HÖGSKOLAN
DALARNA

MASTER THESIS IN MICRODATA ANALYSIS

A Computational model for Trust-based Collaborative Filtering

- An empirical study on hotel recommendations

Author:
Qinzhu Wu

Supervisor:
William Wei Song

June, 2013

Business Intelligence Program
School for Technology and Business Studies
Dalarna University

Abstract

The inherent weakness of the data on user ratings collected from web, such as sparsity and cold-start, has limited the data analysis capability and prediction accuracy in recommender systems (RS). To alleviate this problem, trust has been incorporated in collaborative filtering (CF) approaches with encouraging experimental results. In this paper, we propose a computational model for trust-based CF combined with k-means clustering, k-nearest neighbor (kNN) and three different methods to infer trust, based on a detailed data analysis of hotel dataset collected from *Booking.com*. We apply this model on users' ratings of hotels and show its feasibility by comparing the testing results with conventional CF algorithm using evaluation metrics Mean Absolute Error (MAE) and prediction coverage. Our experimental results indicate that the use of trust can improve prediction accuracy if the definition of trust is reasonable enough.

Keywords*:* Trust, Collaborative filtering, Recommender systems.

Contents

1 Introduction

1.1 Problem description

Recommender systems (RS) are software tools and techniques providing suggestions for items to be use of a user [1]. Although collaborative filtering has proven to be one of the most effective techniques to be incorporated into RS, it still suffers from the inherent weaknesses existed in raw data, such as data sparsity. Data sparsity refers to the situation that users only rate a small portion of the available items, thus resulted in a sparse user-item matrix where we can hardly find co-rated items between users. For instance, in our case, a large number of ratings on hotels were collected from registered users of the well-known accommodation reservation website *Booking.com*[1], however, only 5% of users in the datasets rated more than one hotel, the lack of prior ratings makes it fundamentally difficult to find enough number of similar users and make accurate predictions for an individual with conventional collaborative filtering method. On the other hand, due to the sparse ratings matrix with huge number of null values, large amount of computer memory will be wasted to store the useless values.

Many research works has shown a rising interest in incorporating trust into recommender systems to solve this problem, mainly by quantifying trust into numerical values and build a web of trust (WOT) for each user, using trust inference and trust propagation. The effectiveness of trust has been proved for many times, that it can improve the prediction accuracy efficiently. However, there was not a stationary definition for trust in RS. Thus we want to explore novel methods to define trust and verify the feasibility of trust when combine it with collaborative filtering algorithm, to see how much effect it has on improving the prediction accuracy.

1.2 Solutions

In this paper we propose a computational model with trust-based CF to alleviate the sparsity problem existed in our datasets, as well as verifying the feasibility of using trust. First, large amount of real data was collected from *Booking.com* and has been analyzed from different angles using JMP software. Secondly, all hotels (items) were classified into different clusters based on their attributes with k-means clustering technique and denoted by their cluster id. This idea is enlightened from our data analysis result, as well as from a literature review [15], which we will introduce in section 2.3 in detail. Thirdly, for each testing user chosen from dataset, we find a group of neighbors (k-nearest neighbor) who have similar preference based on their commonly rated clusters and prior rating patterns. Predictions were made based on the conventional CF formula (section 3.1), in which the rating value can be calculated from previous ratings of neighbors. At last, we evaluate and compare the performance of predictions by two metrics: mean absolute error (MAE) and prediction coverage (section 4.2).

To make the prediction results more accurate and verify the feasibility of trust, trust was incorporated in the third step in computation procedure. Trust is an assumed reliance on some people or things that they will not fail us. Here we developed three methods to infer trust based on users' information based on different assumptions. The first assumption we made is that users whose rating values are more proximate to the total rating value of hotel itself, are more trustworthy than others. Second assumption is based on the group type of a user, e.g., family with children is more trustworthy than solo traveler when giving rating to a hotel. Third assumption is that experienced users who has rated more

[1] Source: http://www.booking.com

items before are more trustworthy. Users who were considered to be more trustworthy were assigned a higher trust value. Then we adjust the CF formula by replacing the weights with trust values of neighbors, to see if the use of trust indeed works well in improving prediction accuracy.

1.3 Evaluations

Evaluation metrics (section 4.2) used in this paper are Mean Absolute Error (MAE) and prediction coverage, which are commonly used within the context of RS. MAE is the average absolute deviation of the actual rating values to the predicted values, thus the lower the MAE, the more accurately the RS predicts. Coverage is the percentage of hidden ratings that an algorithm was able to predict, and a higher coverage indicates better result.

Evaluation results (section 4.4) from our experiments have indicated that although the improvement of prediction accuracy is not very significant, the use of trust indeed helps to improve the accuracy of predicted rating values, as long as the measurement of trust is rational enough, e.g., Trust-1 and Trust-2. Parameters like the number of clusters K also act as an important role in prediction accuracy. Experimental results from our case indicate that K=10 performs better than K=20 and K=30.

1.4 Conclusions

In this paper a computation model for trust-based CF was proposed, in which k-means clustering, k-nearest neighbor and three measurements of trust were combined together to improve prediction accuracy on a sparse rating dataset. Our contributions are mainly focused on the test and verification of trust and an algorithm which avoids the waste of computer memory caused by large amount of null values. Future work of our thesis will be centralized in adjusting the method with other datasets, exploration of a more efficient way to incorporate trust into RS and the use of trust propagation.

The remainder of this paper is organized as follows. Section 2 surveys existing research work on recommender systems, collaborative filtering and the rising interest to incorporate trust into CF. Section 3 provides detailed descriptions on conventional collaborative filtering method, weight computation method for neighbors normally incorporated into CF, as well as our trust-enhanced CF method. In section 4 we introduce our hotel data sets and the sparsity problem caused by it, evaluation metrics including mean absolute error (MAE) and Coverage, algorithm with regard to implementation steps and experimental results with detailed analysis. Section 5 provides a conclusion on what we have done, our contributions and future work. References and appendix will be given in section 6 and 7.

2 Related Work

2.1 Recommender systems

Recommender systems (RS) [7] emerged as an independent research area since the appearance of collaborative filtering in the mid-1990s [8]. The first recommender system, Tapestry [6], originally designed to improve efficiency of E-mail filtering by incorporating other users' opinions in the process, can be traced back to 1992. RS normally focused on giving suggestions on items towards individuals who may lack experiences before. Nowadays it is still a popular area to be developed, not only because it can address the information overload problem, but also owing to the far-ranging applications it has brought to us. Examples of such applications can be found everywhere: helping customers decide which products to purchase in an E-commerce website (Amazon.com [10]), recommending songs to music lovers in a radio website (Last.fm [5]), and mobile recommender systems using spatial data [11].

