International Journal of Hybrid Information Technology
Vol.8, No.3 (2015), pp.133-144
http://dx.doi.org/10.14257/ijhit.2015.8.3.13

Similarity Analysis in Social Networks Based on Collaborative Filtering

[1,3]Yingchun Hou, [2]Hui Xie and [1]Jianfeng Ma

[1]School of Computer Science and Technology, Xidian University, Xi'an 710071, P. R.China
[2]School of Mathematics & Computer Science, Jiangxi Science & Technology Normal University, Nanchang 330038, P. R. China
[3]Department of Computer Technology, Shangqiu Polytechnic, Shangqiu 476000, P. R.China
houyingchun11@163.com

Abstract

Collaborative Filtering is of particular interest because its recommendations are based on the preferences of similar users. This allows us to overcome several key limitations. This paper explains the need for collaborative filtering, its benefits and related challenges. We have investigated several variations and their performance under a variety of circumstances. We also explored the implications of these results when weighing K Nearest Neighbor algorithm for implementation. Based on the relationship of individuals, putting forward a new incremental learning collaborative filtering recommendation system, discovery it is a better way to acquire optimum results.

Keywords: *social networks, collaborative filtering, k nearest neighbor algorithm*

1. Introduction

Nowadays, we are witnessing in the expansion of the information on the Internet. All the information we need about a specific topic is available in the network, but in many cases the problem is the difficulty to find the information useful for us, among big amounts of useless one. Choosing among millions of products is challenging for consumers, and recommending products to customers is difficult for these sites. Recommender systems have emerged in response to this problem. A recommender system recommends products that are likely to fit they need. Recommender systems benefit customers by enabling them to find products they like. Conversely, they help the business by generating more sales. Today, recommender systems are deployed on hundreds of different sites, such as Amazon, T mall and eBay.

The representative techniques of Memory-based collaborative filtering (CF) include Neighbor-based CFs [1] and Item-based/user-based top-N recommendations [2]. Model-based CF algorithms, such as Bayesian models [3], clustering models [4], and dependency networks [5, 6], have been investigated to solve the shortcomings of memory-based CF algorithms. The design and development of models (such as machine learning, data mining algorithms) can allow the system to learn to recognize complex patterns based on the training data, and then make intelligent predictions for the collaborative filtering tasks for test data or real-world data, based on the learned models. Figure 1 gives a two layer mode of collaborative filtering recommendation.

ISSN: 1738-9968 IJHIT

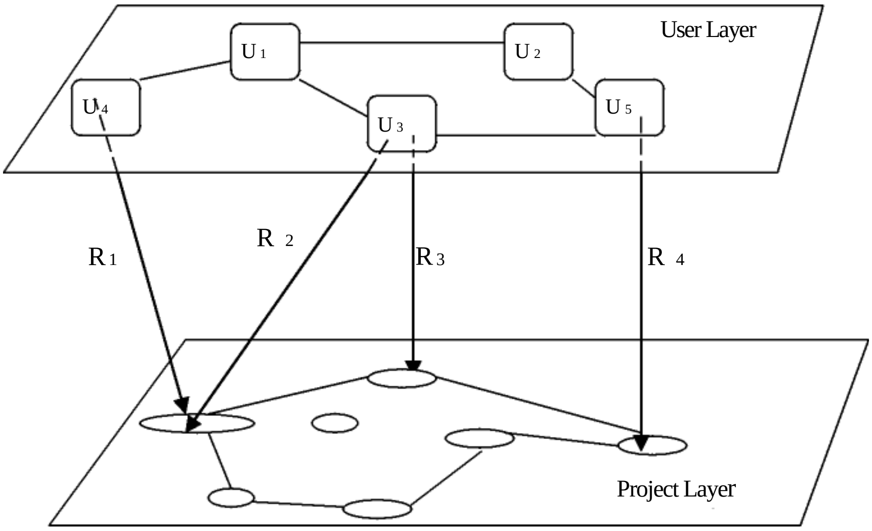

Figure 1. Two Layer Mode of Collaborative Filtering Recommendation

Hybrid CF systems combine CF with other recommendation techniques (typically with content-based system) to make predictions or recommendations. Taking content-based recommender system mentioned in [7] as an example, it makes recommendations by analyzing the content of textual information, gives a higher weight for active user as well as the item that more users rated. By doing so, Hybrid CF improves prediction performance and overcomes CF problems, such as data sparsity and gray sheep. The system has to be independent from the content it is recommending. This means that it is not necessary that the system knows which kind of items it is recommending. The same system should be able to recommend music, films or books if it has past ratings of these kinds of items.

