An empirical study of the recursive input generation algorithm for memory-based collaborative filtering recommender systems

Serhiy Morozov

Mathematics, Computer Science, and Software Engineering,
University of Detroit Mercy,
Detroit, MI 48221-3038, USA
E-mail: morozose@udmercy.edu

Hossein Saiedian*

Electrical Engineering and Computer Science,
The University of Kansas,
Lawrence, KS 66045, USA
Fax: 913-897-8682
E-mail: saiedian@eecs.ku.edu
*Corresponding author

Abstract: Recommender system research has gained popularity recently because many businesses are willing to pay for a way to predict future user opinions. Such knowledge could simplify decision-making, improve customer satisfaction, and increase sales. We focus on the recommendation accuracy of memory-based collaborative filtering recommender systems and propose a novel input generation algorithm that helps identify a small group of relevant ratings. Any combination algorithm can be used to generate a recommendation from such ratings. We attempt to improve the quality of these ratings through recursive sorting. Finally, we demonstrate the effectiveness of our approach on the Netflix dataset, a popular, large, and extremely sparse collection of movie ratings.

Keywords: recommender systems; recommendation accuracy; future behaviour prediction; input generation algorithm.

Reference to this paper should be made as follows: Morozov, S. and Saiedian, H. (2013) 'An empirical study of the recursive input generation algorithm for memory-based collaborative filtering recommender systems', *Int. J. Information and Decision Sciences*, Vol. 5, No. 1, pp.36–49.

Biographical notes: Serhiy Morozov is currently an Assistant Professor at the Department of Mathematics, Computer Science, and Software Engineering at the University of Detroit Mercy. He teaches undergraduate and graduate courses in software engineering and serves on the software engineering accreditation committee. He is currently involved in the recommender systems research, but his other interest include data mining, knowledge discovery, machine learning, and web security/privacy. Prior to his academic career, he worked as a Web Developer for over five years. He received his BA from the Westminster College in 2005, MS from the University of Kansas in 2007, and PhD from the University of Kansas in 2011.

Hossein Saiedian is currently a professor of Software Engineering in the Department of Electrical Engineering and Computer Science at the University of Kansas. His primary area of research is software engineering and in particular software architecture and models for quality software development, both technical and managerial ones. He has also been investigating security issues and in particular secure software engineering. He has over 150 publications in a variety of topics in software engineering, computer science, and information security. His research in past have been supported by NSF as well as regional organisations. He is a senior member of IEEE.

1 Introduction

The increasing popularity and growth of the World Wide Web provide the two things that make recommender systems necessary: a large catalogue of content and a community of users willing to share their opinions. Recommender systems are particularly useful for e-commerce applications, where customers benefit from personal shopping assistance and stores increase sales. Therefore, now is the time to research recommender systems. The information they require is widely available and there is a great need for their services.

Collaborative filtering recommenders suggest items that similar users enjoyed because people who agreed in the past are likely to agree in the future. This approach can recommend new and interesting items that content-based systems fail to recognise due to their over-specialisation tendency. In fact, item content is completely irrelevant, because collaborative filtering recommendations are based exclusively on user opinions. We focus on this approach because it applies to a wide variety of domains.

One type of collaborative filtering system, called the memory-based approach, works directly with the dataset, inspecting it before each recommendation. We choose to research memory-based recommenders because they offer superior accuracy to probabilistic and associative rule models (Breese et al., 1998; Sarwar et al., 2001; Canny, 2002). These models are too general to make personalised recommendations, whereas memory-based methods are simple, accurate, and use new data immediately. However, because calculations are done on-demand, memory-based approaches are notoriously slow when applied to large datasets (Herlocker et al., 1999; Deshpande and Karypis, 2004). Therefore, we focus our efforts on improving recommendation accuracy, not performance.

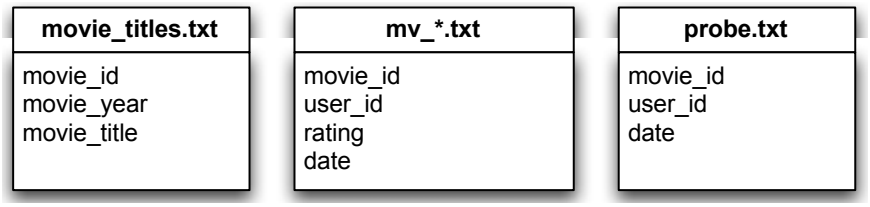
2 Source of the experimental data

We evaluate the fitness of our recommender prototype on a recent and widely published dataset provided by Netflix. In fact, task 1 of the leading data mining and knowledge discovery competition in the world, is based on this dataset (Bennett et al., 2007). One of the most obvious reasons for such popularity is the dataset size. It contains over 100,000,000 actual movie ratings on a discreet scale from one to five. It represents opinions of over 480,000 users and almost 18,000 movies (Bennett et al., 2007). This

dataset is more than 30 times larger than any other available dataset. However, it represents only 1.1% of all possible ratings (Bennett et al., 2007), so approaches that rely on a higher data density may not apply. Accurate memory-based recommendation are particularly difficult on sparse datasets, so we choose to use this challenging dataset to evaluate our system.