Usually recommender systems were classified according to techniques that have been incorporated to them. Based on this, we typically have three different types of recommender systems: collaborative filtering (CF), content-based and hybrid system. The main difference between collaborative filtering and content-based approach is that the former one recommends items that other similar users have liked by computing similarity values between users. As for content-based technique, we only recommend items which are similar to the items one user has liked in the past. One typical hybrid recommender system is Fab [3], by combining both techniques, it may alleviate some weaknesses found in each approach. A state-of-the-art introduction of recommender system can be found in [1].

2.2 Collaborative Filtering

Many literature review have indicated that collaborative filtering is one of the most well-known, successful and widely implemented techniques [1, 4, 13, 14, 19]. The biggest advantage of CF over content-based approach is that it only relies on opinions on items described by users [19]. Instead content-based systems require more detailed descriptions of each item, so as to generate similarities between items. Two general classes of CF algorithms were examined in [12]: Memory-based algorithm and model-based algorithm. Model-based algorithm can be viewed as calculating the expected value of a vote from a probabilistic perspective, based on what we know about the user. Related methods include cluster models and Bayesian networks. As for the memory-based algorithm, we will describe it explicitly in section 3.1.

However, CF approach still suffers from three fundamental challenges [20]: data sparsity, cold-start and scalability. Data sparsity refers to the situation that users only rate a small portion of the available items, thus resulted in a sparse user-item matrix where we can hardly find co-rated items between users. In cold-start problem, the lack of historical information occurs on new items or users consequently lead to a 'dumb' state in RS, that the system fails to consider users with an empty file or items no one has previously rated. Scalability entails a large amount of computation when there are millions of users and items, which is usually the case in reality. In this paper we focused on data sparsity and proposed a model to alleviate this problem.

Several approaches have been adopted in previous work to cope with this challenge and received moderately good results. As we mentioned before, hybrid algorithm combing both CF and content-based techniques can alleviate weakness in both approaches.

Dimensionality reduction methods, such as Singular Value Decomposition (SVD), Latent Semantic Indexing (LSI), reduce the dimensions of matrix by getting rid of unimportant users or items [22]. Huang et al [21] applied an associative retrieval framework and spreading activation algorithms to deal with the sparsity problem. Manos et al [17] used trust inference to alleviate this problem.

2.3 Trust in RS

A rising interest in trust-enhanced recommender systems was found in recent research work [1, 14, 15, 17, 19]. Trust is a common concept in our daily life and it can be defined in various ways, B. Noteboom claims that trust is an expectation that things or people will not fail us, or the neglect or lack of awareness of the possibility of failure [23]. Another notion of trust was presented by P. Sztompka, "It is a type of bet taken on the issue of uncertain future activities of other people"[24]. For trust in recommender systems, there is no stationary definition for it. The main strength of applying trust into recommender systems is to quantify trust into numerical values and build a web of trust (WOT) for each user, using trust inference and trust propagation. Examples of major algorithms for building trust network are Moletrust [25] and Tidaltrust [2].

J. Wang [15] proposed a method to generate trust by incorporating the taste of users' on choosing items. The tastes of users were implied from the classification of items, based on the intuition that users usually trust those who have similar taste with them. The more items that a user rated to a certain cluster of items, the more interest a user showed to them. After that the trust metric is developed from the taste set of a user in all clusters of items. Then trust was propagated throughout a social network to include more similar users. Finally ratings were predicted by summing up all rating values from similar users and the results were evaluated by using MAE and Coverage. Results from experiments have indicated that the use of trust can decrease MAE and increase Coverage in a sparse dataset compared with User Similarity-based CF and Item Similarity-based CF. The k-means clustering method we used in our paper is enlightened by the similar taste ideas from this literature.

3 Methodology

3.1 Conventional Method

Normally, the task in collaborative filtering can be of two forms [13]: prediction and recommendation. Prediction is a numeric value expressing the predicted rating score on an item from a particular user (we will denote this user as the active user). Recommendation is to recommend a list of items the active user will like probably. We choose prediction as our task to implement with the hotel data set, i.e., predict rating values within the same scale (e.g., from 0 to 10) for the active user on hotels he or she has no experience before.

Collaborative filtering usually follows two steps [26]:

1. Find neighbors who share the same rating pattern for the active user a.
2. Assign a weight for each neighbor found in the first step of user a and use their ratings to calculate predictions for user a.

The two steps above can be normalized into several equations and we use the classical memory-based CF formula [18] as the basis of our algorithm:

$$p_{a,j} = \bar{r}_a + \frac{\sum_{i=1}^{n} w(a,i)(r_{i,j} - \bar{r}_i)}{\sum_{i=1}^{n} w(a,i)} \tag{1}$$

where $p_{a,j}$ is the predicted rating of the active user a for item j. $\bar{r}_a$ and $\bar{r}_i$ is the mean rating score for user a and user i respectively. The mean rating value and standard deviation for user i can be defined as:

$$\bar{r}_i = \frac{1}{|\boldsymbol{I}_i|}\sum_{j \in \boldsymbol{I}_i} r_{i,j},\ \sigma_i = \sqrt{\frac{1}{|\boldsymbol{I}_i|}\sum_{j \in \boldsymbol{I}_i}\left(r_{i,j} - \bar{r}_i\right)^2} \tag{2}$$

Here $\boldsymbol{I}_i$ is the set of items on which user i has rated and $|\boldsymbol{I}_i|$ represents the number of observations in a set. $r_{i,j}$ is the rating score on item j from user i. In Equation (1), n is the number of users in the collaborative filtering database with nonzero weights. $w(a,i)$ is the weights of n similar users and it can be defined in several different ways:

- K-nearest neighbor

$$w(a,i) = \begin{cases} 1 & if\ i \in \boldsymbol{N}_a \\ 0 & else \end{cases} \tag{3}$$

- Pearson correlation coefficient

$$w(a,i) = \frac{\sum_j \left(r_{a,j} - \bar{r}_a\right)\left(r_{i,j} - \bar{r}_i\right)}{\sqrt{\sum_j \left(r_{a,j} - \bar{r}_a\right)^2 \sum_j \left(r_{i,j} - \bar{r}_i\right)^2}}$$

- Cosine distance

$$w(a,i) = \sum_j \frac{r_{a,j}}{\sqrt{\sum_{k \in \boldsymbol{I}_a} r_{a,k}^2}} \frac{r_{i,j}}{\sqrt{\sum_{k \in \boldsymbol{I}_i} r_{i,k}^2}}$$

Pearson correlation coefficient (PCC) has proven to be the most efficient and accurate way to express the similarity of rating pattern between two users. However, it requires user a and user i have at least two co-rated items, otherwise $w(a,i) = \frac{0}{0}$. For cosine distance, $w(a,i) = 1$ if j equals to 1, i.e., when user a and user i have only one co-rated item, $w(a,i)$ equals to a constant. In consideration of actual situation in our case, most

users lack experience with accommodation in distinct hotels, i.e., it is difficult to find more than one co-rated items between two users. We denote this as the sparsity problem in our data set and we will explain it in detail in section 4.1. In the light of this, we choose k-nearest neighbor as weighting method due to its flexibility in defining neighbors.