There is a common functionality for the recommender systems. The basic tasks that these systems have to offer to users are [8]:

- First of all the system has to recommend a list of items, that the system considers the most useful for the specific user.
- In other cases when a user asks for an item, the system has to calculate the predicted rating of the item for this specific user.

There are many possibilities to classify the collaborative filtering algorithms. It distinguishes three types [9]:

- Memory-based algorithms, that use all the ratings stored in the database to make the predictions.
- Model-based algorithms, that create a model used to calculate the predictions.
- Hybrid recommenders, those mix collaborative filtering with content based methods.

Here is a table of the main characteristics of each one, which is showed in Table 1.

2. Collaborative Filtering

The entire process of CF-based recommendation system is divided into three sub-tasks namely, representation, neighborhood formation, and recommendation generation as shown in Figure 2.

2.1 Representation

Table1. Types of Collaborative Filtering: Techniques, Advantages &

Disadvantages

In a typical CF-based recommender system, the input data is a collection of historical purchasing transaction of n customers on m products. It is usually represented as an m□n customer-product matrix, R (m,n), which consists of a set of ratings ri,j, such that ri,j is corresponding to the rating for the customer i has on the product j.

(1) Neighborhood formation

The neighborhood formation process is in fact the model-building or learning process for a recommender system algorithm. The most important step in CF-based recommender systems is that of computing the similarity between customers as it is used to form a proximity-based neighborhood between a target customer and a number of like-minded customers.

The main goal of neighborhood formation is to find, for each customer c, an ordered list of

k customers Nc={n1,n2,. . . ,nk}, such that , c**o**N_c, and $sim(c,n_l)\geq sim(c,n_2)\geq\ldots\geq sim(c,n_k)$, where

$sim\ (c,n_i)(1\leq i\leq k)$ indicates similarity between customer *c* and customer n_i.

There are a number of different ways to compute the similarity between items, such as cosine-based similarity, correlation-based similarity [2].

(2) Correlation-based similarity

In this case, similarity between customer *i* and customer *j* is measured by computing the *Pearson-r* correlation $_{corri,j}$. To make the correlation computation accurate the co-rated case $_{Pij}$ must be isolated (*i.e.*, case where the customers rated both *i* and *j*). The *Pearson-r* correlation $corr_{i,j}$ is given by

