Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 99-013

Applying Knowledge from KDD to Recommender Systems

Badrul Sarwar, Joseph Konstan, Al Borchers, and John Riedl

April 18, 1999

Applying Knowledge from KDD to Recommender Systems

Badrul M. Sarwar, Joseph A. Konstan, Al Borchers, John T. Riedl

Department of Computer Science and Engineering

University of Minnesota, Minneapolis

+1 612 625-4002

{sarwar,konstan,borchers,riedl}@cs.umn.edu

Abstract

We investigate a new class of software for knowledge discovery in databases (KDD), called *recommender systems*. Recommender systems apply KDD-like techniques to the problem of making product recommendations during a live customer interaction. These systems are achieving widespread success in E-Commerce today. We extend previously studied KDD models to incorporate customer interaction so these models can be used to describe both traditional KDD and recommender systems. Recommender systems face three key challenges: producing high quality recommendations, performing many recommendations per second for millions of customers and products, and achieving high coverage in the face of data sparsity. One successful recommender system technology is *collaborative filtering*, which works by matching customer preferences to other customers in making recommendations. Collaborative filtering has been shown to produce high quality recommendations, but the performance degrades with the number of customers and products. New recommender system technologies are needed that can quickly produce high quality recommendations, even for very large-scale problems. For example, traditional KDD techniques might be applied in the context of our model to address these challenges. We have explored one technology called *Singular Value Decomposition (SVD)* to reduce the dimensionality of recommender system problems. We report an experiment where we use SVD on a recommender system database, and use the relationship between customers in the reduced factor space to generate predictions for products. We observe significant improvement in prediction quality as well as better online performance and improved coverage. Our experience suggests that SVD has the potential to meet many of the challenges of recommender systems.

Introduction

Knowledge discovery in databases (KDD). The goal in the research community has been exactly that: discover knowledge in the enormous databases collected by every modern corporation (Fayyad et al. 1996). The knowledge discovered has most often been concept learning or clustering (Zytkow 1997). Though the research techniques are often subtle, their application in business has two unsubtle goals. They are to save money by discovering the potential for efficiencies, or to make more money by discovering ways to sell more products to customers. For instance, companies are using KDD to discover which products sell well at which times of year, so they can manage their retail store inventory more efficiently, potentially saving millions of dollars a year (Brachman et al. 1996). Other companies are using KDD to discover which customers will be most interested in a special offer, reducing the costs of direct mail or outbound telephone campaigns by hundreds of thousands of dollars a year (Bhattacharyya 1998, Ling et al. 1998). These applications typically involve using KDD to discover a new model, and having an analyst apply the model to the application.

KDD has been successfully applied to many aspects of business data processing, including inventory management, product planning, manufacturing, and recommending products to customers. In most of these domains the benefit of KDD is to save money by improving efficiencies. For instance, in using KDD for product planning, the models can be used to focus development effort on products that are more likely to be purchased by consumers. Improving the focus of product development reduces the expenses of creating eventually unprofitable products, and shortens the costly product development cycle. However, the most direct benefit of KDD to businesses is increasing sales of existing products by matching customers to the products they will be most likely to purchase. Since our focus in this paper is on

recommender systems that have evolved on the Web primarily in support of E-Commerce, we will focus on this type of KDD system.

KDD systems that are *used* to match products to customers we will call *KDD marketing systems*. Figure 1 shows the flow of information in a typical KDD marketing system, derived from a general KDD flow diagram (Fayyad et al., 1996). In the KDD system, data is brought together from multiple corporate databases into a warehouse. In the warehouse the data is analyzed using data mining tools, creating models for human analysis. Human analysts view and manipulate the models on workstations, creating knowledge in the form of understanding of the data by the humans, and refined models in the system. The new knowledge and refined models are used to modify the behavior of existing marketing systems, or to implement new marketing systems. Those marketing systems that are involved in directly interacting with customers we call *touchpoint systems*.

Recommender systems have evolved in the extremely interactive environment of the Web. They apply KDD techniques to the problem of helping customers find which products they would like to

