Improving K-nearest-neighborhood based Collaborative Filtering via Similarity Support

[1]Xin Luo, [2]Yuanxin Ouyang, [2]Zhang Xiong
*1. Corresponding Author *College of Computer Science, Chongqing University, Chongqing 400044, China, luoxin21@cqu.edu.cn*
[2]School of Computer Science, Beihang University, Beijing 100191, China
{oyyx, xiongz}@buaa.edu.cn
doi:10.4156/jdcta.vol5.issue7.31

Abstract

Collaborative Filtering (CF) is the most popular choice when implementing personalized recommender systems. A classical approach to CF is based on K-nearest-neighborhood (KNN) model, where the precondition for making recommendations is the KNN construction for involved entities. However, when building KNN sets, there exits the dilemma to decide the value of K --a small value will lead to poor recommendation quality, whereas a large one will result in high computational complexity. In this work we firstly empirically validate that the suitable value of K in KNN based CF is affected by the number of the totally involved entities, and then focus on optimizing the KNN building process for providing high recommendation performance as well as maintaining acceptable KNN size. To achieve this objective, we propose a novel KNN metric named Similarity Support (SS). By taking SS into consideration, we designed a series of strategies for optimizing the KNN based CF. The empirical studies on public large, real datasets showed that due to the improvement on KNN construction brought by SS, recommender optimized by our strategies turned out to be superior to original KNN based CF in terms of both recommendation performance and computational complexity.

Keywords: *Recommender System, Collaborative Filtering, Similarity Support*

1. Introduction

Recommender systems which can provide people with suggestions according to individual interests usually rely on Collaborative Filtering (CF) [1-3]. In CF recommenders, the user's preferences on involved items are quantized into a user-item ratings matrix, where high ratings denote strong preferences. So the problem of CF can be regarded as missing data estimation, in which the main task is to estimate the unknown user-item entries in the ratings matrix based on known entries. According to recent progress in CF techniques, there are two primary approaches to build a CF recommender: Neighborhood Based Model (NBM) [4-7] and Latent Factor Model (LFM) [8-10]. NBM constructs the relationship between users or items to build the neighborhoods of corresponding entities, and makes predictions based on the active user's neighbors' existing ratings, or the existing ratings assigned to the active item's neighbors. Different from NBM, LFM applies matrix factorization techniques to analyzing the ratings matrix: it transforms both items and users to the same latent factor space, characterizes each entity with a feature vector inferred from the existing ratings, and makes a prediction of unknown ratings using the inner product of the corresponding vector pair. Compared to NBM, LFM offers highly expressive ability to describe various aspects of the data. However, NBM has advantages in providing high flexibility to address different aspects of the data and to integrate other models, as well as being intuitively explainable for recommendations [7]. So real world applications mostly rely on NBM based CF recommender.

A classical approach to NBM based CF is based on *K*-nearest-neighborhood (*K*NN) model, where the most important task is building the *K*NN for users/items ranked by the rating similarity [4-7]. In traditional *K*NN based methods, there is a dilemma to decide the value of *K* when building the *K*NN sets--a small value of *K* will lead to poor recommendation performance, whereas a large one will result in unacceptable computational complexity. In this work we focus on optimizing the *K*NN set size, and propose the Similarity Support, a novel metric which can play a supplementary but important role when building the *K*NN sets. Based on this metric, we propose a series of strategies that can provide

slightly higher recommendation accuracy and density along with much lower computational complexity at the same time. The main contributions of this paper are:

- The assumption that in *K*NN based CF recommender, the optimal value of *K* depends on the number of totally involved entities; along with corresponding empirical validations.
- A novel metric named Similarity Support which can help improving the quality of *K*NN sets. Based on this metric we propose a series of strategies to optimize the *K*NN size while providing high recommendation performance.
- Experimental evaluation of our strategy on large, real datasets.

The remainder of this paper is organized as follows. Section 2 gives the preliminaries. Section 3 proposes the methods. The experiments are presented in Section 4. Finally, Section 5 concludes.