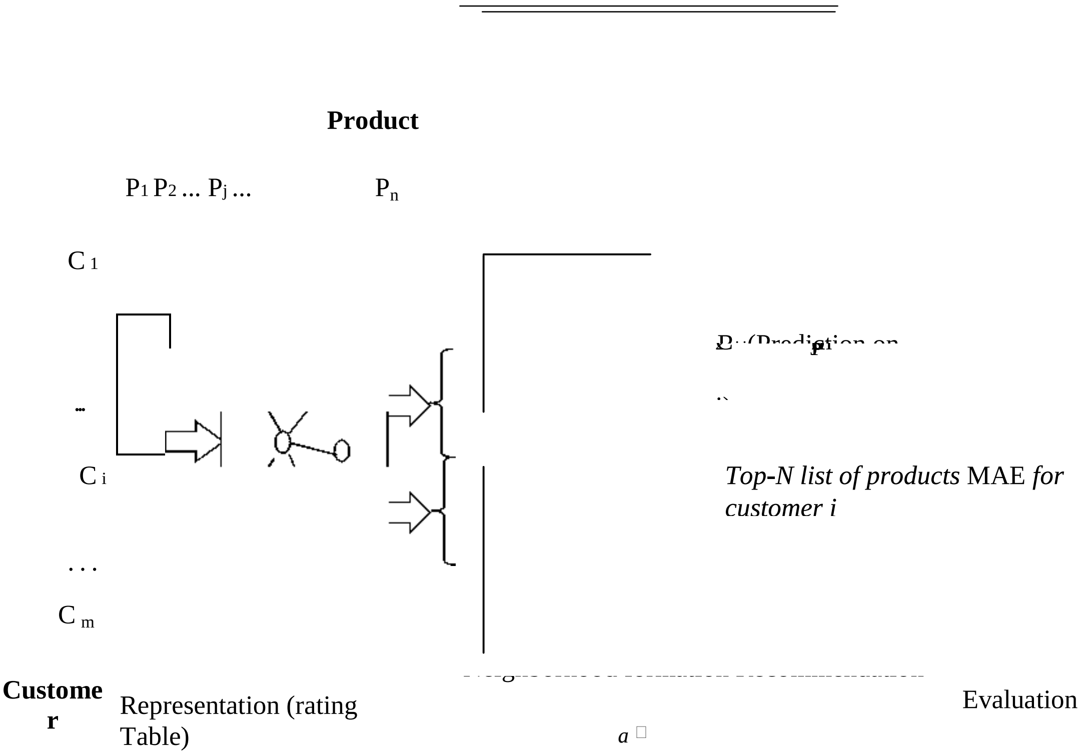
Product
P1 P2 ... Pj ... Pn
C 1
C i
. . .
C m
Customer
Representation (rating Table)
Top-N list of products MAE for customer i
Evaluation
a

Figure 2. The Collaborative Filtering Process [10]

$$sim_{ij} \square \cos_{ij} \quad \square$$

$$b$$

where , and r_{ip} represent the rating of customer *i* on product item *p*. is the rating of customer *i* on the whole product item and is the one of customer *j* .

(3) Cosine-base similarity

In this case, two items are thought of as two vectors in the *m* dimensional customer-space. The similarity between them is measured by computing the cosine of the angle between these two vectors, which is given by

$$\square \quad \frac{b}{}$$

where "." is the dot-product of the two vectors.

2.2 Generation of Recommendation

Once these systems determine the nearest- neighborhood, they produce recommendations that can be of two types:

(1)Prediction

It is a numerical value, $R_{a,j}$, expressing the predicted opinion-score of product p_j for the target customer a. This predicted value is within the same scale as the opinion values provided by a.

(2)Top-N recommendation

It is a list of N products, $TP_r = \{Tp_1, Tp_2, \ldots Tp_N\}$, that the target customer will like the most. The recommended list usually consists of the products not already purchased by the target customer. This output interface of CF algorithms is also known as *Top-N* recommendation.

A widely popular statistical accuracy metric named Mean Absolute Error (MAE) is a measure of the deviation of recommendations from their true customer-specified values.

For each ratings-prediction pair < p_i, q_i >, this metric treats the absolute error between them i.e., $|p_i - q_i|$ equally. The MAE is computed on first summing these absolute errors of the N corresponding ratings-prediction pairs and then computing the average. Formally,

The lower the MAE, the more accurately the recommendation engine predicts customer ratings.

3. K Nearest Neighbor Algorithm in CF

K Nearest Neighbor (KNN) is one of those algorithms that are very simple to understand but works incredibly well in practice. In addition, it is surprisingly versatile and its applications range from vision to proteins to computational geometry to graphs and so on. Most people learn the algorithm and do not use it much that is a pity as a clever use of KNN can make things very simple. It also might surprise many to know that KNN is one of the top 10 data mining algorithms.

KNN is a non-parametric lazy learning algorithm. That is a concise statement. This is useful, as in the real world, most of the practical data does not obey the typical theoretical assumptions made (*e.g.,* Gaussian mixtures, linearly separable *etc.,*). Non-parametric algorithms like KNN come to the rescue here.

It is also a lazy algorithm. What this means is that it does not use the training data points to do any generalization. In other words, there is no explicit training phase or it is minimal. This means the training phase is fast. Lack of generalization means, that KNN keeps all the training data. More exactly, all the training data is needed during the testing phase. (Well this is an exaggeration, but not far from truth). Most of the lazy algorithms – especially KNN – make decision based on the entire training data set (in the best case a subset of them).