The k-nearest neighbors (kNN) rule [9] is one of the simplest and oldest methods for pattern classification. Its performance crucially depends on the distance metrics used to identify nearest neighbors. Equation (3) is a simple expression of how kNN can be combined with CF algorithm. If user i belongs to the neighborhood of user a, i.e., $i \in \boldsymbol{N}_a$, we simply set $w(a,i) = 1$, otherwise $w(a,i) = 0$. The conditions of how can we determine if a user belongs to the neighborhood or not will be introduced in the next paragraph.

To extend the scope of neighbors of users, we use k-means clustering to classify items into K clusters. K-means clustering is a term within the context of data mining, which aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean [27]. Here we partition the item set $\boldsymbol{I}$ into K clusters according to nine numerical attributes of each item: star level, number of total votes, average score, rating values corresponding to five aspects-clean, comfort, location, service, staff and value for money. This step is implemented by JMP software automatically. A detailed description of these attributes can be found in section 4.1. We name this process as 'Fuzzification of items' because all items were fuzzified into K clusters and users who have co-rated hotels in the same cluster can be correlated together. In this sense, much more neighbors can be found even when the active user has voted only one item. According to our data set, the scope of neighbors for an active user a can be defined as follow:

$$\begin{cases} \left||\boldsymbol{C}_a| - |\boldsymbol{C}_i|\right| \le \frac{K}{\theta_1} \\ |\boldsymbol{C}_a \cap \boldsymbol{C}_i| \ge \min\{|\boldsymbol{C}_a|, |\boldsymbol{C}_i|\} \times \theta_2 \\ |\bar{r}_a - \bar{r}_i| \le \theta_3, |\sigma_a - \sigma_i| \le \theta_4 \end{cases} \xrightarrow{yields} i \in \boldsymbol{N}_a \quad (4)$$

Here $\boldsymbol{C}_a$ denotes the set of clusters to which $\boldsymbol{I}_a$ belongs. $|\boldsymbol{C}_a|$ is the number of $\boldsymbol{C}_a$. θ_1 is a threshold parameter combined with K. $\boldsymbol{C}_a \cap \boldsymbol{C}_i$ is the intersection set of clusters that $\boldsymbol{I}_a$ and $\boldsymbol{I}_i$ falls in at the same time. θ_2 refers to a percentage parameter, e.g., 50%. θ_3 and θ_4 are two thresholds for the distance of mean value and standard deviation between two users.

User ID	5698									
Hotel ID	181	642	865	1520	2881	3027	3662	3936	4563	5115
Cluster ID	1	6	18	4	11	20	11	12	11	11
Rating Score	8.8	7.1	9.2	7.5	7.5	9.6	8.3	6.5	7.5	8.3

Table 3-1: Hotel & Cluster Ids of User a (ID: 5698)

User ID	43936									
Hotel ID	1577	2600	2718	2964	3408	3616	3837	3926	4818	5072
Cluster ID	6	20	3	5	12	5	3	3	1	3
Rating Score	7.1	8.3	8.3	10	10	7	10	8.8	10	3.8

Table 3-2: Hotel & Cluster Ids of User i (ID: 43936)

$\boldsymbol{C}_a$	{1, 4, 6, 11, 12, 18, 20}	$\boldsymbol{C}_i$	{1, 3, 5, 6, 12, 20}	$\boldsymbol{C}_a \cap \boldsymbol{C}_i$	{1, 6, 12, 20}
$\lvert\boldsymbol{C}_a\rvert$	7	$\lvert\boldsymbol{C}_i\rvert$	6	$\big\lvert\lvert\boldsymbol{C}_a\rvert - \lvert\boldsymbol{C}_i\rvert\big\rvert$	1
$\bar{r}_a$	8.03	$\bar{r}_i$	8.33	$\lvert\bar{r}_a - \bar{r}_i\rvert$	0.3
σ_a	0.978717982	σ_i	1.977119116	$\lvert\sigma_a - \sigma_i\rvert$	0.998401134

Table 3-3: Computation results of example

Give a specific example to interpret the equation above. Here we have two users' information of a and i about which hotels they have rated previously (Table 3-1 and Table 3-2). From these two tables we can see that they have no co-rated hotels even when they have rated 10 hotels, which is a relatively big number in our dataset. However after all items were classified into K clusters, e.g., $K = 20$, they shared some common cluster ids.

Suppose both of them have voted 10 items, then user a's cluster list for these items might be: 1, 4, 6, 11, 12, 18, 20 (7 clusters) and user i's list is: 1, 3, 5, 6, 12, 20 (6 clusters). If we set $\theta_1 = 10$, so the first condition in Equation (4) is satisfied: $|7 - 6| \leq \frac{20}{10}$. After that we select the common clusters from user a and user i's cluster list: 1, 6, 12, 20 (4 common clusters). So the second condition is satisfied as well (if $\theta_2 = 50\%$): $\min\{7, 6\} \times 50\%$ which equals to 3 is less than 4, the number of common clusters. Finally we compute $|\bar{r}_a - \bar{r}_i|$ and $|\sigma_a - \sigma_i|$ and compare them with θ_3 and θ_4. Here if we set both of them as 0.5, then the third condition in Equation (4) is not satisfied because though they shared a similar mean value, their standard deviation is not similar with each other.

The first equation entails that the two users should have similar level in estimation for items, i.e., if user a rated only one item but user i rated more than 20 they are not at the same level so they can't be similar users, i.e., they are not neighbors. The second requires the two users should share a relatively big proportion of common clusters in items, i.e., they have similar taste when choosing items to use. The third condition indicates that the mean value of ratings from active user a should be similar with user i, i.e., they have similar pattern in rating items, both in the case of mean value and standard deviation.

When implementing the CF algorithm in section 4.3, each time we apply this computation method on the two users that we select from our database, to compute the similarity between them and filter neighbors for the active (testing) user a. And this is the conventional method because we set $w(a, i) = 1$ here. In the next section 3.2, we are going to incorporate trust by changing the weight computation method $w(a, i)$.