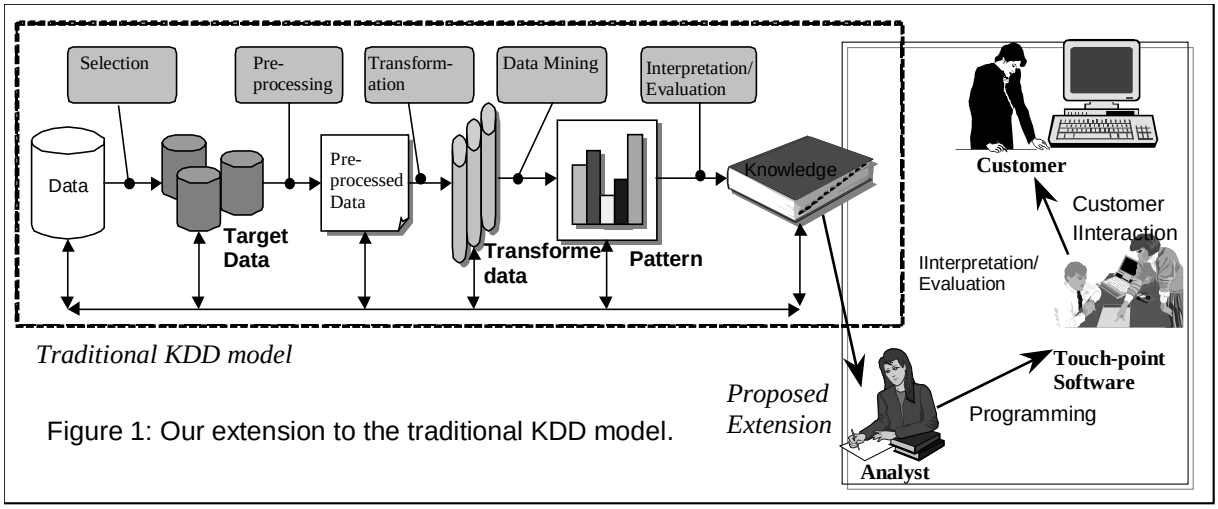


Figure 1: Our extension to the traditional KDD model.

purchase at E-Commerce sites. For instance, a recommender system on Amazon.com (www.amazon.com) suggests books to customers based on other books the customers have told Amazon they like. Another recommender system on

CDnow (www.cdnow.com) helps customers choose CDs to purchase as gifts, based on other CDs the recipient has

liked in the past. The Web presents new opportunities for KDD, but challenges KDD systems to perform interactively. While a customer is at the E-Commerce site, the recommender system must learn from the customer's behavior, develop a model of that behavior, and apply that model to recommend products to the customer. *Collaborative filtering* is the most successful recommender system technology to date, and is used in many of the most successful recommender systems on the Web, including those at Amazon.com and CDnow.com.

The earliest implementations of collaborative filtering, in systems such as Tapestry (Goldberg et al., 1992), relied on the opinions of people from a close-knit community, such as an office workgroup. However, collaborative filtering for large communities cannot depend on each person knowing the others. Several systems use statistical techniques to provide personal recommendations of documents by finding a group of other users, known as *neighbors* that have a history of agreeing with the target user. Usually, neighborhoods are formed by applying proximity measures such as the Pearson correlation between the opinions of the users. These techniques are called *nearest-neighbor techniques.* Figure 2 depicts the neighborhood formation using a nearest-neighbor technique in a very simple two dimensional space of users. Notice that each user's neighborhood is those other users who are most similar to him, as identified by the proximity measure. Neighborhoods need not be symmetric. Each user has the best neighborhood for him. Once a neighborhood of users is found, particular products can be evaluated by forming a weighted composite of the neighbors' opinions of that document.

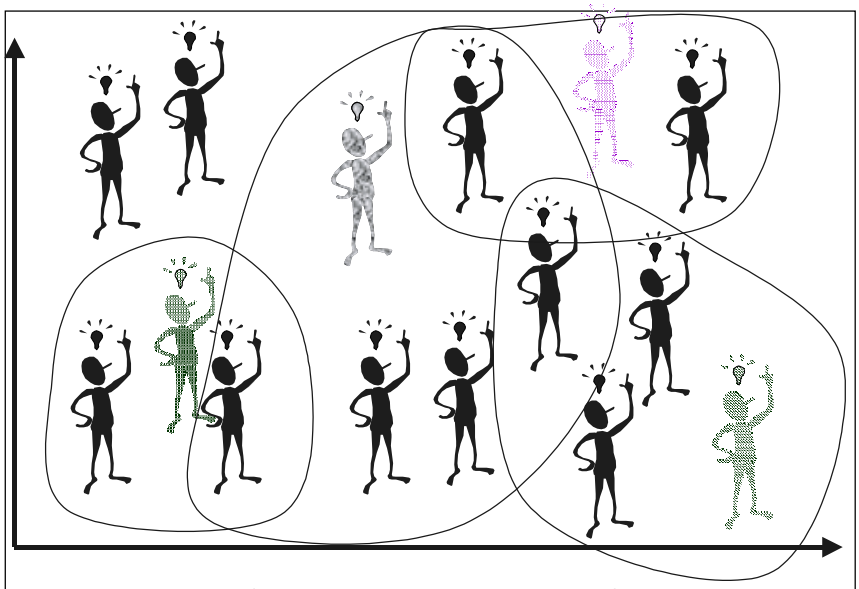


Figure 2. Neighborhood formation. Each neighborhood is created for a single customer.