The dichotomy is obvious here – There is a nonexistent or minimal training phase but a costly testing phase. The cost is in terms of both time and memory. More time might be needed as in the worst case; all data points might take point in decision. More memory is needed as we need to store all training data. The algorithm on how to compute the K-nearest neighbors is as follows:

1. Determine the parameter K = number of nearest neighbors beforehand. This value is all up to you.
2. Calculate the distance between the query-instance and all the training samples. You can use any distance algorithm.
3. Sort the distances for all the training samples and determine the nearest neighbor based on the K-th minimum distance.
4. Since this is supervised learning, get all the Categories of your training data for the sorted value which fall under K.
5. Use the majority of nearest neighbors as the prediction value.

Figure 3 gives the schematic diagram of the KNN classifier.

KNN assumes that the data is in a *feature space*. More exactly, the data points are in a metric space. The data can be scalars or possibly even multidimensional vectors. Since the points are in feature space, they have a notion of distance – this need not necessarily be Euclidean distance although it is the one commonly used.

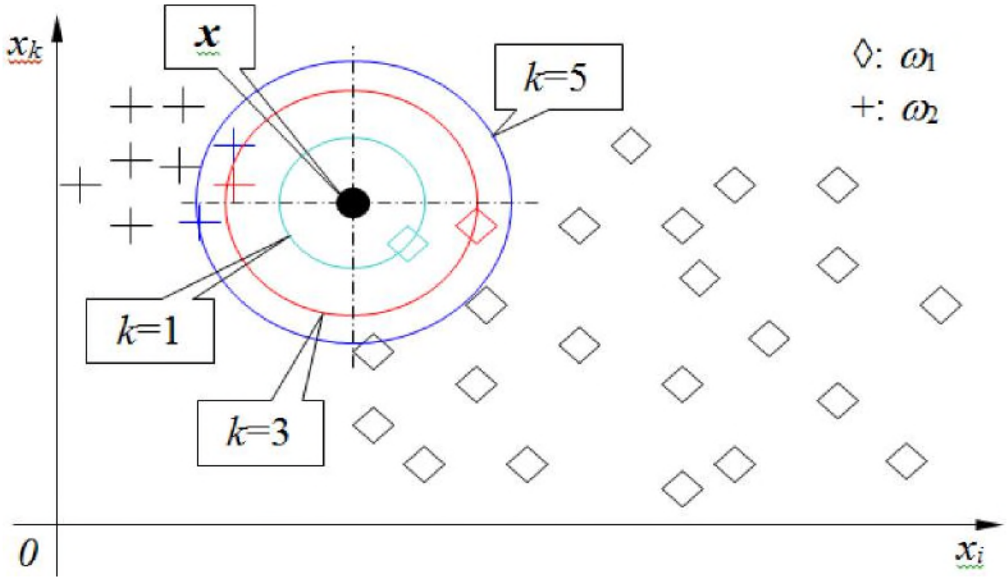

Figure 3. Schematic Diagram of the KNN Classifier

[illegible]

[illegible] [illegible]

2 2

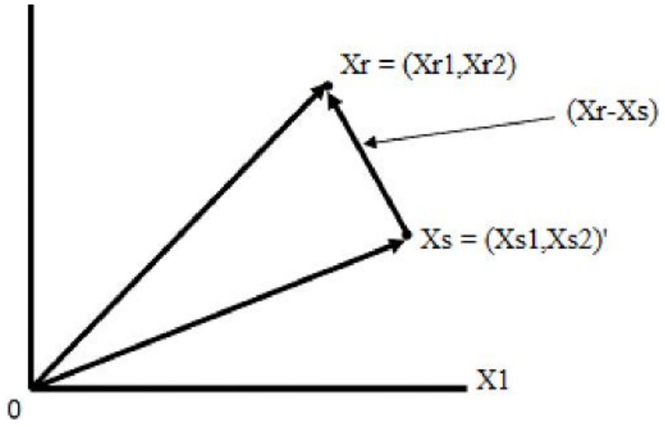

Figure 4. Euclidean Distances between Two Vectors X_r and X_s

We can use the following formula to express the euclidean distances as showed in Figure 4.

$$=|\qquad|=\sqrt{\qquad\qquad\qquad}$$

Each of the training data consists of a set of vectors and class label associated with each vector. In the simplest case, it will be either + or – (for positive or negative classes). But KNN, can work equally well with arbitrary number of classes.