The structure of our dataset is fairly standard. Figure 1 shows how the information about every movie is available in a single file. The ratings are grouped by movie and stored in separate files. The dataset contains a list of 2,800,000 withheld ratings (qualifying/test set). Netflix evaluates the estimates of these opinions. The dataset also contains 1,400,000 known ratings (probe/quiz set) intended for local evaluation. We use the quiz set for our empirical study.

Figure 1 Netflix dataset structure



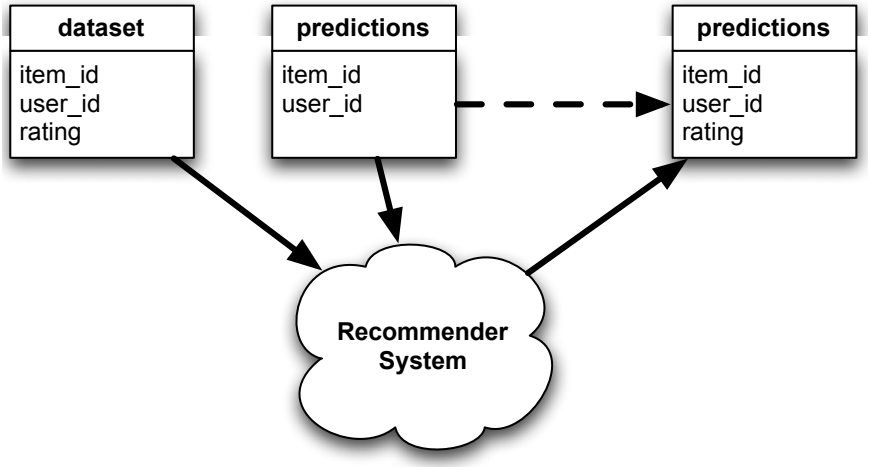
3 Research hypothesis

Recent studies show that it is increasingly difficult to make substantial progress in prediction accuracy. For instance, a 1% improvement on the Netflix prize challenge took nearly two years (Bennett and Stan, 2007). We believe it is because most effort has been directed toward optimising algorithms instead of improving their input data. One of the most common ways to do so is to supplement the dataset (Herlocker et al., 2004). However, users are usually hesitant to provide additional ratings, third party data might not be available, and implicit rating analysis is unreliable (Middleton, 2003; Bell and Koren, 2007a). Accordingly, we do not add more data, but remove data we know is irrelevant. The main contribution of our work is an algorithm that identifies a set of ratings most relevant to a particular recommendation. Even though it is difficult to make accurate assumptions from little evidence, we believe that better input selection can lead to more accurate recommendations, regardless of the combination algorithm.

3.1 Conceptual model of a recommender system

The goal of a recommender system is to learn user preferences from the past and apply this knowledge to predict the future. Figure 2 demonstrates the input and output of a typical collaborative filtering recommender system. It trains on a set of known ratings and produces predictions for a set of unknown ones. At this level, the exact calculations inside a recommender system are irrelevant, as long as it produces suggestions in the desired format. Our system represents movies with item ids and ignores the rest of the movie attributes.

Figure 2 Typical recommender system input/output



More formally, the recommender system may be described as a set of m users $U = \{u_1, u_2,\ldots,u_m\}$ and a set of n items $I = \{i_1, i_2,\ldots,i_n\}$. Each user u has an associated set of items $I_u \subseteq I$, which he/she has rated. Each rating r is assumed to be on a discrete numerical scale, even though continuous rating scales are also common (Nathanson et al., 2007). The user and item for which the prediction is to be made are called *active user* and *active item* (Sarwar et al., 2001; Candillier et al., 2007). A recommender system guesses the opinion of user u on item i, $P_{u,i}$. Usually, production systems predict opinions that have not been previously recorded, $P_{u,i} \mid i \notin I_u$. Development systems guess a set of known ratings, so that the error of each prediction may be computed, $P_{u,i} - r_{u,i}$.

3.2 Evaluation metrics and accuracy goals

A popular way to evaluate the quality of a recommender system is to quantitatively measure its accuracy. One such measure is the root mean squared error (RMSE). It is the square root of the average of squared deviations (Sarwar et al., 2001; Candillier et al., 2007):

$$RMSE = \sqrt{\frac{\sum_{u \in U, i \in I} \left(P_{u,i} - r_{u,i}\right)^2}{|P|}}.$$

The RMSE metric uses the same units as data, i.e., it represents the size of a typical error. However, it does not have an absolute value that is considered satisfactory. Instead, recommender systems are ranked according to their typical error size within a dataset.