2. Preliminaries

The problem of CF can be defined as follows: given an item set I and a user set U, then users' preferences on items could be denoted by a $|U|\times|I|$ matrix R where each row vector denotes a specified user, each column vector denotes a specified item and each entry $r_{ui} \in R$ denotes the user u's preference on item i (usually high value for strong preference). Given a finite set of ratings $T \subset R$ as the training set, the task is to construct a heuristic estimator, so-called recommender, which can estimate the unknown ratings with minimal accumulative error based on known ratings in T. The accumulative error of the recommender is evaluated on a validation set V of which the data are naturally not used during the creation of this recommender.

NBM is a common approach to implementing a CF recommender system because of its scalability and intuitive nature. In NBM, modeling the relationship between items or users is a precondition for making predictions on unknown user-item pairs. Depending on which kind of relationship is going to be built, NBM could be further divided into item-oriented attitudes and user-oriented attitudes. Since in a real world application the count of items is usually far less than that of users, modeling the item-item relationship is a common choice [4-7].

One classical approach to model such relationship is by calculating the vector similarity between the rating vectors applied to corresponding item pair. The vector similarity is usually measured by Cosine, Pearson Correlation and Adjusted Cosine similarity [1]. Then the system will remember each item's *K*NN set ranked by the trained similarity. The predictions on unknown user-item pairs are generated based on the active user's given rating set and the specified item's *K*NN set as follows:

$$\hat{r}_{ui} = \frac{\sum_{j \in KNN(i) \bigcap R(u)} rs_{ij} \cdot r_{uj}}{\sum_{j \in KNN(i) \bigcap R(u)} rs_{ij}}, \tag{1}$$

where *KNN(i)* denotes the *K*NN set of item *i*, *R(u)* denotes the known rating set by user *u* and rs_{ij} denotes the *Rating Similarity* (*RS*) between item *i* and item *j*. It is a common choice to measure rs_{ij} using the Pearson Correlation similarity, which is formally given by:

$$rs_{ij} = \frac{\sum_{u \in R(i) \cap R(j)} (r_{ui} - \overline{r_i}) \cdot (r_{uj} - \overline{r_j})}{\sqrt{\sum_{u \in R(i) \cap R(j)} (r_{ui} - \overline{r_i})^2} \cdot \sqrt{\sum_{u \in R(i) \cap R(j)} (r_{uj} - \overline{r_j})^2}}, \tag{2}$$

where *R(i)* and *R(j)* respectively denote the given rating sets on item *i* and item *j*, $\overline{r_i}$ and $\overline{r_j}$ respectively denote the average ratings on item *i* and item *j*. Prediction rule (1) can be improved by pre-removing the global average rating and observed deviations [6], formally given by:

$$\hat{r}_{ui} = \mu + b_u + b_i + \frac{\sum_{j \in KNN(i) \bigcap R(u)} rs_{ij} \left(r_{uj} - (\mu + b_u + b_j) \right)}{\sum_{j \in KNN(i) \bigcap R(u)} rs_{ij}}, \tag{3}$$

where μ denotes the global average rating of the training set and b denotes the observed deviation of corresponding subject from μ. Generally, these parameters could be estimated by:

$$\begin{aligned} \mu &= \sum_{(u,i) \in T} r_{ui} \Big/ |T|, \\ b_i &= \sum_{(u,i) \in T} (r_{ui} - \mu) \Big/ \left(\beta_1 + |R(i)| \right), \\ b_u &= \sum_{(u,i) \in T} (r_{ui} - \mu - b_i) \Big/ \left(\beta_2 + |R(u)| \right), \end{aligned} \tag{4}$$

where β_1 and β_2 are constants decided by cross validation for shrinkage control from overfitting. We can use $b_{ui} = \mu + b_u + b_i$, and $R_i^K(u) = KNN(i) \bigcap R(u)$ where K denotes the number of K-Nearest-Neighbors, to substitute into (2) for simplification as:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in R_i^K(u)} rs_{ij} \cdot (r_{uj} - b_{uj})}{\sum_{j \in R_i^K(u)} rs_{ij}}. \tag{5}$$

Recommendation rule (5) seems reasonable: the system will firstly recommend the items most similar to those highly scored by the active user. However, there is a dilemma of deciding the size of KNN set for each item: a small value of K will lead to poor recommendation performance, whereas a large one will cause unacceptable computational complexity. Pioneering research suggests that the empirical optimal value for K should lie in the range of [30, 60] interval [4-6], and if K exceeds this range, the recommendation accuracy will hardly increase. Nonetheless, our empirical study given in the following section indicated that this range for optimal value of K was not always reliable when the number of involved items got too large.