We are also given a single number "k”. This number decides how many neighbors (where neighbors are defined based on the distance metric) influence the classification. This is usually an odd number if the number of classes is two. If k=1, then the algorithm is simply called the nearest neighbor algorithm.

3.1 KNN for Density Estimation

Although classification remains the primary application of KNN, we can use it to do density estimation also. Since KNN is non-parametric, it can do estimation for arbitrary distributions. The idea is very similar to use of parzen window. Instead of using hypercube and kernel functions, here we do the estimation as follows – For estimating the density at a point x, place a hypercube centered at x and keep increasing its size till k neighbors are captured. Now estimate the density using the formula,

——

Where n is the total number of V is the volume of the hypercube. Notice that the numerator is essentially a constant and the volume influences the density. The intuition is this: Let's say density at x is very high. Now, we can find k points near x very quickly. These points are also very close to x (by definition of high density). This means the volume of hypercube is small and the resultant density is high. Let's say the density around x is very low. Then the volume of the hypercube needed to encompass k nearest neighbors is large and consequently, the ratio is low.

The volume performs a job similar to the bandwidth parameter in kernel density estimation. In fact, KNN is one of common methods to estimate the bandwidth (*e.g.*, adaptive mean shift).

3.2 KNN Classification

In this case, we are given some data points for training and a new unlabeled data for

c

testing. Our aim is to find the class label for the new point. The algorithm has different

$P^* \square P \square P^*(2 \square P^*)$

1

behavior based on k. $_c$ □

Case 1: k = 1 or Nearest Neighbor Rule

This is the simplest scenario. Let x be the point to be labeled. Find the point closest to x. Let it be y. Now nearest neighbor rule asks to assign the label of y to x. This seems too simplistic and sometimes even counter intuitive. If you feel that this procedure will result a huge error, you are right – but there is a catch. This reasoning holds only when the number of data points is not very large.

If the number of data points is very large, then there is a very high chance that label of x and y is same. An example might help – suppose that you have a (potentially) biased coin. You toss it for 1 million time and you have head 900,000 times. Then most likely, your next call will be head.

Now, assume all points are in a D dimensional plane. The number of points is reasonably large. This means that the density of the plane at any point is high. In other words, within any subspace there is adequate number of points. Consider a point x in the subspace which also has many neighbors. Now let y be the nearest neighbor. If x and y are sufficiently close, then we can assume that probability that x and y belong to same class is same – Then by decision theory, x and y have the same class.

The book "Pattern Classification" by Duda and Hart has an excellent discussion about this Nearest Neighbor rule. One of their striking results is to obtain a tight error bound to the Nearest Neighbor rule. The bound is

Where * is the Bays error rate, c is the number of classes and P is the error rate of Nearest Neighbor. The result is indeed very striking (at least to me) because it says that if the number of points is large then the error rate of Nearest Neighbor is less than twice the Bays error rate.

Case 2: k = K or k-Nearest Neighbor Rule

This is a straightforward extension of 1NN. What we do is that we try to find the k nearest neighbor and do a majority voting. Typically, k is odd when the number of classes is 2. Let us say k = 5 and there are 3 instances of C1 and 2 instances of C2. In this case, KNN says that new point has to label as C1 as it forms the majority. We follow a similar argument when there are multiple classes.

One of the straightforward extensions is not to give 1 vote to all the neighbors. A very common thing to do is *weighted KNN* where each point has a weight which is typically calculated using its distance. For e.g. under inverse distance weighting, each point has a weight equal to the inverse of its distance to the point to be classified. This means that neighboring points have a higher vote than the farther points.

It is obvious that the accuracy might increase when you increase k but the computation cost also increases.