The RMSE metric is most appropriate for our research. The creators of the Netflix dataset deem large errors to be particularly undesirable, i.e., the cost of an error is greater than its size. Because the RMSE measure squares each deviation, larger errors influence it more than smaller ones. Furthermore, because it is impossible to compare accuracy on different rating scales, recommendations on the same dataset usually use the same metric.

Finally, our dataset has a published list of 130,000 RMSE scores on the quiz dataset (Bennett and Stan, 2007), so we use RMSE to evaluate our prototype.

To set our accuracy goals and establish a point of reference for our experiments, we examine some of the well-known results from the Netflix leader board website, http://www.netflixprize.com. It lists the typical prediction errors of many trivial recommendation approaches that suggest the same rating for every item. For instance, recommending a four star rating for each movie is the most accurate option because each recommendation is close to the overall average rating of 3.6 stars. Likewise, recommending 3.6 stars for each movie produces a smaller error of 1.1287. This value may be reduced further by recommending the user or movie average. This results in a typical error of 1.0651 for average user and 1.0533 for average movie approach. In general, any recommender that is consistently off by one or more units is considered inferior. These figures establish the lowest accuracy threshold below which the recommendations are no longer useful.

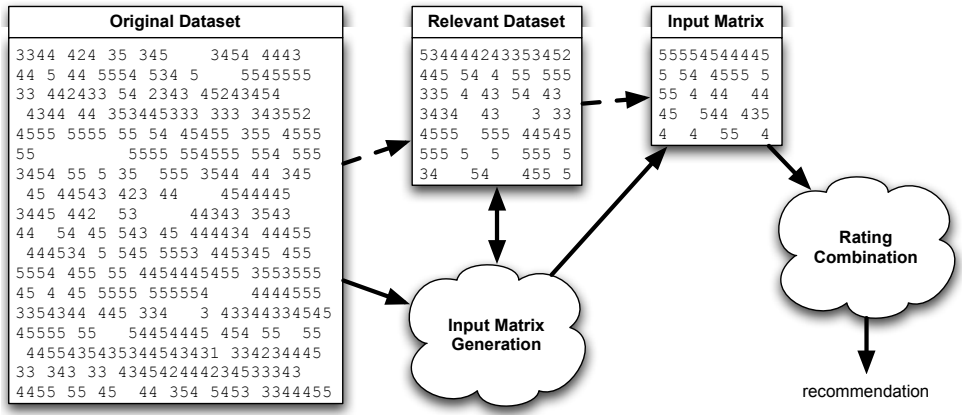
However, the highest accuracy threshold is still largely unknown. Thousands of contestants spent years trying to reduce a typical error of their recommender on the Netflix dataset. On July 26th, 2009 a team named 'BellKor's Pragmatic Chaos' reached a previously impossible RMSE value of 0.8567. The authors of the winning algorithm published three papers detailing their approach (Koren, 2009; Töscher et al., 2009; Piotte and Chabbert, 2009). Their work has motivated our research by showing that a significant improvement in recommendation accuracy is possible on large and sparse datasets. We believe that recommendation accuracy can be improved even further. In fact, we try to get closer to the lowest possible error, dictated by unpredictable human nature.

3.3 Relevance of input generation algorithms

Collaborative filtering systems produce recommendations from a subset of all users, called neighbours, which are similar to the active user. Some versions of this approach view a dataset as a collection of user vectors with a specific number of dimensions, corresponding to items they rated (Sarwar et al., 2000a; Herlocker et al., 1999; Wang et al., 2006). However, item vectors are also possible (Linden et al., 2003; Deshpande and Karypis, 2004; Miller, 2003). In fact, item-based approach produces better results (Sarwar et al., 2001, 2000b; Linden et al., 2003; Huang et al., 2004; Shardanand and Maes, 1995) because item vectors usually contain more ratings and therefore better capture the opinions of a neighbourhood. We model recommender input as a matrix where items are rows and users are columns. Each cell contains a rating that is associated with a single user and an item. A transposed matrix would represent users as rows and items as columns, thus allowing the same input to be used by different algorithms.

Figure 3 shows our prototype data flow. The input generation component locates all relevant ratings and chooses the best ones for the input matrix. The matrix is fed into a combination algorithm that, as the name implies, combines ratings to produce a recommendation. A number of successful combination algorithms exist. We implement a few popular ones, however the main focus of our research is on the input generation component.

Figure 3 Proposed system data flow



To generate an input matrix, we first identify all relevant ratings on items rated by the active user and by users who have rated the active item. For example, if making an input matrix for Alice on Titanic, we consider all movies Alice rated and all users who rated Titanic. This data access task is time-consuming because a large portion of the dataset may be relevant. However, the relevant dataset may contain opinions that are not as valuable as others. Therefore, the input matrix generation algorithm chooses only the most valuable ratings and places them into the input matrix. In general, a recommender system considers ratings from similar vectors to be more valuable. To quantify the similarity between any two vectors, it must first identify common dimensions, i.e., vote overlap. Then we can measure the similarity between the two sets of ratings.

