Isolating Matrix Sparsity in Collaborative Filtering Ratings Matrices

Brian J. Pechkis and Eun-Joo Lee
Computer Science Department, East Stroudsburg University of Pennsylvania, East Stroudsburg, PA, U.S.A.

Abstract— *Collaborative filtering is a widely-used class of methods for providing recommendations of items that are personalized to each individual user's tastes. Although effective at providing easy access to the ratings of user-item pairs, the matrix data structure typically utilized for these tasks is often very sparse. Furthermore, the scalability of these systems suffers when performing operations on a large matrix with mostly null values. This paper proposes a method of reducing the size and sparsity of these matrices by rearranging the order of the rows and columns in such a way that the ratings are clustered into small, easily extractable submatrices. Experimental tests on a representative dataset show that, in addition to achieving those goals, the quality of the predictions within the densest submatrix is high, as indicated by a reduced error rate compared with the full, unaltered matrix.*

Keywords: collaborative filtering, recommender systems, data mining, matrix sparsity

1. Introduction

The continued expansion of the World Wide Web has made it easier to access a wide variety of products for purchase via e-commerce and media content providers, such as Amazon.com and Netflix. However, the large quantity of these products has made it more difficult for users to find products that are both pertinent to their preferences and that are of good quality. For this reason, collaborative filtering recommender systems have been developed to select products, or items, for recommendation to users based on a comparison of their purchase habits to those of other users [1] [2]. These recommendations often rely upon predicting a user's preference value, or rating, on items not previously purchased by the user [3]. Some algorithms for performing this prediction are more statistical in nature (memory-based), while others draw on techniques developed in the field of machine learning (model-based) [2] [3] [4] [5].

One data structure for storing user ratings is the user-item matrix, which contains one dimension for users and another for items, storing the rating of one user for one item where the two intersect. This data structure makes it simple to extract a vector containing the information for one user or one item and lends itself to linear algebra-based feature extraction techniques. However, the matrix tends to be very sparse, due to the users' inability to rate all of the millions of items available to them [4] [6] [7]. Dimensionality reduction through singular value decomposition has been proposed as a solution to this problem, but it has a high complexity due to the need to either compute or estimate eigenvalues and eigenvectors [6]. Other imputation-based or default voting techniques bias a user's average rating toward the imputated or default rating, compromising the effectiveness of the prediction algorithms [8] [9].

Therefore, this paper proposes a method that can effectively reduce the size of the original matrix prior to any preprocessing of the data. It will be shown that rearranging the matrix by creating a permutation of it can create denser regions that can be extracted and used for any further processing and collaborative filtering prediction. Results also indicate that producing a submatrix that is denser than the full, unaltered matrix results incorporates the user-item pairs that will yield more accurate predictions. The primary goal of this method is to reduce the effective size of the matrix.

2. User-Item Matrix Permutations

The goal of rearranging the user-item matrix in a collaborative filtering recommender is to isolate the sparsity of the matrix from newly-formed dense regions. Achieving this requires two steps. First, metrics are computed for each row or column to determine which half of the vector is denser, before they are grouped with vectors having a similar center of density. Second, as an optional additional step, isolation of a greater percentage of sparsity is ensured by preventing one group of vectors (either denser in lower indices or denser in higher densities) from being much larger than the other. A flow chart showing the steps of this algorithm is shown in Figure 1. This procedure produces approximately equal-size submatrices that are simple to extract and to be processed individually by collaborative filtering prediction algorithms.

2.1 Permutation Algorithm

In order to form dense matrix regions, there needs to be some means by which to judge which segments of a row or a column of the matrix are denser than others. Simplifying this idea, it would be beneficial to determine whether a row or a column contains more ratings in its lower-indexed elements or its higher-indexed elements. Two simple metrics have been developed to quantify which of these two ends of a row or a column is denser. Given a specific row or column