4. Our Method of Similarity Analysis by Using KNN Algorithm

Given a collection *S* where the target attribute can take on *k* different values, the entropy of *S* can be defined as:

$$Entrop(S) \equiv \sum_{=1} - p \; log_2 \; p$$

(1) Where p_i is the proportion of *S* belonging to class *i*. With this entropy we can calculate the information gain, *Gain*(*S*, *A*) of an attribute *A*, relative to a collection of examples *S* as follows:

$$n(S, \;) \equiv Entrop(S) - \sum_{\in(A)} \frac{|\,|}{|\,|} Entrop(S)$$

(2) Where *Values* (A) is the set of all possible values for attribute A, and S_v is the subset of *S* for which attribute *A* has values *v*. In this equation, the first term is the entropy after *S* is partitioned using attribute *A*. The second term describes the expected entropy which is the sum of the entropies of each subset S_v, weighted by the expected reduction in entropy caused by knowing the value of attribute *A*. The gain ratio incorporates the *split information* [16], that is sensitive to how broadly and uniformly the attribute splits the data:

$$Sptnorton(S,) \equiv - \sum_{=1} \frac{|\,|}{|\,|} log_2 \frac{|\,|}{|\,|}$$

(3) Where S_1 through S_k are the *k* subsets of examples resulting from partitioning *S* by the *k*-valued attribute *A*. The *GainRatio* measure is defined in terms of the *Gain* measure, as well as the *SplitInformation* that discourages the selection of attributes with many uniformly distributed values:

$$nto(S, \;) \equiv \frac{(,)}{(,)}$$

(4) For attributes with continuous values, new discrete-valued attributes are dynamically defined that partition the continuous attribute value into a discrete set of intervals. For an attribute *A* that is continuous-valued, the algorithm can dynamically create a new boolean attribute A_c that is true if $A < c$ and false otherwise. The threshold *c* is the value that produces the greatest gain ratio. To find the threshold *c*, the collection of examples *S* is first sorted on the values of the attribute *A* as $\{v_1, v_2, \ldots, v_m\}$. Any threshold value lying between v_i and v_{i+1} can split *A*, so there are only $m - 1$ candidate thresholds. These candidate thresholds can then be evaluated by computing the gain ratio of each candidate threshold.

In equations (2), (3), (4) it is assumed that the values of the attributes are known. When the value of an attribute is unknown, it is not possible to calculate the gain, the split information, and thus the gain ratio of an attribute. To calculate the gain of an attribute whether the values is known or not, the gain has to be modified as follows [12]:

Let *F* be the fraction that the value of an attribute *A* is known. Then the gain can be calculated as:

$$n(S,) \equiv probability \quad is \quad known \quad * \quad Entrop(S) \quad {}_{\in(A)} + probability \; is \; not \; known * 0$$

$$- \sum K|S| \; |S| \; Entrop(S\rrbracket \;)$$

$$\equiv F * (Entrop(S) - \sum_{\in(A)} \frac{|\,|}{|\,|} Entrop(S))$$

(5) Where only the known values of *A* are taken into account by $Entrop(S)$ and $\sum_{\in(A)} \frac{|\,|}{|\,|} Entrop(S\,)$. The split information can be modified by considering the

cases of *A* with unknown values as an extra group:

$$\mathrm{SplitInformation}(S,A)\ E\ -\sum_{L=1}^{+1} \frac{|s|}{|s|}\, j''''' \grave{}\text{-'}5_2 \frac{Isi|}{|s|} \qquad (6)$$

The attribute that best classifies the training examples is selected and used as the test at the root node of the tree. A child node of the root node is created for each of the two subsets that are split by that attribute and its threshold, and the training examples are sorted with weights for each case to the appropriate child node. If the case has a known value, the weight for that case is 1. If the case does not have a known value, the weight for this case is the probability that this case will have the outcome of the appropriate child node. The subsets that are created for the root node are collections of possible fractional cases. This process is then repeated using the training examples associated with each child node to select the best attribute to test at that point in the tree. During this process, the algorithm never backtracks to consider earlier choices.

The recommender systems are characterized by managing large dataset. One of the most important challenges for them after giving good recommendations is to work with this amount of data in a reasonable computing time.

In order to test how the built system works from the computational time point of view, we run some tests as explained in the previous sections.