3.2 Trust-enhanced Method

After making rating predictions for the items used by users with conventional method, another question was triggered naturally: Can we make the predicted results more accurate? The answer is undoubtedly true. Past experiences in attempting to incorporate the concept of trust into recommender systems have shown satisfactory results in improving prediction accuracy. In some researches, trust information already exists between users in the database therefore there is no need to infer trust value. In our case, trust inference is still a problem that we need to solve. Here we define trust value for a user i as $trust_i$ and it can be generated from three different angles based on our data sets:

- **Trust-1**: $trust_i$ equals to the distance between the mean rating value $\bar{r}_i$ and the total mean rating value $\bar{t}_i$ for user i. As can be shown in Equation (5), we simply define $trust_i$ as the reciprocal of the absolute distance between $\bar{r}_i$ and $\bar{t}_i$ for user i.

$$trust_i = \frac{1}{|\bar{r}_i - \bar{t}_i|} \tag{5}$$

The intuition behind this computation is that if user i's actual rating value is nearly the same as the total rating value then user i is more trustworthy, i.e., $trust_i$ is allocated a relatively large value than other users. On the other hand, if the distance between $\bar{r}_i$ and $\bar{t}_i$ is big, then user i's opinion is less trustworthy. The total rating value t_j can be considered as the most objective and truthful rating value for an item j, as described in section 4.1, due to the fact that it is the average rating value generated by hundreds of users who have used item j. And the mean total rating value $\bar{t}_i$ can be defined as below:

$$\bar{t}_i = \frac{1}{|\boldsymbol{I}_i|} \sum_{j \in \boldsymbol{I}_i} t_j \tag{6}$$

- **Trust-2**: $trust_i$ is determined by the group type of user i, e.g., family with children or couple usually has higher trust value than group of friends. Stemmed from this idea, we assigned different values for each user according to their group type and listed them in Table 3-4.

Group type	Group id	$trust_i$	Percentage	Count
Family with young children	FY	70%	6.8%	7686
Family with older children	FO	70%	8.7%	9945
Group of friends	GF	40%	18.5%	21056
Young couple	YC	55%	21.3%	24276
Solo traveler	ST	50%	22.2%	25276
Mature couple	MC	60%	22.5%	25599
Total	-	-	100%	113838

Table 3-4 Trust value based on different group types and their percentage

From Table 3-4 we can see that users whose group type is family with children or mature couple were assigned a relatively big trust value, while group of friends and solo traveler were allocated a relatively small value. This is on the basis of our common sense that family with children usually needs to think about the children's feelings and will probably give more objective and impartial ratings. Experienced older people like mature couple's judgment are more trustworthy than young people as well.

- **Trust-3**: $trust_i$ can be evaluated by the number of items that user i has voted $|\boldsymbol{I}_i|$. We assume that users who have more experience should be more trustworthy. Hence we simple define $trust_i$ as $|\boldsymbol{I}_i|$, as indicated in Equation (7):

$$trust_i = |\boldsymbol{I}_i| \tag{7}$$

The next step of out trust-enhanced method is simply to adjust weight computation measure by replacing $w(a,i)$ with $trust_i$ in Equation (1). Thus we have a new Equation (8) which entails the effect of trust:

$$p_{a,j} = \bar{r}_a + \frac{\sum_{i=1}^{n} trust_i (r_{i,j} - \bar{r}_i)}{\sum_{i=1}^{n} trust_i} \tag{8}$$

The intuition behind this computation is that users who have higher trust value will account for a greater proportion when giving suggestions to the active user a. In the former situation, $w(a,i) = 1$ is a constant so that each neighbor account for the same proportion when giving suggestions to the active user a. Therefore we assume that this adjustment on weight computation will make the predictions more accurate.

3.3 Summary

In this section firstly we introduced the conventional collaborative filtering method, in which we apply the k-nearest neighbor to search similar users combined with k-means cluster, instead of using the commonly used PCC or cosine distance. In section 3.2, for the purpose of making predictions more accurate, three different methods of trust inference were measured and incorporated into the weight computation step. In the next section, we are going to introduce the hotel datasets, tools and sparsity problem lies in our dataset. Evaluation metrics and algorithms will be formulated as well and we compare the effect of incorporating different measurements of trust into the CF algorithm by doing experiments separately with conventional method (CM), Trust-1, Trust-2 and Trust-3. Finally an empirical analysis will be given on the strength of experimental results as well.

4 Experiment

4.1 Datasets and Tools

The purpose of this section is to implement and compare the conventional method with trust-enhanced method presented in section 3, based on our hotel datasets. Our hotel datasets were generated from *Booking.com*, which is an informative website that provides travelers with accessible accommodation. It was based in Amsterdam in the Netherlands and supported internationally by offices in over 50 countries around the world. Established in 1996, it offers over 280,000 hotels and apartments in 180 countries and attracts over 30 million unique visitors each month at present. More than 18 million unbiased reviews from real guests can be found from *Booking.com* up to now.

We randomly selected a number of rating records from hotels located in 8 cities in Europe from May, 2011 till July, 2012 to form the original dataset. These cities are Barcelona, Berlin, London, Paris, Prague, Rome, Stockholm and Zürich. After fundamental data analysis and processing, unimportant columns and duplicated rows were removed from original dataset, totally 123,296 ratings were obtained from 113,838 users and 5,291 hotels, ranging from 2.5 to 10.0. This leads to the sparsity problem that we have mentioned previously, because most users' experience were not enough to generate a reasonably distributed dataset: 95% of all guests reviewed only one hotel, as indicated in Table 4-1.

Number of hotels one user has rated	1	2	3	4	5	6	7	8	9	10	…
Number of users	107922 (95%)	4558 (4%)	778 (0.7%)	248	110	52	42	24	17	9	…

Table 4-1: Distribution of number of hotels that one user has rated

$$\begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{array} \begin{array}{c} \begin{array}{ccccc} i_1 & i_2 & i_3 & i_4 & i_5 \end{array} \\ \begin{bmatrix} 0 & 7.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6.2 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 9.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8.0 & 0 \end{bmatrix} \end{array}$$

Figure 4-1: Example of sparse user-item rating matrix

To illustrate the sparsity problem more specifically, we use an example of rating matrix to represent our data. In Figure 4-1, rows of the matrix denote users and columns denote items. Therefore we have 5 users and 5 items as well as the ratings from users to related items. Here '0' denotes null value, which means the user has not used this item yet. Obviously there are much more '0' than non-zero values in a sparse rating matrix. In practical situations, this problem will be much more serious due to huge number of users and items. This weakness lies in data has disadvantages from two aspects when CF algorithm is applied to make recommendations. Firstly, sparsity makes it difficult to find co-rated items for two users, as indicated in Figure 4-1, only u_2 and u_5 have co-rated item i_4, thus making it difficult to use the effective similarity computation method such as Pearson Correlation Coefficient (PCC). Secondly, huge number of useless zeros will occupy a lot of memory and decrease the operating speed of computers. To this end, we

develop a method to store the data instead of using rating matrix and memory can be saved as well.

