Zha, H., and Zhang, Z. (1999). On matrices with Low-Rank-Plus-Shift Structure: Partial SVD and Latent Semantic Indexing. *SIAM Journal of Matrix Analysis and Applications*, vol. 21.