The results obtained are as showed in Table 2:

Table 2. Similarity of Different User (u) and Product (p)

File	100u	500u	1000u	2000u	3000u	4000u	5000u	6000u
	100p	500p	1000p	1500p	2000p	2500p	3000p	3900p
SimFor1	0.884	0.893	0.896	0.895	0.894	0.894	0.891	0.885
SimFor2	0.896	0.887	0.897	0.895	0.893	0.894	0.891	0.885
SimFor3	0.886	0.892	0.895	0.896	0.895	0.894	0.891	0.884
SimFor4	0.901	0.890	0.899	0.895	0.893	0.895	0.891	0.885

After these results we calculate the values for the hypothesis contrast as in Table 3:

Table 3. Similarity of Different user (u) and Product (p)

File	100u	500u	1000u	2000u	3000u	4000u	5000u
	100p	500p	1000p	1500p	2000p	2500p	3000p
SimFor1- mSimFor1 -morm SimFor1- SimFor2- m morm SimFor2- SimFor3- m	-0.66	1.69	-0.24	0.46	0.41	0.73	-0.97
	-0.09	0.35	1.13	-0.64	-1.73	-0.16	-0.39
	-0.95	0.90	-1.67	1.04	1.29	-1.45	-0.23
	0.56	-1.48	1.31	-1.08	-2.04	-0.95	0.55
	-0.29	-0.84	-1.39	0.54	0.83	-2.30	0.70
	-0.85	0.62	-2.59	1.69	2.95	-1.36	0.15

There are differences between the similarity formulas 2, 4 and 5, and the precision is significant better in one formula and with other files it is the opposite, as showed in Figure 5.

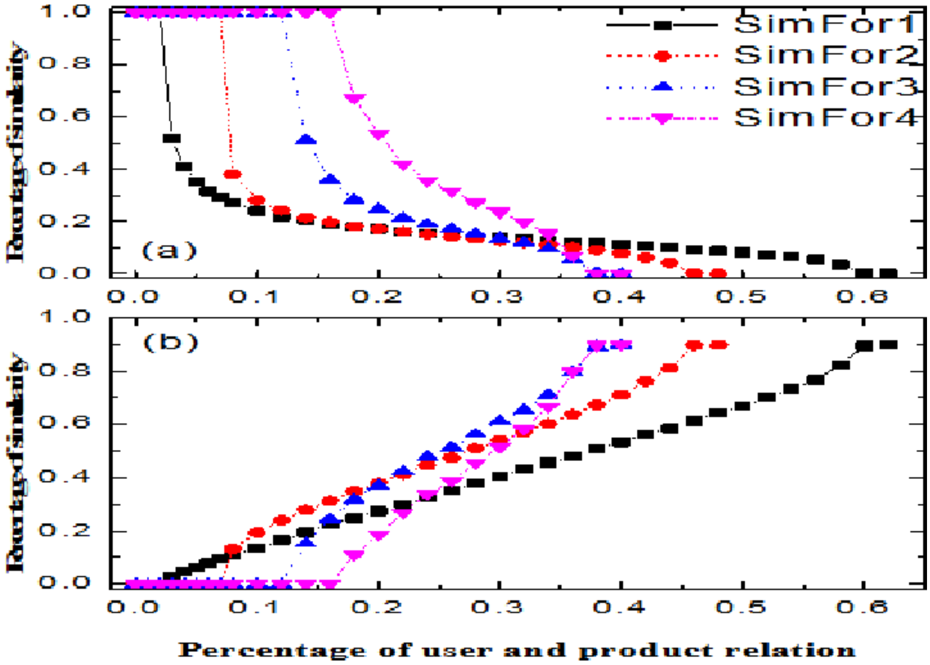

Figure 5. Simulation Results for Tables, 2 and 3

5. Conclusions and Future Work

A variety of approaches to information overload in recommender system have been proposed in the paper. First, it introduces recommender system and CF models. Second, it actually implemented KNN algorithm in the system by using KNN algorithm. Third, the evaluation results of the KNN algorithm point out that based on the basic recommendation methods, the approach may be a better policy due to the balancing issue among accuracy, prediction coverage and system run-time. The experimental results, as well as the analysis of the users' perception showed this approach has a positive impact on recommender systems.