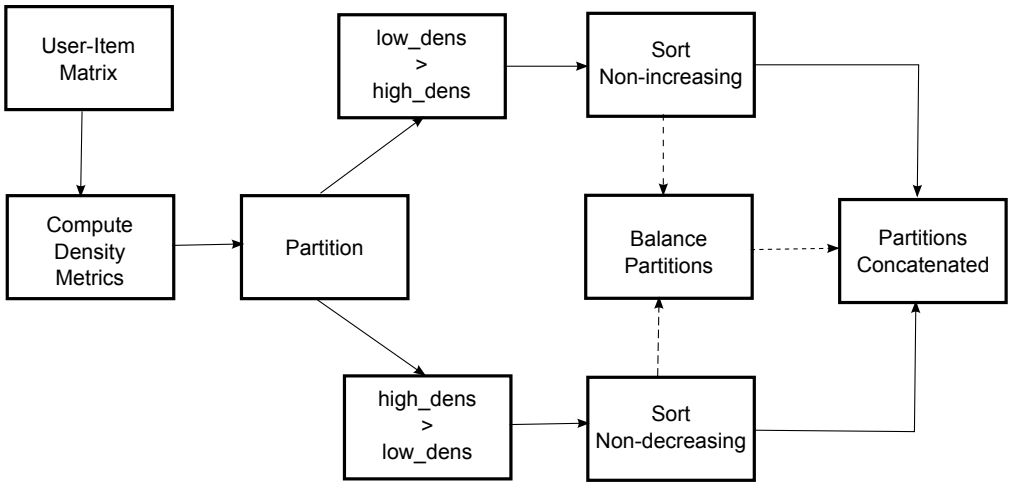

Fig. 1: Permutation algorithm flowchart

vector from the user-item matrix, v, of length n, we define Equation 1 and Equation 2.

$$low_{dens} = \sum_{i=1}^{n}((n+1)-1)^2 \quad \forall v(i) \neq null \tag{1}$$

$$high_{dens} = \sum_{i=1}^{n} i^2 \quad \forall v(i) \neq null \tag{2}$$

In Equation 1 , we see that, as $i \to \infty$, the terms in the summation become lower, resulting in higher overall values for vectors with larger numbers of lower-indexed ratings. Conversely, as $i \to \infty$, the terms within the summation in Equation 2 become higher, resulting in higher overall values for vectors containing more higher-indexed ratings. If these two metrics are computed for each of the rows and columns in the user-item matrix, a strong overall picture of the dense areas in the matrix is acquired.

This scheme of measuring vector density can be used to cluster vectors with similarly-located dense regions. Rows with greater density toward its lower-indexed elements can be grouped into one partition, and those with a greater density toward higher-indexed elements can be grouped into the other. Such a scheme can be duplicated for columns. In essence, this procedure examines each vector along one dimension of the matrix and decides which of the above-mentioned partitions to which it belongs. Then, it creates a permutation of the indices along the dimension by sorting the indices of vectors with a higher low_{dens} score by non-increasing order of low_{dens} metric and sorting the indices with a higher $high_{dens}$ score in non-decreasing order. If the dimension of interest is the rows of the matrix, we say that this algorithm performs a row permutation of the matrix, while it performs a column permutation if the columns of the matrix are considered.

The intent of this procedure is to create distinguishable regions with high numbers of non-null ratings in the matrix. A row permutation can be performed to create a matrix with a certain number of rows having more of its ratings to the left and a certain number of rows having more of its ratings to the right side of the matrix. A column permutation can be performed in addition to the row permutation to further solidify this pattern. The ultimate result is that we can find four clear regions in the matrix - two with a very high density of ratings and two with a rather sparse density of ratings. If these dense regions can be extracted from the matrix, the collaborative filtering algorithm can focus its efforts on them and have less exposure to sparsity, since many of the empty matrix elements have been moved to the sparse regions.

The efficiency of this algorithm comes from the fact that the ratings do not yet need to be in matrix form to perform the above-mentioned processing. It requires only a list of the non-zero elements of the matrix, which it adjusts once it has decided the proper ordering of the vectors along the specified dimension. When implemented in the C language, as was done for the tested implementation, the permutation algorithm becomes a simple exercise in list processing. Later, when processed as an actual matrix, the submatrices will exhibit the pattern described above. Therefore, assuming a dataset of w elements from an $m \times n$ matrix, this algorithm will only require w operations instead of the $m \times n$ needed to, at minimum, decide whether a matrix element is null or not. Although w could begin to approach $m \times n$ for denser matrices, this is unlikely to happen in the application of collaborative filtering. For instance, in the case of the MovieLens 100k dataset, $w = 100,000 << m \times n \approx 1.5M$. Therefore, the most computationally complex aspect of the algorithm is the sorting algorithm selected to sort the scores of the vectors in each partition. An efficient sorting algorithm designed for sorting numeric values can be utilized for this task.