And moreover, traditional methods intend to construct the KNN sets of items only depending on the rating similarity, and it is very common for the specified item to have highly correlated neighbors being based on very small numbers of co-rating users. Usually, these correlations based on few observations are proved to be highly responsible for terrible recommendations [11]. From these two points of view, we propose a novel KNN metric, the Similarity Support (SS), which is defined as follows:

Definition 1. *The Similarity Support (SS) is the number of supporting observations for the Rating Similarity. The SS between item i and item j is the count of users who have co-rated both items, given by* $ss_{ij}=|U_{ij}|$*, where* U_{ij} *denotes the set of users who have co-rated item i and item j.*

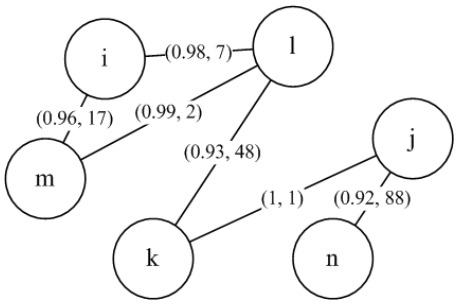

Figure 1. Measuring the relationships between items with the 2-tuples (RS, SS)

Then we can build the *K*NN set for items based on two metrics: *RS* and *SS.* When modeling the relationship between items, high values of *RS* indicate strong agreements on rating patterns, and high values of *SS* denote perfect matches on rating preferences. Actually, if we denote the correlations between items with an undirected weighted graph, then the weight on each edge could be denoted by the 2-tuple (*RS*, *SS*) as shown in Fig.1. And it is reasonable to obtain some improvement in recommendation quality if we can construct the *K*NN set with consideration on both *RS* and *SS* simultaneously and seamlessly.

There are a few literatures on this point. In [11], Herlocker et al. propose a strategy to adjust the user rating similarity according to the number of co-rated items. In [6], Bell et al. propose a shrinkage rule to shrink the vector similarity based on the count of the supporting observations. However, their strategies differ from ours significantly: i) they don't explicitly give the formal definition of Similarity Support; ii) both their strategies need to set a predefined threshold that can be only decided by cross-validation, which will introduce additional computational complexity, whereas our strategy does not have this drawback. We will discuss our strategy in detail in the following section.

3. Methods

In this section we will first validate the assumption that the optimal value for *K* is closely related to the total number of totally involved items, and then we will describe our strategy in detail.

3.1. Assumption Validation

Pioneering research advocates that when building the *K*NN set of items, the empirical optimal value for *K* should between the [30, 60] interval [4-6], any value exceeds this interval will hardly bring improvement in recommendation performance. However, we found in practice that this assertion was not very precise when the number of involved items became too large. So we assumed that the optimal *K* value was closely related to the total number of involved item: when the size of the item set got larger, then a larger value of *K* is necessary for maintaining recommendation quality. In order to validate this assumption, we designed a simple experiment which was set as follows:

Dataset: We chose MovieLens 10K Dataset (hereafter referred to as ML10K) to validate our assumption. This dataset is collected by the GroupLens Research Project at the University of Minnesota through the MovieLens web site, and contains 10K ratings by 943 users on 1682 items with a rating scale lying in the range of [0, 5].