We demonstrate the applicability of association rules in a different domain: user and product. In the future work, our approach could be improved by allowing the manager the specification of more constraints to the recommender system, in addition to the user level and product pool constraints. Other hybridization methods could also be explored to see how these methods perform compared to each other and to the content-based and collaborative recommender systems.

The optimization could also be done for each simulation step separately on as a part of the training set to see if the performance will improve. This way each user would have its own optimized Weak-parameter values.

Acknowledgments

The work was sponsored by The National Natural Science Foundation of China (Grant No.61003234, No. U1304606), the Research Foundation for Humanities and Social Sciences at Universities in Jiangxi Province, China (No. JC1428).

References

[1] M. R. McLaughlin and J. L. Herlocker, “A collaborative filtering algorithm and evaluation metric that accurately model the user experience,” in Proceedings of 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR ’04), **(2004),** pp. 329-336, Sheffield, UK.

[2] M. Deshpande and G. Karypis, “Item-based top-N recommendation algorithms,” ACM Transactions on Information Systems, vol. 22, no. 1, **(2004),** pp. 143-177.

[3] K. Miyahara and M. J. Pazzani, “Improvement of collaborative filtering with the simple Bayesian classifier,” Information Processing Society of Japan, vol. 43, **(2002),** pp. 11-18.

[4] X. Su, M. Kubat, M. A. Tapia and C. Hu, “Query size estimation using clustering techniques,” in Proceedings of the 17th International Conference on Tools with Artificial Intelligence (ICTAI ’05), **(2005)** November, pp. 185-189, Hong Kong.

[5] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite and C. Kadie, “Dependency networks for inference, collaborative filtering, and data visualization,” Journal of Machine Learning Research, vol. 1, no. 1, **(2001),** pp. 49-75.

[6] D. Nikovski and V. Kulev, “Induction of compact decision trees for personalized recommendation,” in Proceedings of the ACM Symposium on Applied Computing, vol. 1, **(2006),** pp. 575-581, Dijon, France.

[7] P. Melville, R. J. Mooney and R. Nagarajan, “Contentboosted collaborative filtering for improved recommendations,” in Proceedings of the 18th National Conference on Artificial Intelligence (AAAI ’02), **(2002),** pp. 187-192, Edmonton, Canada.

[8] B. Schafer, D. Frankowski, J. Herlocker and S. Sen, “Collaborative Filtering Recommender Systems”. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.): The Adaptive Web: Methods and Strategies of Web Personalization. Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg New York, vol. 4321, **(2007)**.

[9] X. Su and T. M. Khoshgoftaar, “A survey of collaborative filtering techniques. Advances in Artificial Intelligence”, vol. 2009, **(2009),** pp. 1-20.

[10] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, "Item-Based Collaborative Filtering Recommendation Algorithms", in Tenth International World Wide Web Conference (WWW10), **(2001),** pp. 285-295, Hongkong, China.

[11] J. Quan and Y. Fu, "A Novel Collaborative Filtering Algorithm Based on Bipartite Network Projection", JDCTA, vol. 6, no. 1, **(2012),** pp. 391-397.

[12] A. Selamat and S. G. Moghaddam. "Improved Collaborative Filtering on Recommender Based Systems Using Smoothing Density-Based User Clustering", IJACT, vol. 4, no. 13, **(2012),** pp. 352-359.

[13] J. Ye, L. Wang, J. Chen and W. Chen, "Algorithm Based on The Interest of The User, Collaborative Filtering and Resources Clustering", JDCTA, vol. 6, no. 21, **(2012),** pp. 472-481.

[14] Z. Yao and F. Yu-qiang, "Hybrid Recommendation method IN Sparse Datasets: Combining content analysis and collaborative filtering", JDCTA, vol. 6, no. 10, **(2012),** pp. 52-60.

[15] Z. Chen, Y. Jiang and Y. Zhao. "A Collaborative Filtering Recommendation Algorithm Based on User Interest Change and Trust Evaluation", JDCTA, vol. 4, no. 9, **(2010),** pp. 106-113.

[16] T. M. Mitchell, “Machine learning”, McGraw-Hill, New York, NY, USA, **(1997)**.