These statistical approaches, known as *automated collaborative filtering*, typically rely upon *ratings* as numerical expressions of user preference. Several ratings-based automated collaborative filtering systems have been developed. The GroupLens Research system (Resnick et al. 1994) provides an pseudonymous collaborative filtering solution for Usenet news and movies. *Ringo* (Shardanand et al. 1995) and *Video Recommender* (Hill et al. 1995) are email and web systems that generate recommendations on music and movies

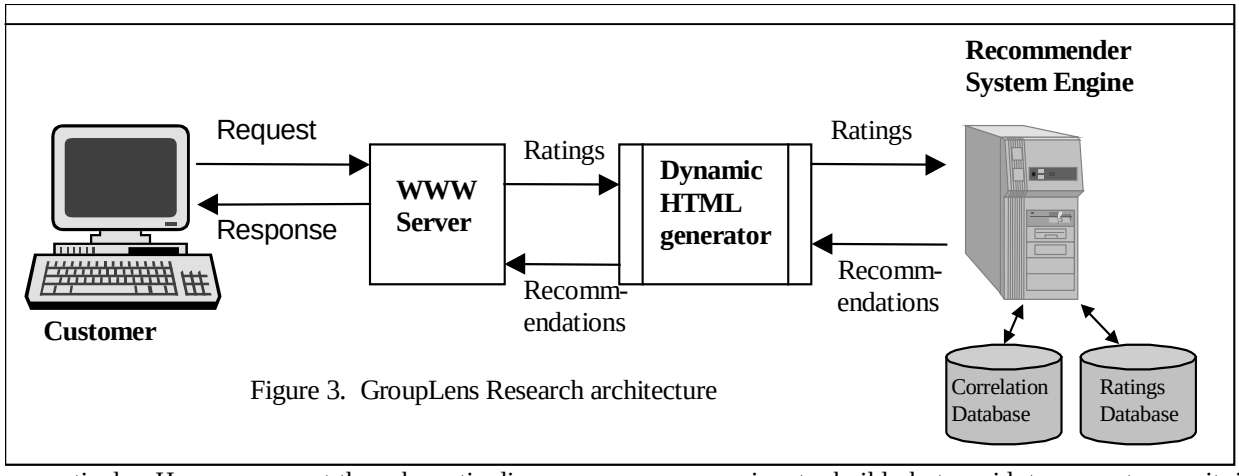


Figure 3. GroupLens Research architecture

respectively. Here we present the schematic diagram of the architecture of the GroupLens Research collaborative filtering engine in figure 3. The user interacts with a Web interface. The Web server software communicates with the recommender system to choose products to suggest to the user. The recommender system, in this case a collaborative filtering system, uses its database of ratings of products to form neighborhoods and make recommendations. The Web server software displays the recommended products to the user.

In traditional KDD systems the interface between the KDD system and the customer touchpoint is mediated by an analyst. The algorithms used in KDD develop as their output high-level data structures, such as Bayesian networks, classifier functions, rules bases, or data clusters (Heckerman, 1996; Cheeseman, 1990; Agarwal et al., 1993; Fayyad et al., 1996). Typically the model is expensive to build, but rapid to execute, so it is recomputed only after sufficient changes have occurred in the database. (Ongoing work is developing incremental KDD algorithms, but these are currently rare in practice.) The model, once produced, is high-level, powerful, and abstract, so its

interface to the touchpoint software is mediated by a human.

Recommender systems use very different interfaces, typically clustered into an API that can be used directly by the touchpoint software. The most common API calls are:

1. Recommend. Given a customer, recommend a list of products that customer will be interested in.
2. Predict. Given a customer, and a list of potential products, predict which of those products the customer will be interested in. The input list of products is the difference between recommend and predict. The list might have been produced as the result of a customer search, for instance.
3. Rate. Express an opinion of a customer about a product.
4. Find neighbors. Return a list of the nearest neighbors of a customer, for community applications, such as chat groups.

Recommender system APIs are simpler, concrete, and efficient, so they can be directly implemented in the touchpoint software.

The largest Web sites operate at a scale that stresses the direct implementation of collaborative filtering. Model-based techniques, such as those developed by KDD researchers, have the potential to contribute to recommender systems that can operate at the scale of these sites. However, these techniques must be adapted to the real-time needs of the Web, and they must be tested in realistic problems derived from Web access patterns. We are currently running MovieLens Web site, a recommender system research Web site. This site provides an excellent test-bed for recommender system algorithms, since we have a large repository of historical usage data, and a large community of continuing users. We are using MovieLens to test new recommender system algorithms and user interfaces. The present paper describes our experimental results in applying a model-based technique, Latent Semantic Indexing (LSI), that uses a dimensionality reduction technique, Singular Value Decomposition (SVD), to our recommender system.