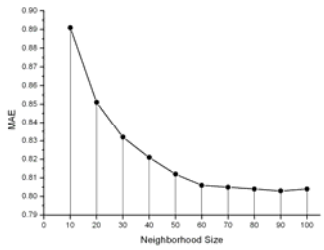

Figure 2. MAE on I100

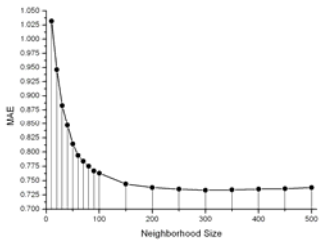

Figure 3. MAE on I500

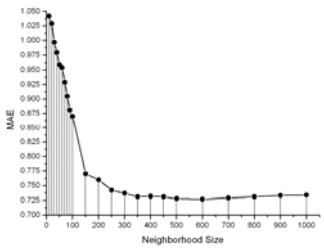

Figure 4. MAE on I1000

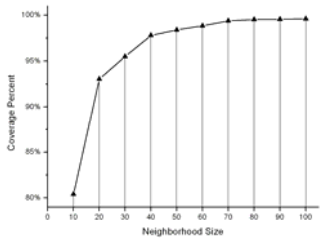

Figure 5. Coverage on I100

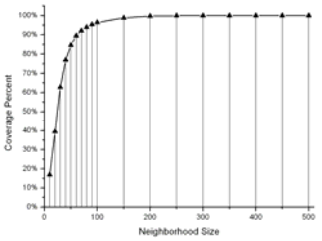

Figure 6. Coverage on I500

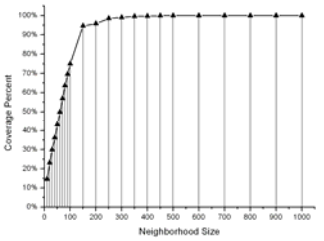

Figure 7. Coverage on I1000

Evaluation Metrics: We chose Mean Absolute Error (MAE) and Coverage as the evaluation metrics. MAE is a widely used metric for evaluating the statistical accuracy of recommenders [12]. To compute the MAE of a recommender system on given dataset, it is necessary to firstly compute the Mean User Absolute Error (MAUE) of each user, which is given by:

$$MAUE_u = \sum\nolimits_{i \in IP(u) \cap IR(u)} | r_{ui} - \hat{r}_{ui} | \Big/ | IP(u) \cap IR(u) |, \tag{6}$$

where *IP(u)* denotes the prediction set generated by the recommender for user *u*, and *IR(u)* denotes the item set rated by user *u* in the testing data. Then the MAE of the recommender can be computed as:

$$MAE = \sum\nolimits_{u \in U} MAUE_u \Big/ | U |. \tag{7}$$

The coverage of a recommender is a measure of the domain of items in the system over which the system can form predictions or make recommendations [12]. The Coverage of a recommender can be formally given by:

$$Coverage = \sum_{u \in U} \left| IP(u) \cap IR(u) \right| \Big/ \sum_{u \in U} \left| IR(u) \right|. \tag{8}$$

Set Up. We sequentially selected 100 items, 500 items and 1000 items by random and extract corresponding ratings from the M10*K* dataset. These three experimental datasets were respectively labeled as I100, I500 and I1000. Each dataset was randomly split to generate a 90%-10% train-test setting. We built *K*NN based CF recommenders by training person similarity between items on the training set and recorded the MAE and Coverage on the testing set with the value of *K* increasing.

Results. Fig.2, Fig.3 and Fig.4 respectively depict the plots of MAE with *K* increasing on I100, I500 and I1000. It is obvious that on I100, the optimal value for *K* was 60, right in the range of [30, 60]. However, this rule did not work on I500 and I1000: on I500, the suitable value for *K* was around 200; and on I1000, the corner point for *K* was around 350.

The plots of Coverage with *K* increasing on I100, I500 and I1000 are respectively depicted in Fig.5, Fig.6 and Fig.7. From the plots it is obvious that the optimal value for *K* which can make the recommender obtain satisfying recommendation density with reasonable computational complexity on I100, I500 and I1000 were respectively around 70, 200 and 350.

Assumption Validation Conclusion. Based on the experiment results, we can draw the conclusion that our assumption of the relationship between the optimal size of *K*NN set and the count of involved items in *K*NN based CF turns out to be correct.

3.2. Improving KNN based CF by SS

From Definition 1, we can find that *SS* actually denotes the credibility of the corresponding *RS* between items, where a high value of *SS* indicates a strong credibility of the corresponding *RS* and vice versa. So it is reasonable to improve the quality of the *K*NN sets if we can employ *SS* and *RS* simultaneously and seamlessly during the *K*NN building process, and it is expectable that the improvement on the *K*NN quality will finally result in the improvement on the recommendation quality. So we designed several strategies to combine these two metrics, as described in the following.