2.2 Partition Balancing

The primary deficiency of the permutation algorithm is that it fails when faced with an imbalance in the data. For example, we may have many users (rows) who have rated more of the lower indexed items. Consequently, the balance of rows belonging to the low_{dens} partition will be very high. Should this same condition occur with the columns, we could be left with one very large submatrix and a few small ones. This would limit how many of the null values can be eliminated from consideration upon extraction of the submatrices. As a result, an add-on to this algorithm can help create a better balance between the partitions. After performing the initial partitioning of the vectors, the low_{dens} partition is sorted by non-decreasing order of $high_{dens}$ score or the $high_{dens}$ partition is sorted by non-increasing order of low_{dens} score, depending upon which has more member vectors. Then, the boundary marker is moved until there are an (approximately) even number of vectors in each partition. In effect, this process moves some of the vectors from the large partition that scores well in the other partition to that partition. The premise is that these vectors would still fit in well with the vectors of the other partitions, so it is acceptable to move them.

Once the permutation algorithm has been executed along both dimensions of the user-item matrix, the submatrices on which further processing and the collaborative filtering algorithms are run must be extracted. This process is simplified by the fact that the algorithm's execution has resulted in knowledge of where the boundary between the two density classes of vectors exists. If we divide the matrix along these boundary lines, it creates four submatrices of various densities that can be subjected to further preprocessing and to the collaborative filtering algorithms. Note that the full implementation of the permutation algorithm handles both the training and the corresponding test sets together. Once the submatrices are formed, the ratings that were part of the training set in the original matrix become part of the training set for the submatrix and likewise for the test ratings.

3. Experimental Setup

In addition to permutation algorithm described above, the system requires some additional components to constitute an operating collaborative filtering system. In this section, a description of the dataset used for the testing, as well as the preprocessing and collaborative filtering algorithms utilized are discussed.

3.1 Dataset

The chosen dataset for our testing was a widely-used, real-life dataset, the MovieLens 100k dataset [10]. It contains ratings data collected through the GroupLens research group's MovieLens movie recommendation website. The users of this website expressed their critique of various movies using a 1-5 scale, with 1 indicating a poor opinion of a movie and 5 indicating a high approval of the movie. The 100k dataset contains 943 unique users and 1682 unique movies (i.e. the "item" in this particular context), with its users submitting a total of 100,000 ratings out of a possible 1.59M ratings (a density of 6.3%). The MovieLens dataset contains a series of splits of training and test data, with each in the series containing a unique 20% subset used for test data, allowing for the use of 5-fold cross validation as the test method. This dataset meets the demands of testing collaborative filtering techniques by providing real-world data and, although not as large as many recommender systems, is large enough to discern the effects of the techniques in an operational setting.

3.2 Preprocessing

In collaborative filtering, it is rarely feasible to execute collaborative filtering algorithms on a raw set of ratings. These raw values usually are not numerous enough to be directly used to compute the Pearson correlations in a memory-based scenario nor do these alone serve as adequate features for training a model-based algorithm. As a result, a preprocessing of the data was needed to provide such features. The method used for preprocessing the training set parallels that described in [6]. Mean imputation is performed using the column mean, and then mean normalization is performed using the row mean. Once this was completed, the singular value decomposition could be computed for the matrix. Next, the element-wise square root of the matrix, S, was computed, so that any collaborative filtering prediction algorithm could use $U * \sqrt{S}$ as a set of features for each user and $\sqrt{S} * V^T$ could be used as a set of features for each item. Dimensionality reduction was delayed until execution of the collaborative filtering algorithms, so that the testing of multiple numbers of singular values to determine the permutation algorithm's effect on the optimal number of singular values. This procedure was performed in MATLAB, as its SVD routine was found to be faster than the one in the GNU Scientific Library used for implementing the collaborative filtering algorithms.

3.3 Collaborative Filtering Algorithms

After creating permutations of the user-item matrix, processing the matrix, and computing the singular value decomposition of the matrix, the process of producing predictions for each test rating in system can be performed. First, the training set for each submatrix is subjected to the training step for the algorithm while the test set is subjected to the prediction algorithm, producing the figures MAE and RMSE.