Relationship of Recommender Systems to other KDD Systems

There are several types of KDD systems that seem at first sight to have very similar goals to recommender systems, but that are actually quite different in practice. This section briefly discusses these systems, and explains their relationship to recommender systems.

Online Analytic Processing (OLAP) systems enable the analyst to look at the database in different cross-sections while the database is online. OLAP is most often applied to systems that enable rapid analysis of multidimensional databases. These systems do not automatically build models, but assist analysts in exploring possible models (Uthurusamy 1996).

Online KDD systems refer to KDD systems that enable the analyst to interactively participate in the creation of the model. For instance, one such system develops association rules in conjunction with the analyst (Aggarwal, Su, and Yue 1998). These systems have online interaction with the analyst, but the analyst still must take the resulting model and separately integrate it with the touchpoint software.

Interactive mining has been used to refer to using "human inspection and guidance at intermediate stages" of the data mining process (Zytkow 1991). These systems are closely related to online KDD systems, but do not require that the inspection and guidance be interactive.

Contributions

The contributions of this paper are:

1. A model for KDD that includes both traditional KDD and emerging recommender systems.
2. An explanation of how model-based technologies, such as those used in KDD can fit into recommender systems.
3. The details of how one model-based technology, LSI/SVD, was applied in a recommender systems.
4. The results of our experiments with LSI/SVD on our MovieLens test-bed.

Other than the introduction and conclusion, the body of the paper is laid out in two sections:

1. How we apply LSI/SVD to Recommender Systems.
2. Our experimental test-bed, design, and results.

Applying SVD for Collaborative Filtering in Recommender Systems

Background

Most recommender systems based on collaborative filtering have used the *weighted average of nearest neighbors* method, using Pearson correlation as a measure of proximity (Shardanand et al. 1995, Resnick et al. 1994). These systems have been successful in several domains, but the algorithm is not well suited to large, sparse ratings databases (Billsus et al. 1998) Pearson neighbor algorithms require computation that grows with both the number of customers and the number of products. Furthermore, by relying upon exact matches, the algorithms may sacrifice recommender system coverage and accuracy. In particular, since the correlation coefficient is only defined between customers who have rated at least two products in common, many pairs of customers have no correlation at all. In practice, many commercial recommender systems are used to evaluate large product sets (e.g., Amazon.com recommends books and CDnow recommends music albums). In these systems, even active customers may have rated well under 1% of the products (1% of 2 million books is 20,000 books--a large set on which to have an opinion). Accordingly, Pearson nearest neighbor algorithms may be unable to make many product recommendations for a particular user. This problem is known as *reduced coverage*, and is due to sparse ratings of neighbors. Furthermore, the accuracy of recommendations may be poor because fairly little ratings data can be included. An example of a missed opportunity for quality is the loss of neighbor transitivity. If customers Pete and Sue correlate highly, and Sue also correlates highly with Paul, it is not necessarily true that Pete and Paul will correlate. They may have too few ratings in common or may even show a negative correlation due to a small number of unusual ratings in common.

The weakness of Pearson nearest neighbor for large, sparse databases led us to explore alternative recommender system algorithms. Our first approach attempted to bridge the sparsity by incorporating semi-intelligent filtering agents into the system (Sarwar et al. 1998). These agents evaluated and rated each product, using syntactic features. By providing a dense ratings set, they helped alleviate coverage problems. Quality also improved as we programmed agents to look for features that matched some customer tastes. The filtering agent solution, however, did not address the fundamental problem of poor relationships among like-minded but sparse-rating customers.

This paper reflects our second effort to address sparsity problems. We recognized that the KDD research community had extensive experience learning from sparse databases. After reviewing several KDD 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* (Deerwester et al. 1990). Given a term-document-frequency matrix, LSI is used to construct two matrices of reduced dimensionality. In essence, these matrices represent latent attributes of terms, as reflected by their occurrence in documents, and of documents, as reflected by the terms that occur within them. IR researchers often use reduction to dimension 2 or 3 to allow the space to be explored graphically. In the new 2 or 3 dimension space related terms and documents appear closer together.

LSI maps nicely into the collaborative filtering recommender algorithm challenge. We are trying to capture the relationships among pairs of customers based on ratings of products. By reducing the dimensionality of the product space, we can increase density and thereby find more ratings. Intuitively, this is analogous to "discovering" that a wide set of products are treated similarly by customers (e.g., customers tend to like or dislike 100% recycled office supplies); by recognizing this similarity, we can find agreement among customers who may have no single product in common (e.g., one may have rated recycled letter-sized pads and another may have rated recycled memo pads).