One popular measure that works well in sparse datasets is cosine similarity. It compares two users by taking a cosine of the angle between their rating vectors (Sarwar et al., 2000a; Zhang et al., 2002; Miller et al., 2004). Cosine similarity is accurate because it considers the difference between ratings to be more important than their quantity (Candillier et al., 2007; Sarwar et al., 2001). The formula below shows a cosine similarity as the sum of products of commonly rated items divided by the product of vector lengths:

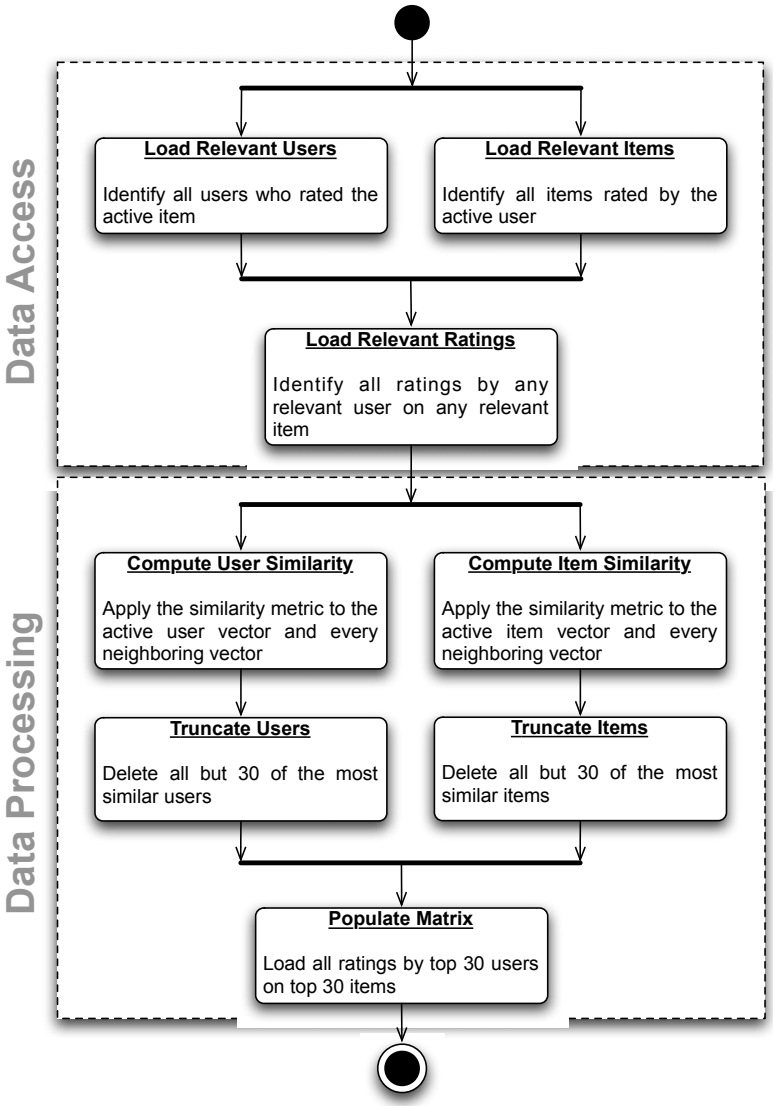
$$sim(a,u) = \frac{\sum_{i \in I_a \cap I_u} r_{a,i} r_{u,i}}{\sqrt{\sum_{i \in I_a \cap I_u} r_{a,i}^2 \sum_{i \in I_a \cap I_u} r_{u,i}^2}}.$$

The cosine similarity measure has been widely used in information retrieval research. However, it does not consider users' rating scales. For instance, a three star rating could mean 'average' to one person and 'good' to another. The cosine similarity uses actual ratings, so unless two users have the same rating scales, their similarity is lost. Alternatively, linear regression approximations can recognise similarity in such situations (Herlocker et al., 1999; Bell and Koren, 2007b). For example, Pearson's correlation measures linear dependence between two user vectors:

$$sim(a,u) = \frac{\sum_{i \in I_a \cap I_u} \left(r_{a,i} - \overline{r}_a\right)\left(r_{u,i} - \overline{r}_u\right)}{\sqrt{\sum_{i \in I_a \cap I_u} \left(r_{a,i} - \overline{r}_a\right)^2 \sum_{i \in I_a \cap I_u} \left(r_{u,i} - \overline{r}_u\right)^2}}.$$

This measure is similar to cosine similarity, except individual ratings are normalised by the vector average. This adjustment improves accuracy of even the most basic recommenders. In fact, it is most effective in sparse datasets, where linear regression is more easily established (Sarwar et al., 2001; Bennett et al., 2007). We examine the effects of both similarity measures in our empirical study.