SS Ranking (SSR). This strategy simply replaced *RS* with *SS* during the *K*NN building process: the system would reserve the top-*K* neighbors ranked by *SS* to construct the specified item's *K*NN set. *SS* would not affect the recommendation process: the system will only use *RS* to generate prediction ratings like in recommendation rule (5).

SSR simply took advantage of the fact that *SS* actually described the credibility of corresponding *RS*. It was quite similar to the interpersonal relationship: people usually cannot evaluate the credibility of others based on few contacts; on the other hand, if two persons have frequent contacts with each other, they will probably regard each other as reliable. So it was also reasonable to rank neighbors of the

specified item with *SS:* high *SS* denoted the "frequent contacts" with the active item, which standed for high reputation of the corresponding neighbor to the specified item, and vice versa.

Interval Significance Weighting (ISW). ISW transformed the *RS* between items with consideration on the value of corresponding *SS*, formally given by:

$$rs'_{ij} = rs_{ij} \cdot \frac{ss_{ij} - \min(ss)}{\max(ss) - \min(ss)}, \tag{9}$$

where min(*ss*) and max(*ss*) respectively denote the minimum value and the maximum value of *SS* on the training set. This strategy would cause a vast spread in the values of *RS* between items, but from recommendation rule (5) we can find that the numerical change in *RS* could be canceled out by the divisor, so it would not cause magnitude difference between recommendation results. ISW would affect the *K*NN building process as follows: the system would prefer the training examples with relatively low *RS* but high *SS* with active item to those with high *RS* but rather low *SS.*

Gaussian Weighting (GW). We designed this strategy by modeling the SS values of the whole involved item set as Gaussian random variables $N(\mu,\sigma^2)$. We could estimate the parameters μ and σ^2 using the maximum likelihood estimator as follows:

$$\hat{\mu} = \overline{ss} = \frac{1}{N}\sum ss_{ij},\ \hat{\sigma}^2 = \frac{1}{N}\sum (ss_{ij} - \overline{ss})^2, \tag{10}$$

where N denotes the overall count of SS values on the given training dataset. Then we could use the Cumulative Distributive Function (CDF) to denote the Gaussian weight between each item pair, formally given by:

$$gw_{ij} = F(ss_{ij} \mid \hat{u}, \hat{\sigma}^2) = \int_{-\infty}^{ss_{ij}} f(ss \mid \hat{u}, \hat{\sigma}^2) dss, \tag{11}$$

where $F(ss_{ij} \mid \hat{u}, \hat{\sigma}^2)$ denotes the CDF, and $f(ss \mid \hat{u}, \hat{\sigma}^2)$ denotes the probability density function. Actually, the value of gw_{ij} denotes the probability that the corresponding ss_{ij} is greater than or equal to others. Then we could adjust each *RS* by multiplying corresponding Gaussian weight, formally given by

$$rs'_{ij} = rs_{ij} \cdot gw_{ij} \tag{12}$$

The GW strategy would adjust the *K*NN building process as follows: the system would prefer neighbors with extremely high SS with active item, and vice versa; most neighbors with middle *SS* values would be assigned Gaussian weight around 0.5, which would have little effect on the rank inside the specified item's neighborhood.

4. Experiments

4.1. General Settings

Dataset description: ML10K Dataset--We used the whole ML10K dataset mentioned in Section 3.1 for the first part of our experiment.

Netflix Dataset--This is the dataset for the Netflix Prize. This dataset contains over 100 million ratings from 480 thousand randomly-chosen, anonymous Netflix customers over 17K movie titles. The rating scale lies in the range of [1, 5]. In our experiment, we extracted a subset consisting of the ratings on the first 1000 movies. This subset is hereafter referred to as NF5M, which contains over 5 million ratings by 400*K* users.

Evaluation Metrics: We also chose MAE and Coverage mentioned in Section 3.1 as the performance evaluation metrics.