LSI, which uses singular value decomposition as its underlying matrix factorization algorithm, seemed particularly promising. Berry et al. (1995) point out that the reduced orthogonal dimensions resulting from SVD are less noisy than the original data and capture the latent associations between the terms and documents. Earlier work (Billsus et al. 1998) took advantage of this semantic property to reduce the dimensionality of feature space. The reduced feature space was used to train a neural network to generate predictions. We decided

to build a LSI-based personal recommender algorithm that uses SVD to reduce dimensionality and then computes the inner product of the reduced matrices to generate predictions. The rest of this section presents the construction of an SVD-based recommender algorithm; the following section describes our experimental setup, evaluation metrics, and results.

Singular Value Decomposition (SVD)

It is a well-known matrix factorization technique that factors an $m \times n$ matrix R into three matrices as the following: $R = U \cdot S \cdot V'$

Where, U and V are two orthogonal matrices of sizes $m \times r$ and $n \times r$ respectively; r is the rank of the matrix R. S is a diagonal matrix of size $r \times r$ having all singular values of matrix R as its diagonal entries. All the entries of matrix S are positive and stored in decreasing order of their magnitude. The matrices obtained by performing SVD are particularly useful for our application because of the following property:

SVD provides the best lower rank approximations of the original matrix R, in terms of Euclidean norm. More specifically, it is possible to reduce the $r \times r$ matrix S to have only k largest diagonal values to obtain a matrix S_k, $k < r$. If the matrices U and V are reduced accordingly, then the reconstructed matrix $R_k = U_k.S_k.V_k'$ is the closest rank-k matrix to R. In other words, R_k minimizes the norm $||R- R_k||$ over all rank-k matrices.

The optimal choice of the value k is critical to high-quality prediction generation. We are interested in a value of k that is large enough to capture all the important structures in the matrix yet small enough to avoid overfitting errors. Unfortunately, finding an exact value of such k is still an open problem. We experimentally find a good value of k by trying several different values.

Computing Recommendation Scores from a Customer-Product Ratings Matrix.

SVD cannot be used to factor a sparse matrix. Accordingly, without adding additional information, we must fill our customer-product ratings matrix. We tried two different approaches to fill-in the missing values: using the average ratings for a customer and using the average ratings for a product. We found the product average produce a the better result. We also considered two normalization techniques: conversion of ratings to z-scores and subtraction of customer average from each rating. We found the latter approach to provide better results. Filling and normalization are not used in the published Pearson/nearest neighbor algorithms, so we did not use them in our comparison system.

Given an implementation of SVD and a filled, normalized matrix, we factor the filled, normalized matrix; Then, the remaining step is the generation of a recommendation based on the factored matrices. Based on the LSI algorithm described in (Deerwester et al. 1990), we:

- reduce the matrix $\hat{S}$ to dimension k
- compute the square-root of the reduced matrix S_k, to obtain $S_k^{1/2}$
- compute two resultant matrices: $U_k S_k^{1/2}$ and $S_k^{1/2} V_k'$

These resultant matrices can now be used to compute the recommendation score for any customer c and product p. Recall that the dimension of $U_k S_k^{1/2}$ is $m \times k$ and the dimension of $S_k^{1/2} V_k'$ is $k \times n$. To compute the recommendation score, we compute the dot product of the c^{th} row of $U_k S_k^{1/2}$ and the p^{th} column of $S_k^{1/2} V_k'$.

Experiments

Experimental Platform

We used data from our MovieLens recommender system to evaluate the effectiveness of our LSI-based recommendation algorithm. MovieLens (www.movielens.umn.edu) is a web-based research recommender system that debuted in Fall 1997. Each week hundreds of users visit MovieLens to rate and receive recommendations for movies. The site now has over 8000 users who have expressed opinions on 2500 different movies.[1]

We randomly selected enough users to obtain 100,000 ratings from the database (we only considered users that had rated twenty or more movies). We divided the ratings into an 80,000-rating training set and a 20,000-rating test set. The

[1] In addition to MovieLens' users, the system includes over two million ratings from more than 45,000 EachMovie users. The EachMovie data is based on a static collection made available for research by Digital Equipment Corporation's Systems Research Center.

training data was converted into a user-movie matrix R that had 943 rows (i.e., 943 users) and 1682 columns (i.e., 1682 movies that were rated by at least one of the users). Each entry $r_{i,j}$ represented the rating (from 1 to 5) of the i^{th} user on the j^{th} movie.

We also entered the 80,000 training ratings into DBLens, a collaborative filtering recommendation engine that employs the Pearson nearest neighbor algorithm. DBLens is a flexible recommendation engine that implements collaborative filtering algorithms in a commercial SQL database. We configured DBLens to use the best published Pearson nearest neighbor algorithm and configured it to deliver the highest quality without concern for performance (i.e., it considered every possible neighbor to form optimal neighborhoods).