Figure 4 Matrix generation steps

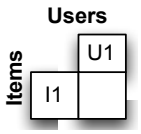


4 Standard input generation process

Because our dataset contains millions of records, combining all relevant ratings is not feasible. Instead, we make recommendations from a subset of the most relevant ratings. Figure 4 shows how we identify the users and items that comprise an input matrix. This process consists of a series of simple operations, many of which could be executed in parallel to increase performance. However, there are two synchronisation points in this workflow when only one operation is running. Therefore, we break it down in two steps at the synchronisation boundary. This division is not necessary in a production system, but it helps us separate a strictly data access task from a strictly data processing task.

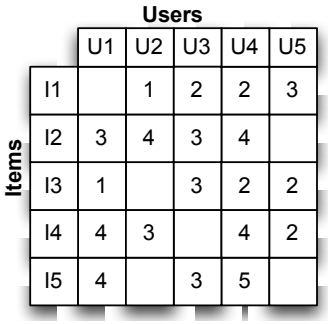
The standard process constructs an input matrix by first creating a larger one and then reducing its size. We start with a single cell matrix for the active user and item. Figure 5 shows the initial state of an input matrix. In this case, we are going to predict a rating by user U1 on item I1. This step is necessary because we want to make sure that the active user and item vectors are in the matrix regardless of whether we know the active rating. If the active rating is unknown, the two vectors would be an exception to the following step.

Figure 5 Input matrix generation – initial state



The second step populates the matrix with all relevant user and item vectors as well as any ratings they have. At this point, columns are all users who rated the active item and rows are every item that the active user rated. Figure 6 shows the populated matrix. In this case, the active user has rated five items, and five users, including the active user, rated the active item. Note that the first row and the first column of the input matrix always have values, with the exception of the rating we are trying to predict. This step concludes the data access task of the input generation algorithm.

Figure 6 Input matrix generation – load users and items



The data processing task sorts columns and rows, while preserving rating association. More similar vectors are positioned closer to the active vector. Figure 7 shows the matrix after the third step of our process. The sorting order does not make a difference and performing it in parallel or in series will produce the same result. Finally, we remove the

least similar rows and columns. Figure 8 shows the final truncated matrix, where we keep the top three vectors. Since all neighbours are organised in order of decreasing similarity from the active vector, truncating the matrix deletes only the least relevant data. As a result, the finished matrix contains more relevant ratings.

Figure 7 Input matrix generation – sort vectors, (a) sorting by items/rows (b) sorting by users/columns

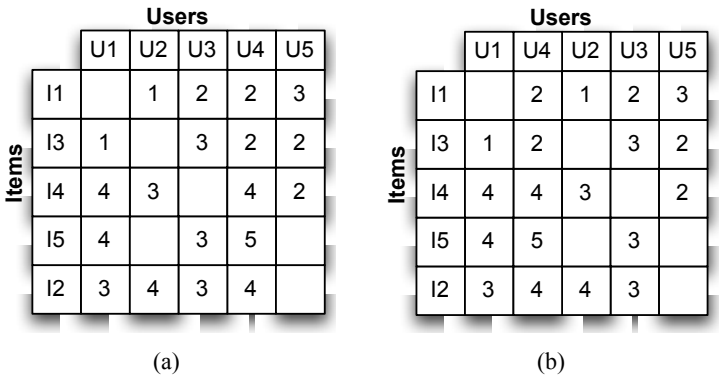
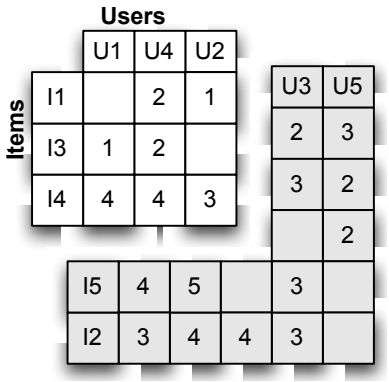


Figure 8 Input matrix generation – truncate matrix



Our standard input generation approach consists of a single pass through the data access and data processing tasks. It is similar to other methods of selecting input with one notable exception, bidirectional truncation. Other approaches compute vector similarity and truncate the matrix across only one dimension, i.e., either users or items (Mobasher et al., 2000; Good et al., 1999; Bell et al., 2007). We truncate both dimensions because it produces a more balanced input that may be used by any combination algorithm.