For the convenience of processing data and programming, the original dataset was divided into three co-related sets: Rating set, User set and Hotel set. Table 4-2, 4-3 and 4-4 illustrates detailed description of these datasets.

	User id	Hotel id	Rating score
1	1	3421	7.5
…	…	…	…
123296	113838	4161	9.6

Table 4-2: Rating set

	User id	User name	User location	Group id	Group name
1	1	×××	×××	GP	Group of friends
…	…	…	…	…	…
113838	113838	×××	×××	ST	Solo traveler

Table 4-3: User set

Each user belongs to a specific type of group. These groups are: family with young children (FY), family with older children (FO), group of friends (GF), mature couple (MC), solo traveler (ST) and young couple (YC). This attribute can be used to assign trust value for each user as described in section 3.2.

	Hotel id	Hotel name	City	Total number	Star level	Total score	Clean score	××× score	Value for money score	Cluster id
1	1	×××	×××	29	3	8.1	7.8	…	8.2	11
…	…	…	…	…	…	…	…	…	…	…
5291	5291	×××	×××	411	5	8.1	8.7	…	7.1	12

Table 4-4: Hotel set

In hotel dataset, there are 9 numerical attributes corresponding with each hotel, i.e., total number, star level, total score, clean score, comfort score, location score, service score, staff score and value for money score. Total score can be viewed as the most accurate evaluation of a hotel by reason of it is based on rating scores from all guests who have reviewed it as described in Trust-1 in section 3, and total number explained how many users have contributed on this total score. Star level is commonly employed to categorize hotels, usually on a scale from 1 to 5 stars. In our dataset, 0 star means no information about star level for a hotel. Cluster id is on a scale from 1 to K, which were generated based on the 9 numerical attributes of hotels.

We use JMP® Pro 10.0.0 to explore and sort data into desired form, as well as generate cluster id for hotels with k-means clustering. After that we implement algorithms (section 4.3) with Microsoft Visual C++ 2010 Express.

4.2 Evaluation Metrics

For the purpose of evaluating and comparing the performance of each algorithm, we follow the most commonly used approach of hiding a certain percentage of rating scores from the dataset then applying each algorithm in turn to predict the value of hidden ratings. In this case the accuracy of each algorithm can be evaluated from the difference between actual rating value and predicted value. We apply two metrics which are

commonly used in the field of recommender systems to evaluate each algorithm: Mean Absolute Error (MAE) and Coverage [13, 14, 17, 19].

- Mean Absolute Error (MAE)

  Mean Absolute Error (MAE) is the average absolute deviation of the actual rating values to the predicted values as can be shown in Equation (6), where $p_{i,j}$ is the predicted rating value that user i give to item j, $r_{i,j}$ is the actual rating value and P is the number of hidden ratings which are able to be predicted by the algorithm. Thus the lower the MAE, the more accurately the recommender algorithm predicts user ratings because the predicted values do not vary very far from true ratings.

$$MAE = \frac{\sum_{i=1}^{P} |p_{i,j} - r_{i,j}|}{P} \tag{6}$$

- Coverage

  Another important metric to estimate recommender engines is coverage for rating predictions, which is simply the percentage of hidden ratings that an algorithm was able to predict. As shown in Equation (7), P refers to the number of predictions made by the algorithm and N is the total number of hidden ratings, which is the requested number of predictions. As a result, higher coverage values indicate an algorithm can provide a relatively complete prediction for users.

$$Coverage = \frac{P}{N} \tag{7}$$

4.3 Algorithm

In this section a detailed algorithm was given to implement the CF method and trust – enhanced techniques described in methodology part. Mainly it aims at finding neighbors for hidden users and making prediction for them. Evaluation of predicted results was also included in this algorithm. Implications of symbols that used in this algorithm can be found in Appendix-A (section 7.1).

When executing the computation steps in this algorithm, we vary different parameters that affect the selection of neighbors and prediction results. Thus we run this algorithm for 36 times and each time we get different MAEs and Coverage (see Appendix-B). We will analyze these results in detail in the next section and show the effect of incorporation of trust, as well as the effect of changing parameters.

Set $\boldsymbol{R'} = \emptyset, \boldsymbol{R'_t} = \emptyset, \boldsymbol{U'_t} = \emptyset, \boldsymbol{N}_i = \emptyset, \boldsymbol{P}_i = \emptyset$. For each $r_{i,j}$ in $\boldsymbol{R}$, if $\lvert \boldsymbol{I}_i \rvert > 1$, add $r_{i,j}$ into $\boldsymbol{R'}$. Randomly hide (1- α) percentage ratings from $\boldsymbol{R'}$ as test data, add them into $\boldsymbol{R'_t}$. For each $r_{i,j}$ in $\boldsymbol{R'_t}$, add user *i* into $\boldsymbol{U'_t}$. For each user *i* in $\boldsymbol{U'_t}$, set user *i* as active user *a*: For each user *i* in $\boldsymbol{U}$ - *a*, if condition (4) is satisfied, set $w(a,i) = 1$ *or* $trust_i$ and add user *i* into $\boldsymbol{N}_a$. For each user *i* in $\boldsymbol{N}_a$, add $\boldsymbol{I}_i$ into $\boldsymbol{P}_a$. Calculate $p_{a,j}$ for each item *j* in $\boldsymbol{P}_a$ using formula (1). Calculate Coverage and MAE based on $r_{i,j}$ in $\boldsymbol{R'_t}$ and $\boldsymbol{P}_i$ for each user *i*.

Table 4-5: Implementation steps of CF algorithm combined with trust

4.4 Experimental Results

In this section we present our experimental results of applying both conventional method and trust-enhanced method to implement the predictions and evaluate the results by using MAE and Coverage. First four different weight computation measures were compared using MAE to illustrate their prediction accuracy (4.4.1). Sensitivity of the effects of varying different parameters on experimental results was analyzed in this part as well. These parameters include: training ratio of experimental ratings - α (4.4.2), threshold parameters to filter neighbors for experimental users - θ_2, θ_3, θ_4 (4.4.3) and total number of clusters for items - K (4.4.4). For the convenience of comparing experimental results, we set $\theta_1 = 5$ in each condition and do not change this parameter. Finally a summary was given in 4.4.5 based on the analysis from different angles in this section.