Three collaborative filtering algorithms were implemented. First, a user-based, memory-based algorithm was implemented that used the rows of $U * \sqrt{S}$ as the basis upon which Pearson correlation values were computed. The SVD-based algorithm described in [6] was also implemented. Fi-

Table 1: Percentage of non-null matrix elements (density) in submatrices extracted from permutation compared with density of original matrix

	No Balancing	Balancing
Unaltered	5.04%	-
Perm. submtx. 1	6.36%	3.36%
Perm. submtx. 2	3.11%	0.09%
Perm. submtx. 3	0.56%	14.92%
Perm. submtx. 4	8.41%	1.83%

Table 2: Number of non-null matrix elements in submatrices extracted from permutation compared with count from original matrix

	No Balancing	Balancing
Unaltered	80000.0	-
Perm. submtx. 1	68424.0	13390.4
Perm. submtx. 2	6006.4	368.8
Perm. submtx. 3	1502.4	59027.2
Perm. submtx. 4	4067.2	7213.6

nally, a linear regression algorithm that learns the parameter matrix, λ, in the formula $P = (U * \sqrt{S}) * \lambda$ through a process of gradient descent was implemented to examine a more traditional model-based method.

4. Experimental Results

4.1 Evaluation Criteria

The testing of the permutation-enhanced collaborative filtering system required metrics through which to judge its effectiveness. This process was performed in two steps. First, it was evaluated how well the permutation algorithm clustered the ratings into dense submatrices, effectively isolating the sparsity to specific submatrices. This could be judged using two metrics - the density of the submatrices (i.e. what percentage of the submatrices' elements were non-null) and the number of ratings contained in the submatrix. The second of these two helped judge how many of the non-null elements from the original matrix were contained in each of the submatrices.

In addition, it needed to be determined whether performing these permutations had any effect on the error of the predicted ratings made by the collaborative filtering algorithm. The well-established collaborative filtering error metric - the mean absolute error (MAE) - was computed as shown in Equation 3.

$$MAE = \frac{1}{n} \sum_{R_{i,j} \neq null} |P_{i,j} - R_{i,j}| \tag{3}$$

4.2 Submatrix Density

The first step in evaluating the above collaborative filtering system was to fine-tune the parameters of the permutation algorithm. These parameters included the order in which the two dimensions of the matrix were permutated, the number of iterations of the algorithm, and whether the partition balancing routine was used or not. The primary means of evaluation was the density of the submatrices and the number of ratings in each submatrix.

The densities of the submatrices when partition balancing was used and was not used are shown in Table 1. The submatrices are numbered from left-to-right and top-to-bottom according to their position in the matrix. As anticipated, the permutation created when partition balancing is not used creates one large submatrix with a much greater density than the original matrix, but its size is very small compared to the second-densest submatrix. In addition, this second-densest submatrix contains a majority of the ratings in the original dataset, as shown in Table 2. When partition balancing was not used, the sizes of the submatrices are more uniform, and the largest one still contains a large percentage of the ratings from the original matrix while the others have a smaller density. However, the second-densest one contains enough ratings that, if combined with the densest, the number of ratings approaches that of the large, dense matrix in the other test, except that the total number of matrix elements (both null and non-null) is less. Our tests revealed that the order in which the dimensions were permutated did not alter the densities of the submatrices.

The permutation algorithm was also tested with a number of iterations varying between 1 and 20. Since it was found that the densities of the submatrices did not change significantly after 5 iterations, numbers of iterations between 1 and 5 were also tested. Overall the change in density was minute with a changing number of iterations, varying only by a fraction of a percentage point after the first few iterations. However, it appeared that 10 iterations maximized the densities of the two densest submatrices, and therefore, this number of iterations was used for any further tests. The optimal number of iterations was the same for the case in which the partition balancing was not used.

4.3 Collaborative Filtering Prediction Error

Submatrices created by the permutation algorithm were subjected to the three collaborative filtering algorithms mentioned above. A wide range of numbers of singular values was tested. The row permutation was performed first, and only the matrices created using the partition balancing routine were tested, as it was a goal to test the error rate of the concatenation of the two densest submatrices.

Table 3 on the final page shows the error rate of the linear regression collaborative filtering algorithm on the submatrices with changing numbers of singular values. The values between 10-20 were tested since the results in [6], upon which the SVD algorithm is based, indicates a minimum MAE within this range. Higher values were also tested to

determine the difference in MAE with increasing numbers of singular values. The SVD-based algorithm demonstrated a similar trend in its results, although the MAE of submatrices 1 and 4 had a slightly higher MAE than in the linear regression case. It reveals a trend that, if the submatrix is of greater density than the original, full matrix, then the error rate will be lower, In addition, the concatenation of the two densest submatrices did not show an error rate that was as low as the densest submatrix alone. It appears that concatenating the second-densest submatrix (which by itself has a lower density than the original matrix) introduces enough sparsity back into the data that the error rates are slightly reduced. However, the one advantage is that these two submatrices permitted all of the users to be included in the combined dataset.