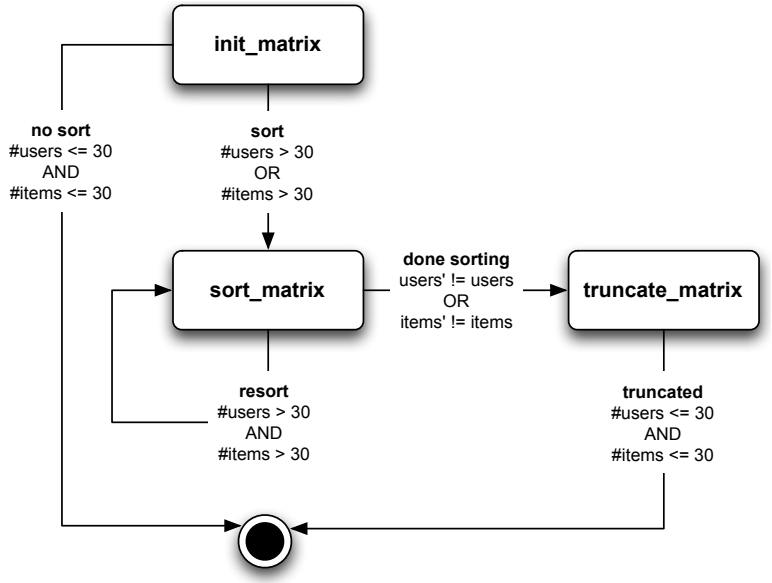
5 Recursive input generation process

Our recursive approach adds an additional pass through the data processing task, which ensures that each neighbour is among the best available vectors according to the best dimensions. It resorts the matrix according to the top items and users established during

the first pass. Instead of truncating all but the most similar vectors, we re-evaluate the vector similarities, but only across the most relevant dimensions. The selected vectors do not necessarily agree in every shared dimension, but they are the most similar vectors in the entire dataset according to the dimensions in the input matrix.

To guarantee that the input is generated properly, we establish the correct order of matrix states in Figure 9. It relates the three main states of a matrix as we refine the ratings within it. The initial matrix contains all relevant ratings, which are only sorted if the matrix is large enough. If the matrix is small, sorting accomplishes nothing because no data is removed. If the matrix is large, sorting establishes vector relevance, so the least valuable data may be discarded. If one of the matrix dimensions is small enough, resorting the matrix does not affect the result because rearranging vector dimensions does not change vector similarities. In either case, the matrix is truncated to a uniform size as the last step of the input generation process.

Figure 9 Matrix state changes during the input generation process



Note that we may want to sort the matrix multiple times. This model of the input generation component supports such behaviour. However, when resorting the matrix, vector similarity is evaluated only across dimension that were not truncated in the previous step. In fact, each subsequent sorting iteration depends on the best dimensions established during the previous iteration, which is the essence of the recursive input generation algorithm.

6 Empirical study

We researched the available collaborative filtering approaches and implement three popular combination algorithms: robust singular value decomposition (RSVD) (Pazzani, 1999; Salakhutdinov et al., 2007; Bennett et al., 2007), k-nearest neighbours (KNN)

(Breese et al., 1998; Sarwar et al., 2001; Canny, 2002), and neural network (NN) (Salakhutdinov et al., 2007; Lam, 2003; Bell and Koren, 2007a). We then inspected the accuracy of different configurations on standard input matrices sorted according to cosine similarity. Unfortunately, even the best combination algorithm was typically within 0.9574 of an actual rating. Therefore, algorithm tuning has little to do with recommendation accuracy. The list below summarises the final configurations of our combination algorithms:

- *RSVD:* [features = 1, cycles = 25, learning rate = 0.02, learning rate reduction = 0.25]
- *KNN-item:* [K = 0.875, N = 30]
- *KNN-user:* [K = 1, N = 30]
- *NN-item:* [hidden nodes = 0, training cycles = 50, learning rate = 0.1]
- *NN-user:* [hidden nodes = 10, training cycles = 40, learning rate = 0.1].

Even though some combination algorithms were better than others, we believe that the substance of input data affects recommendation accuracy the most. Figure 10 shows how the choice of an input matrix generation algorithm affects recommendation accuracy. The combination algorithms are the same for the standard and recursive approach, but the way we choose data in the input matrix changes. The standard approach sorts and truncates the matrix according to the cosine similarity or Pearson's correlation. The recursive approach performs the same process recursively. Because the vectors may be chosen based on cosine similarity or Pearson's correlation, we consider five different configurations of this approach over two, three, and four passes.