Training and Testing: We implemented the unmodified *K*NN model based CF as the benchmark, of which the recommendation rule was given by (5). CF based on the *K*NN model adjusted by SSR, ISW and GW were respectively implemented and tested. We chose Pearson Correlation similarity as the similarity measurement between items. Either dataset was randomly split to generate a 90%-10% train-test setting.

4.2. Results and Discussion

Results: Fig.8 depicts the MAE comparison on ML10K with the value of *K* increasing, and Fig.9 depicts the Coverage comparison result on ML10K. In all legends, CF denotes the benchmark; SSR_CF, ISW_CF and GW_CF respectively represent CF based on the KNN model adjusted by SSR, ISW and GW strategies.

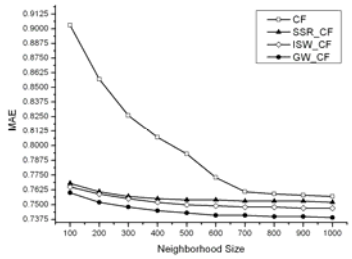

Figure 8. MAE Comparison on ML10K

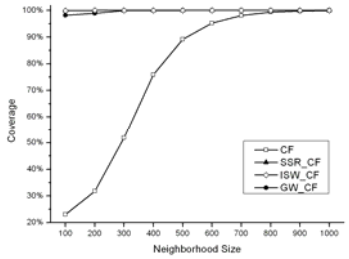

Figure 9. Coverage Comparison on ML10K

As depicted in Fig. 8 and Fig.9, the effect of *SS* based strategies on improving the recommendation accuracy was quite encouraging. Compared to the benchmark, SSR_CF, ISW_CF and GW_CF all had significant advantage in decreasing the MAE. More importantly, these strategies converge much faster than the unmodified model. CF adjusted by the best strategy GW could obtain even better recommendation accuracy, along with equal coverage with *K*=200 compared to the benchmark with *K*=1000, which means 80% improvement on the computational complexity over the traditional *K*NN model while providing even better recommendation quality.

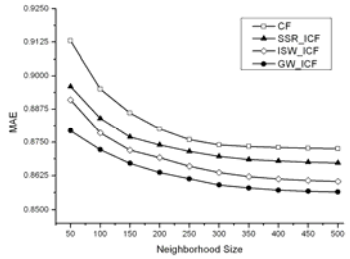

Figure 10. MAE Comparison on NF5M

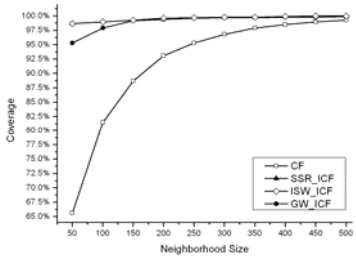

Figure 11. Coverage Comparison on NF5M

Fig.10 and Fig.11 respectively depict the MAE and Coverage comparison on NF5M with the value of *K* increasing. The experiment result on NF5M was nearly the same with that on ML10K. GW_CF also had the leading position, by providing better performance and 80% improvement on computational complexity compared to the benchmark.

Besides the improvement on computational complexity, another characteristic of GW_CF was that it could provide quite significant improvement on the recommendation accuracy, partially because it applied a reliable model to adjust the *RS* between items: when building the *K*NN set for the active item, most of its neighbors with common *SS* values would be assigned to median Gaussian weights around the average, while those with extremely high/low *SS* values would have much greater/less importance than others.

Based on the experiment results, it is obvious that once taking *SS* into consideration, the recommender could achieve high recommendation accuracy and density while maintaining reasonable *K*NN size. To give a more plain view, we pick out the data points of *K* at which the benchmark could achieve the highest recommendation accuracy and density in both experiments with comparison to those points at which the adjusted methods could obtain similar performances. These data are summarized in Table 1 and Table 2.