The memory-based algorithm was also tested with the same datasets. However, none of the submatrices had a lower error rate than the original matrix. This likely is due to the fact that memory-based algorithms compute predictions from the weighted averages of already-existing ratings. Therefore, predictions computed from a smaller subset of a user's or an item's ratings are based on less information, and the error rates are bound to be higher.

5. Conclusion

Creating permutations of a user-item matrix in a collaborative filtering recommender system has two primary benefits. First, the size of the matrix can be reduced by about half while losing only approximately 10% of the ratings in the original dataset. Furthermore, when such a permutation is created, the highest quality predictions are contained within the densest submatrix, as indicated by a lower error rate compared to the full, unaltered matrix. These more accurate predictions are ultimately more useful when making recommendations to the user. In conclusion, this method addresses two of the major drawbacks of collaborative filtering recommender systems - sparsity and scalability - in a simple algorithm.

References

[1] F. Ricci, L. Rokach, and B. Shapira, "Introduction to recommender systems handbook," in *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. U.S.: Springer, 2011, pp. 1–35.

[2] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," *IEEE Trans. Knowl. Data Eng.*, vol. 17, pp. 734–749, Jun. 2005.

[3] J. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," in *The Adaptive Web*, ser. Lecture Notes in Computer Science, P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds. Berlin/Heidelberg, Germany: Springer, 2007, vol. 4321, pp. 291–324.

[4] J. Zhou and T. Luo, "Towards an introduction to collaborative filtering," in *Int. Conf. Computational Science and Engineering*, vol. 4, 2009, pp. 576–581.

[5] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proc. 10th Int. Conf. World Wide Web*, ser. WWW '01. New York, NY, USA: ACM, 2001, p. 285âĂŞ295. [Online]. Available: http://doi.acm.org/10.1145/371920.372071

[6] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl, "Application of dimensionality reduction in recommender system - a case study," in *ACM Web KDD Workshop*, 2000.

[7] J. A. Konstan, J. Riedl, A. Borchers, and J. L. Herlocker, "Recommender systems: A grouplens perspective," in *Proc. 1998 Workshop Recommender Systems*, 1998.

[8] X. Su, T. M. Khoshgoftaar, and R. Greiner, "A mixture imputation-boosted collaborative filter," in *Proc. 21th Int. Florida Artificial Intelligence Research Society Conf.*, 2008, p. 312âĂŞ317.

[9] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proc. 14th Conf. Uncertainty and Artificial Intelligence*, ser. UAI'98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, p. 43âĂŞ52. [Online]. Available: http://dl.acm.org/citation.cfm?id=2074094.2074100

[10] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. T. Riedl, "An algorithmic framework for performing collaborative filtering," in *Proc. 1999 Conf. Research and Development Information Retrieval*, Aug. 1999.

Table 3: Prediction error of linear regression algorithm

Num. Singular Values	Unaltered	Perm. submtx. 1	Perm. submtx. 2	Perm. submtx. 3	Perm. submtx. 4	Perm. Submtx. 1&3
10	0.7658	0.8212	2.2212	0.7414	0.8329	0.7533
11	0.7656	0.8211	2.2227	0.7411	0.8324	0.7532
12	0.7654	0.8207	2.2216	0.7410	0.8324	0.7531
13	0.7653	0.8206	2.2211	0.7409	0.8324	0.7530
14	0.7651	0.8206	2.2216	0.7407	0.8323	0.7529
15	0.7651	0.8204	2.2214	0.7407	0.8323	0.7528
16	0.7649	0.8204	2.2210	0.7405	0.8322	0.7528
17	0.7648	0.8203	2.2217	0.7404	0.8321	0.7527
18	0.7647	0.8202	2.2214	0.7404	0.8319	0.7527
19	0.7646	0.8202	2.2199	0.7403	0.8320	0.7526
20	0.7645	0.8200	2.2197	0.7402	0.8318	0.7526
30	0.7639	0.8198	2.2218	0.7398	0.8306	0.7522
40	0.7636	0.8193	2.2194	0.7396	0.8294	0.7521
50	0.7632	0.8190	2.2198	0.7394	0.8283	0.7520
60	0.7629	0.8192	2.2206	0.7390	0.8273	0.7518
70	0.7626	0.8192	2.2198	0.7388	0.8267	0.7517
80	0.7625	0.8191	2.2198	0.7387	0.8259	0.7516
90	0.7623	0.8192	2.2198	0.7386	0.8252	0.7516
100	0.7622	0.8193	2.2198	0.7385	0.8247	0.7515