Figure 10 Accurate ways of selecting ratings

Method	RSVD	KNN-Item	KNN-User	NN-Item	NN-User
Cosine Standard	1.291	1.301	1.305	1.344	1.317
Pearson's Standard	0.867	0.786	0.826	1.144	0.868
Cosine Recursive (2)	0.923	0.980	0.945	1.164	0.998
Cosine Recursive (3)	0.985	0.975	0.989	1.177	1.008
Pearson's Recursive (2)	0.665	0.423	0.465	1.010	0.611
Pearson's Recursive (3)	0.762	0.465	0.528	0.949	0.623
Pearson's Recursive (4)	0.722	0.456	0.519	0.938	0.600

Each algorithm produces 30 × 30 input matrices because this neighbourhood size has been empirically shown to be most accurate (Sarwar et al., 2001; Herlocker et al., 1999; Miller et al., 2004; Breese et al., 1998). Such matrices introduce enough evidence to support a particular recommendation, without unnecessary noise. Finally, square matrix design is convenient because we can transpose it for user and item-based approaches without regenerating it.

We produced 1,000 recommendations randomly chosen from the quiz dataset to compare the two similarity metrics for the two types of input generation process. The results show that using Pearson's correlation is considerably better than cosine similarity. In fact, this similarity measure significantly reduced the RMSE score for all combination algorithms. Also, recursive approach was more accurate for cosine as well as Pearson's

correlation values. Finally, anything over two iterations of the Pearson's recursive algorithm did not improve accuracy.

Our final recommender system chooses its ratings according to two passes of the recursive method. The first pass computes Pearson's correlations across user and item vectors. It also records the top 30 most similar users and items. The second pass recalculates Pearson's correlations for user and item vectors, but only according to the top items/users established in the first pass. As a result, the input matrix contains only the most relevant ratings.

To ensure the trustworthiness of our prototype, we performed multiple experiments with 1,000, 10,000, and 50,000 recommendations randomly chosen from the quiz dataset. Figure 11 shows the average RMSE scores of each experiment. The results demonstrate that more recommendations increase the typical error slightly. However, these results also show the effectiveness of our approach on increasingly larger samples of the Netflix quiz dataset. Our algorithm is too slow to predict the entire quiz dataset, but its accuracy is remarkable.

Figure 11 Final prototype evaluation

Rating Count	RSVD	KNN-Item	KNN-User	NN-Item	NN-User
1k	0.6342	0.4054	0.4239	0.7079	0.6150
10k	0.6817	0.4470	0.4537	0.7782	0.6352
50k	0.6719	0.4105	0.4281	0.7466	0.6096

7 Conclusions

Recent research shows that it is possible to make accurate recommendations from content analysis, user behaviour interpretation, and collaborative filtering techniques. We focus our work on the latter approach. However, the main obstacle in making such recommendations is the size and sparsity of modern datasets. It takes a long time to thoroughly analyse large amounts of data, so recommendation accuracy is often sacrificed to improve performance. Additionally, sparse datasets contain little usable information and establish unreliable evidence for making recommendations. Therefore, the challenge is to find a way to make sense of little data, regardless of its original size.

We believe that a small number of relevant ratings is sufficient to make an accurate recommendation. Such ratings may be chosen with a recursive approach, instead of a more traditional method. The main purpose of this input generation approach is to produce small, yet relevant input. It selects relevant vectors according to various similarity versions. One of them relies on relative ratings or opinions normalised to that vector's mean rating. Our empirical results show that such ratings produce better results because users' rating scales do not affect their similarities. We also experiment with recursively resorting the input matrix, since vector similarities of users and items are mutually dependent. The experiments show that just two rounds of similarity computations produce the best results. The recursive input generation algorithm is slow because a large number of ratings must be retrieved, compared, and sorted multiple times. However, the resulting recommendation accuracy justifies the performance drawback.

References

Bell, R.M. and Koren, Y. (2007a) 'Lessons from the Netflix prize challenge', *ACM SIGKDD Explorations Newsletter*, Vol. 9, No. 2, pp.75–79.

Bell, R.M. and Koren, Y. (2007b) 'Scalable collaborative filtering with jointly derived neighborhood interpolation weights', *Proceedings of the Seventh IEEE International Conference on Data Mining*, IEEE Computer Society, pp.43–52.

Bell, R.M., Koren, Y. and Volinsky, C. (2007) 'The Bellkor solution to the Netflix prize', Technical report, AT&T Labs – Research.

Bennett, J. and Stan, L. (2007) 'The Netflix prize', *KDD Cup and Workshop 2007*.

Bennett, J., Elkan, C., Liu, B., Smyth, P. and Tikk, D. (2007) 'KDD cup and workshop 2007', *ACM SIGKDD Explorations Newsletter*, Vol. 9, No. 2, pp.51–52.

Breese, J.S., Heckerman, D. and Kadie, C. (1998) 'Empirical analysis of predictive algorithms for collaborative filtering', *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pp.43–52.

Candillier, L., Meyer, F. and Boullé, M. (2007) 'Comparing state-of-the-art collaborative filtering systems', *Proceedings of the 5th International Conference on Machine Learning and Data Mining in Pattern Recognition*, Vol. 4571 of LNCS, pp.548–562, Springer, available at http://dx.doi.org/10.1007/978-3-540-73499-4_41.