4.4.1 Performance of different prediction methods

We implement the collaborative filtering algorithm based on four different weight computation measures as described in methodology part. For each prediction method, we use training set to implement algorithm and predict rating values for hidden ratings in testing set. Parameters α, θ_2, θ_3, θ_4, K were varied and a set of MAE values were generated for each method. Then we compute the average value of MAE for each method.

Figure 4-2 shows the experimental result. It can be concluded from this figure that although the difference of performance between each measure is not very significant, the use of trust still affects the result. Trust-1 and Trust-2 proves to generate more accurate predictions than conventional method on average with our dataset. However, Trust-3 does not show advantage in improving prediction accuracy. This result indicates that if we inference trust value based on the difference between the actual rating value and total rating value (Trust-1) or based on group type of users (Trust-2), the prediction might be more accurate than using conventional weight computation method, although in some cases the conventional method proves to be the best one (see Table 7-1 in Appendix).

The average value of coverage (0.112212) here is the same for each method by reason of the weight computation measure does not influence the prediction coverage. Only θ_2, θ_3, θ_4 and K affect the coverage value.

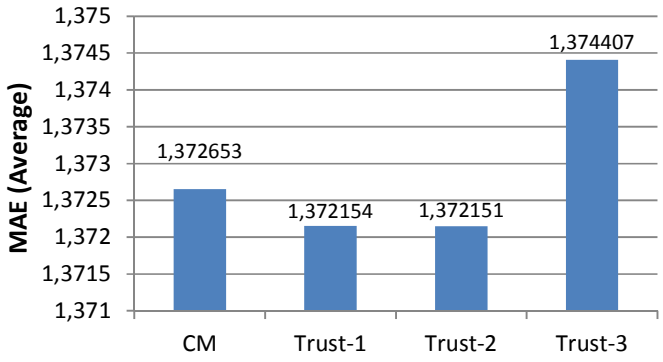

Figure 4-2: Impact of the weight computation measure on collaborative filtering algorithm

4.4.2 Sensitivity of training ratio α

The first parameter we varied in our experiment is training ratio of experimental ratings α. Totally 15,229 ratings were selected as experimental data from 123,296 ratings. After that we set training ratio α as 0.9, 0.7 and 0.5 respectively, with a result of 1,363, 3,798 and 5,882 ratings hidden from experimental data. Then we find neighbors for the hidden users

corresponding to the hidden ratings and make predictions for them using different methods. A set of MAE and Coverage values were generated under each condition.

Figure 4-3 illustrates the average value of MAE and Coverage when α equals to different values. It can be observed from this figure that when $\alpha = 0.5$, the prediction result is more accurate on average but the prediction coverage is comparatively lower than in other conditions, i.e., a smaller proportion of hidden ratings can be predicted when α is lower. When α is set to 0.9, the Coverage is higher but this is at the cost of prediction accuracy.

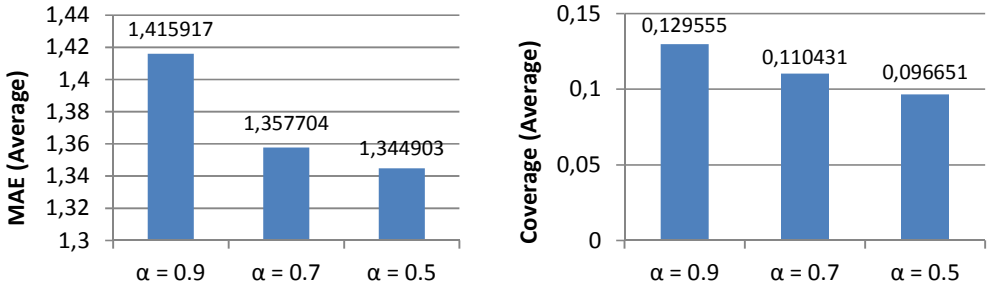

Figure 4-3: Sensitivity of training ratio α on MAE and Coverage

4.4.3 Sensitivity of threshold parameters $\theta_2, \theta_3, \theta_4$

The second parameters we varied in our experiments are threshold parameters that help to filter neighbors for testing users. As described in section 3, θ_2 represents the degree of similarity of two users when selecting items to use. Hence if θ_2 was set to a larger value, less neighbors can be found for testing users because it requires that two users should share a relatively big proportion of common clusters in items, i.e., they have strongly similar taste in choosing items to use. θ_3 and θ_4 are threshold parameters help to measure similarity of two users when giving ratings to items they have used. θ_3 is threshold for the difference of mean rating values between two users and θ_4 is threshold for the difference of standard deviations of rating values. Hence if θ_3 and θ_4 were set to a smaller value, more users will be filtered out because it requires higher level of similarity in rating items.

We can see from Figure 4-4 that the average MAE and Coverage are also affected by this parameter. When we set (θ_2, θ_3, θ_4) as (0.4, 0.6, 0.6), more neighbors can be found for testing users therefore the Coverage is higher but the prediction accuracy is lower. On the contrary, if we set (θ_2, θ_3, θ_4) as (0.7, 0.3, 0.3), the prediction result is more accurate at the cost of high prediction coverage.

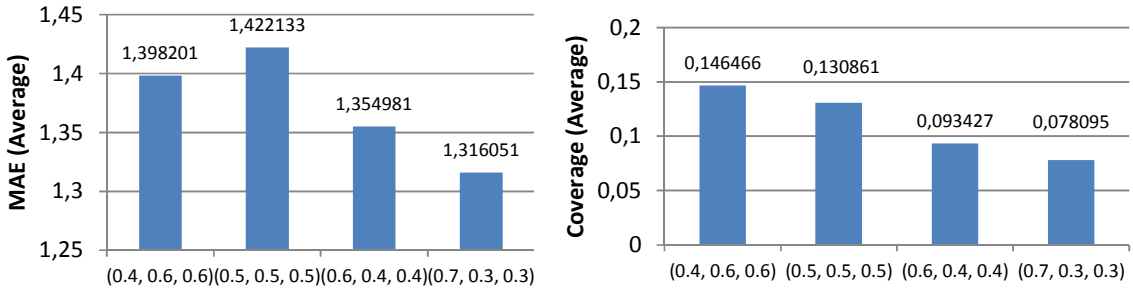

Figure 4-4: Sensitivity of threshold parameters $\theta_2, \theta_3, \theta_4$ on MAE and Coverage

4.4.4 Sensitivity of number of clusters *K*

The third parameter we varied in our experiments is the total number of clusters for items *K*. Obviously *K* also influences the prediction accuracy and coverage as can be seen from Figure 4-5. The Coverage becomes lower as *K* is set to be higher. This might be due to the reason that items were partitioned into smaller clusters thus making it difficult to find similar users who share the same proportion of clusters for items. However, there is no obvious pattern about how *K* affects the mean absolute error. When $K = 30$, MAE proves to be the least one but the Coverage is too low to generate reasonable predictions. When $K = 10$, though MAE = 1.37 is a little higher than MAE = 1.34 when $K = 30$, the Coverage = 0.17 is much better than that in any other conditions. Thus it can be concluded that it is better to set *K* as a relatively small number, e.g., $K = 10$.