Table 1. Performance Comparison on ML10K

Strategy	MAE	Coverage	K
CF	0.7573	100%	1000
SSR-CF	0.7569	100%	300
ISW-CF	0.7554	100%	300
GW-CF	0.7521	98.9%	200

Table 2. Performance Comparison on NF5M

Strategy	MAE	Coverage	K
CF	0.8726	99.3%	500
SSR-CF	0.8716	99.7%	250
ISW-CF	0.8720	99.3%	150
GW-CF	0.8722	97.9%	100

Complexity analysis: Based on the experiment results, we can find that introducing *SS* into the *K*NN building process is very effective in reducing the computational complexity while providing even better recommendation performance during the recommending process. However, this will also increase the computational complexity during the training process of the recommender. In traditional *K*NN based CF recommender, the time complexity and space complexity during the training process are both $O(n^2)$. Due to the introduction of *SS*, the computational complexity will increase in two aspects:

- The storage of SS will require an extra space of $O(n^2)$, so the space complexity is twice of that in traditional KNN based CF.
- In ISW and GW, since the system need to adjust each *RS* according to the corresponding *SS*, the system needs to iterate one more time over the trained *RS* set, which will cost an extra $O(n^2)$ of time.

5. Conclusion

One critical task for traditional KNN based CF recommender is building the *K*-nearest-neighborhood for items, where the dilemma to decide the value of K exists: a small value of *K* will lead to poor recommendation accuracy, whereas a large one will result in unacceptable computational complexity. Pioneering research on item-oriented collaborative filtering suggests that the optimal value of *K* should between the [30, 60] interval, and if the neighborhood size exceeds this interval, the improvement on accuracy will be quite limited. However, we have empirically validated that the optimal value for K relies heavily on the number of involved items, which again make it a problem to optimize the value of *K*. In this work we propose a novel *K*NN metric *SS* to address this problem. By taking this factor into consideration during the *K*NN building process, we designed three strategies SSR, ISW and GW with intention to improve the *K*NN quality by employing *RS* and *SS* simultaneously. The empirical studies on large, real datasets showed that compared to traditional *K*NN based CF, our strategies have significant advantage in alleviating the computational complexity while providing even better recommendation quality.

6. Acknowledgements

The authors are very grateful to the anonymous referees for their informative and insightful reviewing comments which lead to the further improvement of this work. This work is fully supported by National Key Technology R&D Program of China under Grant No. 2011BAH25B01.

7. References

[1] Adomavicius G, Tuzhilin A, "Toward the Next Generation of Recommender Systems: a Survey of the State-of-the-art and Possible Extensions", IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 6, pp.734-749, 2005.
[2] Zhimin Chen, Yi Jiang, Yao Zhao, "A Collaborative Filtering Recommendation Algorithm Based on User Interest Change and Trust Evaluation", International Journal of Digital Content Technology and its Applications, vol. 4, no. 9, pp.106-113, 2010.
[3] Qiang Guo, Caiming Zhang, "Image Interpolation using Collaborative Filtering", International Journal of Digital Content Technology and its Applications", vol. 5, no. 4, pp.12-17, 2011.
[4] Sarwar B, Karypis G, Konstan J, Reidl J, "Item-based collaborative filtering recommendation algorithms", In proceedings of the 10th International Conference on World Wide Web, pp.285-295, 2001.
[5] Deshpande M, Karypis G, "Item-based Top-N Recommendation Algorithms", ACM Transactions on Information Systems, vol. 22, no. 1, pp.143-177, 2004.
[6] Bell M, Koren Y, "Improved neighborhood-based collaborative filtering", In proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.7-14, 2007.
[7] Koren Y, "Factor in the Neighbors: Scalable and Accurate Collaborative Filtering", ACM Transactions on Knowledge Discovery from Data, vol. 4, no. 1, pp.1-24, 2009.
[8] Kurucz M, Benczúr A, Csalogány K, "Methods for large scale SVD with missing values", In proceedings of the KDD Cup Workshop at the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.7-14, 2007.
[9] Paterek A, "Improving regularized singular value decomposition for collaborative filtering", In proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.39-42, 2007.
[10] Takács G, Pilászy I, Németh B, Tikky D, "Investigation of various matrix factorization methods for large recommender systems", In proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.1-8, 2008.
[11] Herlocker J, Konstan J, Riedl J, "An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms", Information Retrieval, vol. 5, no. 4, pp.287-310, 2002.
[12] Herlocker J, Konstan J, Terveen L, Riedl J, "Evaluating Collaborative Filtering Recommender Systems", ACM Transactions on Information Systems, vol. 22, no. 1, pp.5-53, 2004.