Canny, J. (2002) 'Collaborative filtering with privacy via factor analysis', *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, New York, NY, USA, pp.238–245.

Deshpande, M. and Karypis, G. (2004) 'Item-based top-n recommendation algorithms', *ACM Transactions on Information Systems*, Vol. 22, No. 1, pp.143–177.

Good, N., Schafer, J.B., Konstan, J.A., Borchers, A., Sarwar, B., Herlocker, J. and Riedl, J. (1999) 'Combining collaborative filtering with personal agents for better recommendations', *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, AAAI Press, pp.439–446.

Herlocker, J.L., Konstan, J.A., Borchers, A. and Riedl, J. (1999) 'An algorithmic framework for performing collaborative filtering', *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, New York, NY, USA, pp.230–237.

Herlocker, J.L., Konstan, J.A., Terveen, L.G. and Riedl, J.T. (2004) 'Evaluating collaborative filtering recommender systems', *ACM Transactions on Information Systems*, Vol. 22, No. 1, pp.5–53.

Huang, Z., Chen, H. and Zeng, D. (2004) 'Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering', *ACM Transactions on Information Systems*, Vol. 22, No. 1, pp.116–142.

Koren, Y. (2009) *The BellKor Solution to the Netflix Grand Prize*, August, available at http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf.

Lam, C.P. (2003) 'Collaborative filtering using associative neural memory', in Mobasher, B. and Anand, S.S. (Eds.): *Itwp*, Vol. 3169 of Lecture Notes in Computer Science, pp.153–168, Springer, available at http://dx.doi.org/10.1007/11577935_8.

Linden, G., Smith, B. and York, J. (2003) 'Amazon.com recommendations: item-to-item collaborative filtering', *IEEE Internet Computing*, Vol. 7, No. 1, pp.76–80.

Middleton, S.E. (2003) 'Capturing knowledge of user preferences with recommender systems', PhD thesis, Faculty of Engineering and Applied Science, Department of Electronics and Computer Science.

Miller, B.N. (2003) 'Toward a personal recommender system', PhD thesis, University of Minnesota.

Miller, B.N., Konstan, J.A. and Riedl, J. (2004) 'PocketLens: toward a personal recommender system', *ACM Transactions on Information Systems*, Vol. 22, No. 3, pp.437–476.

Mobasher, B., Dai, H., Luo, T., Nakagawa, M., Sun, Y. and Wiltshire, J. (2000) 'Discovery of aggregate usage profiles for web personalization', *Proceedings of the WebKDD Workshop*.

Nathanson, T., Bitton, E. and Goldberg, K. (2007) 'Eigentaste 5.0: constant-time adaptability in a recommender system using item clustering', *Proceedings of the 2007 ACM Conference on Recommender Systems*, ACM, New York, NY, USA, pp.149–152.

Pazzani, M.J. (1999) 'A framework for collaborative, content-based and demographic filtering', *Artificial Intelligence Review*, Vol. 13, Nos. 5–6, pp.393–408.

Piotte, M. and Chabbert, M. (2009) *The Pragmatic Theory Solution to the Netflix Grand Prize*, August, Pragmatic Theory Inc., Canada.

Salakhutdinov, R., Mnih, A. and Hinton, G. (2007) 'Restricted Boltzmann machines for collaborative filtering', *Proceedings of the 24th International Conference on Machine Learning*, ACM, New York, NY, USA, pp.791–798.

Sarwar, B., Karypis, G., Konstan, J. and Reidl, J. (2001) 'Item-based collaborative filtering recommendation algorithms', *Proceedings of the 10th International Conference on World Wide Web*, ACM, New York, NY, USA, pp.285–295.

Sarwar, B., Karypis, G., Konstan, J. and Riedl, J. (2000a) 'Analysis of recommendation algorithms for e-commerce', *Proceedings of the 2nd ACM Conference on Electronic Commerce*, ACM, New York, NY, USA, pp.158–167.

Sarwar, B.M., Karypis, G., Konstan, J.A. and Riedl, J.T. (2000b) 'Application of dimensionality reduction in recommender systems – a case study', *ACM WebKDD Workshop*.

Shardanand, U. and Maes, P. (1995) 'Social information filtering: algorithms for automating 'word of mouth'', *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, Vol. 1, pp.210–217.

Töscher, A., Jahrer, M. and Bell, R.M. (2009) *The BigChaos Solution to the Netflix Grand Prize*, September, available at http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf.

Wang, J., de Vries, A.P. and Reinders, M.J.T. (2006) 'Unifying user-based and item-based collaborative filtering approaches by similarity fusion', *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, New York, NY, USA, pp.501–508.

Zhang, Y., Callan, J. and Minka, T. (2002) 'Novelty and redundancy detection in adaptive filtering', *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, New York, NY, USA, pp.81–88.