For each of the 20,000 ratings in the test data set, we requested a recommendation score from DBLens and computed a recommendation score from the matrices $U_k S_k^{1/2}$ and $S_k^{1/2} V_k'$.

Evaluation Metrics

Recommender systems research has used three types of measures for evaluating the success of a recommender system.

Coverage metrics evaluate the number of products for which the system could provide recommendations. Overall coverage is computed as the percentage of customer-product pairs for which a recommendation can be made. In controlled dataset experiments, a commonly used coverage metric is the percentage of test-set ratings for which a recommendation could be made.

Statistical accuracy metrics evaluate the accuracy of a system by comparing the numerical recommendation scores against the actual customer ratings for the customer-product pairs in the test dataset. Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and Correlation between ratings and predictions are widely used metrics. Our experience has shown that these metrics typically track each other closely. We therefore report MAE because it is most commonly used and easiest to interpret directly.

Decision support accuracy metrics evaluate how effective a prediction engine is at helping a user select high-quality products from the set of all products. These metrics assume the recommendation process as a binary operation—either products are recommended (good) or not (bad). With this observation, whether a product has a recommendation score of 1.5 or 2.5 on a five-point scale is irrelevant if the customer only chooses to consider recommendations of 4 or higher. The most commonly used decision support accuracy metrics are reversal rate, weighted errors and ROC sensitivity. Reversal rate is the frequency with which the system makes recommendations that are extremely wrong, e.g., off by 3 points or more on a five-point scale. Weighted error measures give extra weight to large errors that occur when the customer has a strong opinion about a product. For example, these may double the weight of errors when the customer actually considers the product to be a top (5 out of 5) or bottom (1 out of 5) product. ROC sensitivity is a measure of the diagnostic power of a filtering system. Operationally, it is the area under the receiver operating characteristic (ROC) curve—a curve that plots the sensitivity and specificity of the test (Le et al. 1995). Sensitivity is the probability of a randomly selected good product being accepted by the filter. Specificity is the probability of a randomly selected bad product being rejected by the filter. The ROC curve plots sensitivity (from 0 to 1) and *1 – specificity* (from 0 to 1), obtaining a set of points by varying the recommendation score threshold above which the product is accepted. The area under the curve increases as the filter is able to retain more good products while accepting fewer bad ones. ROC sensitivity ranges from 0 to 1 where 1 is perfect. A random filter is expected to accept half of the good products and half of the bad ones and hence provides an ROC sensitivity of 0.5.

We used ROC sensitivity as our decision support accuracy measure. To use ROC sensitivity as a metric, we must determine which movies are "good" and which are "bad." A movie rating of 4 or 5 is deemed to be a good movie for that user (signal); a rating of 1, 2, or 3 is deemed to be a bad movie for that user (noise). The ROC sensitivity measure therefore is an indication of how effectively the system can steer people towards movies that they will rate highly.

Experimental Steps

Each entry in our data matrix R represents a rating on a 1-5 scale, except that in cases where the user i didn't rate movie j the entry $r_{i,j}$ is null. We then performed the following experimental steps.

- Compute the average ratings for each user; we perform this step by computing the row-average $\underline{r_i}$ of non-null matrix entries.

- Compute the average ratings for each movie; we perform this step by computing the column-average $\underline{c_j}$ of non-null matrix entries.
- Fill the null entries in the matrix by replacing each null entry with the column average for the corresponding column.
- Normalize all entries in the matrix by replacing each entry $r_{i,j}$ with $(r_{i,j} - \underline{r_i})$. Note that the saved user average considers only actual ratings, not filled ratings, so the row means may not be zero.
- Load the 80,000 training set ratings into DBLens and request recommendation scores on each of the 20,000 test set ratings.
- Compare the original customer ratings with the obtained recommendation scores from the SVD system. Compute MAE and ROC-sensitivity of the results. Compute MAE and ROC values for the DBLens recommendation scores and compare the two sets of results.
- Any improvement in results was checked for statistical significance. We used the SPSS statistical package to perform a *Wilcoxon test* to assess the significance on MAE results using a 95% confidence level. Statistical significance assessment for ROC sensitivity was also done at the 95% confidence level. Since we computed the area under the ROC curve as our metric, we compared two such results by finding the values of control point (Le et al. 1995) on the ROC curve and then using the data to generate the significance test.