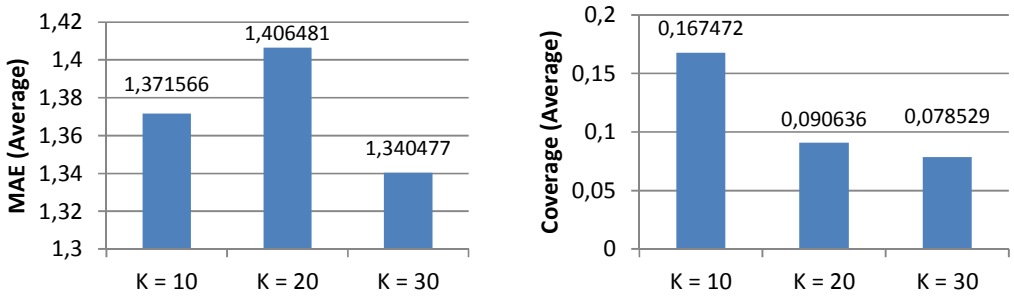

Figure 4-5: Sensitivity of total number of clusters for items *K* on MAE and Coverage

4.4.5 Summary

In this section an empirical analysis on experimental results was given to evaluate the prediction accuracy and coverage with different weight computation methods and different parameters. It can be concluded from the discussion above that the use of trust is able to influence the prediction accuracy as described in other research work, though the improvement is not very significant here. Trust-2 proves to perform better than the other weight computation methods, which means the weight computation based on group type of users is better than the other weight computation methods. A series of parameters also act as important roles when using the same algorithm to do predictions for testing users. In general, rules from our analysis indicate that there exists a trade-off between the prediction accuracy and coverage, i.e., prediction accuracy is at the cost of prediction coverage and vice versa. However, when we set number of clusters *K* as 10, the Coverage (0.167472) is much higher than in any other conditions and the MAE is also better than the average level in Trust-2 ($1.371566 < 1.372151$).

5 Conclusion

5.1 What have we done?

In this paper, a computational model for trust-based collaborative filtering was proposed and implemented with our hotel data sets collected from *Booking.com.* Our goal is to explore an effective method which is able to make rating value predictions for users even when there exists sparsity problem on data sets and test the effect of trust, in line with the conventional CF algorithm.

On the basis of data analysis, we choose k-nearest neighbor as our method to find neighbors with similar preference and taste for testing users, instead of applying the commonly used Pearson correlation coefficient or Cosine distance method, for the reason that it is difficult to find users with co-rated hotels. Trust has been applied into the conventional CF algorithm by adjusting the weight between testing users and their neighbors, according to the user information of neighbors like group type or how much experience they have. The empirical analysis on experimental results has shown that the use of trust can improve prediction accuracy only when the definition of trust is reasonable enough, e.g., Trust-1 and Trust-2. Therefore the measurement and inference of trust is a critical step when incorporating trust into recommender systems. Parameters like number of item clusters K also plays an important role when making predictions for users, e.g., $K = 10$ performs better when considering both MAE and Coverage compared with $K = 20$ and $K = 30$.

5.2 Contributions

Our contributions in this paper are mainly focused on the following aspects. Firstly, we collected large amount of data on hotel ratings and user information from a well-known accommodation booking website *Booking.com*, then we analyzed and sorted data from different angles using JMP software. Secondly, instead of using the well-known Pearson correlation coefficient method to compute weight between users, we apply k-nearest neighbor to filtering neighbors which can be easily handled with a sparse rating dataset. Thirdly, we make three different assumptions of trust and substitute the neighbor weight with them in CF formula, and tested it with our hotel data sets to verify the feasibility of trust. Fourthly, we avoid using the conventional rating matrix to store data due to large amount of null values, instead we use some data structures like User, Item and Rating which helps to save memory when programming with C++.

One critical aspect of contributions in this paper is that we use k-means clustering to assign each hotel a cluster id. This step makes it easier for us to find the relevance of two users on their preference when choosing and rating items, on the basis of the common clusters of items that two users has voted, and their rating pattern on these clusters. This is a direction that maybe we can explore and improve to alleviate the sparsity problem in the future.

5.3 Future Work

Future work of our thesis work will be centralized in several areas: Can this model be applied to other data sets? Is there a more effective way to infer trust that can improve the prediction accuracy significantly? What will happen if we apply trust propagation into our model? What should be noticed here is the complexity of trust, which was denoted by Elizabeth Chang [16], that trust is composed by three fuzzy and dynamic characteristics: Implicitness, Asymmetry and Transitivity in trust. However, in our case, asymmetry and

transitivity had not been taken into consideration thus this is something that we can improve in future work.

There are several different ways to answer the first question: Can this model be applied to other data sets? Due to time limitation, we were not able to generate a relatively reasonable distributed dataset, i.e., too many users and hotels generate few ratings. Thus if we are able to collect or get a more reasonably distributed dataset from other websites, we may test on it and compare with the results from the dataset we used in this paper. Some datasets such as *Epinions.com* provide trust value matrix by itself thus we don't need to infer trust under this case.

For the method of trust inference, the assumptions that we made in this paper were generated based from information in our case thus it has limitations when applying to other cases. Therefore we may need to develop a more general method of trust inference which can be widely applied. Another inspiration from our experimental results is that maybe we can combine Trust-1 and Trust-2 to improve the prediction accuracy because both of them act as positive factor when affecting the experimental results.

Trust propagation is actually the exemplification of the Transitivity characteristic of trust. A good example of trust propagation can be found in [17], in which trust was propagated according to a weighted sum of plus or minus sign and numerical values of users' similarity. However, it didn't take the Asymmetry characteristic of trust into consideration. Therefore a more improved way of incorporating trust propagation should consider both Transitivity and Asymmetry of trust.

6 References

[1] Ricci, Francesco, Lior Rokach, and Bracha Shapira. *Recommender Systems Handbook* (2011).

[2] Golbeck, Jennifer, and James Hendler. "Filmtrust: Movie recommendations using trust in web-based social networks." *Proceedings of the IEEE Consumer communications and networking conference*. Vol. 96. University of Maryland, 2006.