	Data set 1		Data set 2	
Dimension, *k*	**ROC sensitivity**	**MAE**	**ROC sensitivity**	**MAE**
2	0.76754	0.74875	0.76418	0.76028
5	0.77456	0.73957	0.77725 *	0.75178 *
10	0.77573 *	0.73821 *	0.77816 *	0.75035 *
15	0.77647 *	0.73801 *	0.77835 *	0.75001 *
18	0.77669 *	0.73747 *	0.77892 *	0.74923 *
19	0.77657 *	0.73793 *	0.77898 *	0.74902 *
20	0.77660 *	0.73788 *	0.77895 *	0.74919 *
50	0.77410	0.74069	0.77458 *	0.75422 *
100	0.76562	0.75066	0.77071 *	0.75995 *
DBLens Results	*0.77413*	*0.74041*	*0.75739*	*0.77897*

Table 1: Experiment results, expressed as ROC sensitivity and MAE. (* = statistically significant)

- MATLAB was used to compute the SVD of the filled and normalized matrix *R*, producing the three component matrices of equation (1). We call them *U, S* and *V'* accordingly. *S* is the matrix that contains the singular values of matrix *R* sorted in decreasing order.
- Perform the dimensionality reduction step by retaining only *k* largest singular values and replacing the rest of the diagonal entries (i.e., from *k+1* to *r*) with 0.
- We computed the square root of the reduced matrix and computed the matrices $U_k S_k^{1/2}$ and $S_k^{1/2} V'_k$ as mentioned above.
- Multiply the matrices $U_k S_k^{1/2}$ and $S_k^{1/2} V'_k$ producing a 943 x 1682 matrix. Since the inner product of a row from $U_k S_k^{1/2}$ and a column from $S_k^{1/2} V_k$ gives us a recommendation score, this resultant matrix *P* holds the recommendation score for each user-movie pair *i,j* in P_{ij}. De-normalize the matrix entries by adding the user average back into each recommendation score.

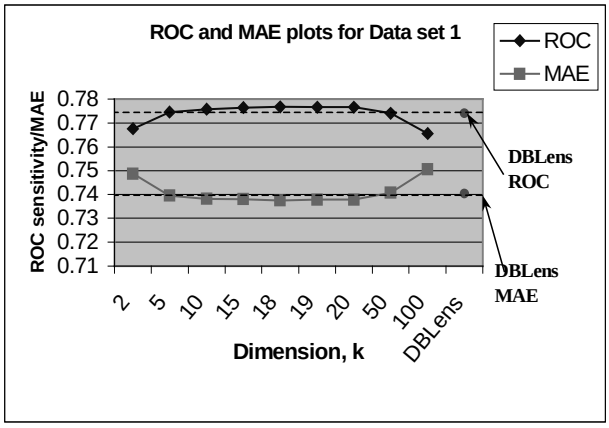
We repeated the entire process for *k* = 2,3,5,10,15,18,20,50 and 100, and for two sets of data.

Results and Discussion

Table 1 shows our experimental results for two different data sets. The data sets were obtained from the same sample of 100,000 data, by randomly selecting two different trial and test data sets. The rows of the table represent a test result for a particular number of dimensions, k, for these two data sets. The last row (italicized entries) shows the results from our DBLens experiments on the same sets of data. The asterisks "*" in the table indicate

which results were statistically significantly different from the DBLens results. We observe that the quality of prediction increases when the reduced SVD feature space contains more dimensions as evident by the increase of ROC sensitivity and decrease of MAE values with increasing k for k = 2, 5, 10, 15 and 18. This increasing trend of quality with k suggests that the higher the number of dimensions, the better is SVD in capturing underlying relationships. However, when k > 18, we observe a decrease in quality as measured by both ROC and MAE.

Figure 4 charts the change in MAE and ROC for increasing values of k.. In case of data set 1, we see that the ROC value is highest for k=18. In case of data set 2, the same occurs at k=19. We interpret the decrease in quality when k is increased beyond its optimal level to reflect overfitting the model to unimportant and noisy data.



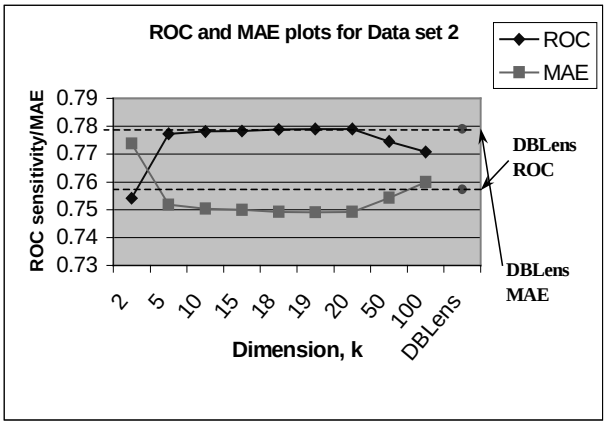


Figure 4. Experimental results by number of dimensions

We then compared the results from our SVD algorithm against DBLens. Table 1 shows that SVD outperformed the Pearson nearest neighbor algorithm whenever the value of k was anywhere close to optimal. We find this encouraging, since it suggests that the value of the SVD implementation is not dependent on finding a perfectly optimal value of k.

Overall the results are encouraging for the use of SVD in collaborative filtering recommender systems. The SVD algorithms fit well with the collaborative filtering data, and they result in good quality recommendations.

Conclusions

Singular Value Decomposition (SVD) is effective in providing high quality recommendations for recommender systems. The SVD approach we studied is a straightforward application of SVD to reduce the dimension of the ratings matrix from a collaborative filtering system. This technique produces higher quality recommendations than the best published collaborative filtering algorithms. Furthermore, once the SVD has been computed, the resulting model provides fast online performance, requiring just a few simple arithmetic operations for each recommendation. Computing the SVD is expensive. If the same quality can be achieved with incremental SVD algorithms, even the model computation could be done online. There are many other ways in which SVD could be applied to recommender systems problems, including using SVD for neighborhood selection, or using SVD to create low-dimensional visualizations of the ratings space. These remain as future work.

This project shows that KDD and recommender systems are closely related, as we hypothesized in the introduction. The two systems originated to serve different business purposes. KDD originated to enable business analysts to search for meaning in large corporate databases. Recommender systems evolved to enable Web sites to respond interactively to customers with recommendations of products to purchase. However, the resulting technologies are similar. It is likely that we can continue to learn from each other.

References

[1] Agarwal, R., Imielinski, T., and Swami, A. 1993. "Mining Association Rules between sets of Items in Large Databases. In *Proceedings of ACM SIGMOD conference on Management of Data*, pp. 207-216.

[2] Aggarwal, C. C., Sun, Z., and Yu, P. S. 1998. "Online Generation of Profile Association Rules." In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pp. 129-133.

[3] Berry, M. W., Dumais, S. T., and O'Brian, G. W. 1995. "Using Linear Algebra for Intelligent Information Retrieval". SIAM Review, 37(4), pp. 573-595.

[4] Billsus, D., and Pazzani, M. J. 1998. "Learning Collaborative Information Filters". In Proceedings of Recommender Systems Workshop. Tech. Report WS-98-08, AAAI Press,

[5] Bhattacharyya, S. 1998. "Direct Marketing Response Models using Genetic Algorithms." In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pp. 144-148.

[6] Brachman, R., J., Khabaza, T., Kloesgen, W., Piatetsky-Shapiro, G., and Simoudis, E. 1996. "Mining Business Databases." *Communications of the ACM*, 39(11), pp. 42-48, November.

[7] Cheeseman, P. 1990. "On Finding the Most Probably Model." In *Computational Models of Scientific Discovery and Theory Formation*, ed. Shrager, J. and Langley, P. San Francisco: Morgan Kaufmann.

[8] 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), pp. 391-407.

[9] Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., Eds. 1996. "Advances in Knowledge Discovery and Data Mining". *AAAI press/MIT press*.

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

[11] Heckerman, D. 1996. "Bayesian Networks for Knowledge Discovery." In *Advances in Knowledge Discovery and Data Mining*. Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., Eds. *AAAI press/MIT press*.

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

[13] Le, C. T., and Lindgren, B. R. 1995. "Construction and Comparison of Two Receiver Operating Characteristics Curves Derived from the Same Samples". *Biom. J.* 37(7), pp. 869-877.

[14] Ling, C. X., and Li C. 1998. "Data Mining for Direct Marketing: Problems and Solutions." In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pp. 73-79.

[15] Piatetsky-Shapiro, G., and Frawley, W. J., Eds. 1991. "Knowledge Discovery in Databases". *AAAI press/MIT press*.

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

[17] Sarwar, B., M., Konstan, J. A., Borchers, A., Herlocker, J., Miller, B., and Riedl, J. 1998. "Using Filtering Agents to Improve Prediction Quality in the GroupLens Research Collaborative Filtering System." In *Proceedings of CSCW '98*, Seattle, WA.

[18] Shardanand, U., and Maes, P. 1995. "Social Information Filtering: Algorithms for Automating 'Word of Mouth'." In *Proceedings of CHI '95*. Denver, CO.

[19] Uthurusamy, R. 1996. "From Data Mining to Knowledge Discovery: Current Challenges and Future Directions." In *Advances in Knowledge Discovery and Data Mining*. Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., Eds. *AAAI press/MIT press*.

[20] Zytkow, J., and Baker, J. 1991. "Interactive Mining of Regularities in Databases." In *Knowledge Discovery in Databases.* Piatetsky-Shapiro, G., and Frawley, W. J. Eds. AAAI Press/MIT Press.

[21] Zytkow, J. M. 1997. "Knowledge = Concepts: A Harmful Equation." In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*.