[3] Balabanović, Marko, and Yoav Shoham. "Fab: content-based, collaborative recommendation." *Communications of the ACM* 40.3 (1997): 66-72.

[4] Burke, Robin. "Hybrid recommender systems: Survey and experiments." *User modeling and user-adapted interaction* 12.4 (2002): 331-370.

[5] Levy, Mark, and Klaas Bosteels. "Music recommendation and the long tail." *1st Workshop On Music Recommendation And Discovery (WOMRAD), ACM RecSys, 2010, Barcelona, Spain*. 2010.

[6] Goldberg, David, et al. "Using collaborative filtering to weave an information tapestry." *Communications of the ACM* 35.12 (1992): 61-70.

[7] Resnick, Paul, and Hal R. Varian. "Recommender systems." *Communications of the ACM* 40.3 (1997): 56-58.

[8] Adomavicius, Gediminas, and Alexander Tuzhilin. "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions." *Knowledge and Data Engineering, IEEE Transactions on* 17.6 (2005): 734-749.

[9] Weinberger, Kilian Q., John Blitzer, and Lawrence K. Saul. "Distance metric learning for large margin nearest neighbor classification." *In NIPS*. 2006.

[10] Linden, Greg, Brent Smith, and Jeremy York. "Amazon.com recommendations: Item-to-item collaborative filtering." *Internet Computing, IEEE* 7.1 (2003): 76-80.

[11] Ge, Yong, et al. "An energy-efficient mobile recommender system."*Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010.

[12] Breese, John S., David Heckerman, and Carl Kadie. "Empirical analysis of predictive algorithms for collaborative filtering." *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence.* Morgan Kaufmann Publishers Inc., 1998.

[13] Sarwar, Badrul, et al. "Item-based collaborative filtering recommendation algorithms." *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001.

[14] O'Doherty, Daire, Salim Jouili, and Peter Van Roy. "Trust-based recommendation: an empirical analysis." *Submitted to: Proceedings of the Sixth ACM SIGKDD Workshop on Social Network Mining and Analysis SNA-KDD, Beijing, China, ACM*. 2012.

[15] Wang, Jing, et al. "Trust-based collaborative filtering." *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*. Vol. 4. IEEE, 2011.

[16] Chang, Elizabeth, Tharam S. Dillon, and Farookh K. Hussain. *Trust and reputation for service-oriented environments: Technologies for building business intelligence and consumer confidence*. John Wiley & Sons, P.110-114, 2006.

[17] Papagelis, Manos, Dimitris Plexousakis, and Themistoklis Kutsuras. "Alleviating the sparsity problem of collaborative filtering using trust inferences." *Trust management* (2005): 125-140.

[18] Resnick, Paul, et al. "GroupLens: an open architecture for collaborative filtering of netnews." *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM, 1994.

[19] Massa, Paolo, and Paolo Avesani. "Trust-aware collaborative filtering for recommender systems." *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE* (2004): 492-508.

[20] Huming, Gao, and Li Weili. "A Hotel Recommendation System Based on Collaborative Filtering and Rankboost Algorithm." *Multimedia and Information Technology (MMIT), 2010 Second International Conference on*. Vol. 1. IEEE, 2010.

[21] Huang, Zan, Hsinchun Chen, and Daniel Zeng. "Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering." *ACM Transactions on Information Systems (TOIS)* 22.1 (2004): 116-142.

[22] Su, Xiaoyuan, and Taghi M. Khoshgoftaar. "A survey of collaborative filtering techniques." *Advances in Artificial Intelligence* 2009 (2009): 4.

[23] Nooteboom, Bart. *Trust: Forms, foundations, functions, failures and figures*. Edward Elgar Pub, 2002.

[24] Sztompka, Piotr. Zaufanie: fundament społeczeństwa. Znak, 2007.

[25] Massa, Paolo, and Paolo Avesani. "Controversial users demand local trust metrics: An experimental study on epinions. com community." *Proceedings of the National Conference on artificial Intelligence*. Vol. 20. No. 1. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[26] http://en.wikipedia.org/wiki/Collaborative_filtering

[27] http://en.wikipedia.org/wiki/K-means_clustering

7 Appendix

7.1 Appendix-A: Notation and Terminology

Sets:

$\boldsymbol{U}$ = set of all users in dataset denoted by i, $i = \{1, \ldots, M\}$
$\boldsymbol{I}$ = set of all items in dataset denoted by j, $j = \{1, \ldots, N\}$
$\boldsymbol{R}$ = set of all ratings in dataset denoted by k, $k = \{1, \ldots, L\}$
$\boldsymbol{I_i}$ = set of items on which user i has rated
$\boldsymbol{C_i}$ = set of clusters to which $\boldsymbol{I_i}$ belongs
$\boldsymbol{N_i}$ = set of neighbors of user i
$\boldsymbol{P_i}$ = set of predicted items of user i
$\boldsymbol{R'}$ = set of experimental ratings from users whose $|\boldsymbol{I_i}| > 1$
$\boldsymbol{R'_t}$ = set of hidden ratings with training ratio α from $\boldsymbol{R'}$
$\boldsymbol{U'_t}$ = set of users whose ratings belong to $\boldsymbol{R'_t}$

Variables:

t_j = total rating value of item j
$r_{i,j}$ = rating value of item j by user i
$p_{i,j}$ = predicted rating value of item j by user i
$\bar{t}_i$ = mean total rating value for user i
$\bar{r}_i$ = mean rating value for user i
σ_i = standard deviation of rating values for user i
$w(a, i)$ = weight between active user a and user i
P = number of predictions made by one algorithm
N = number of ratings expected to be predicted by one algorithm

Parameters:

K = total number of clusters for items set $\boldsymbol{I}$
α = training ratio of experimental ratings set $\boldsymbol{R'}$
$\theta_1, \theta_2, \theta_3, \theta_4$ = threshold parameters to filter neighbors for user i in condition (4)

Table 7-1: Notation and Terminology in this article

7.2 Appendix-B: Experimental Results

Training ratio α	Threshold parameters $(\theta_2, \theta_3, \theta_4)$	Number of cluster K	MAE_{CM}	$MAE_{Trust-1}$	$MAE_{Trust-2}$	$MAE_{Trust-3}$	Coverage
0.9	(0.4, 0.6, 0.6)	10	1.35885	1.3606	1.35972	1.35919	0.23991
		20	1.50017	1.49752	1.49961	1.49702	0.146
		30	1.47728	1.47516	1.47709	1.48479	0.13353
	(0.5, 0.5, 0.5)	10	1.37709	1.37674	1.37782	1.37999	0.21717
		20	1.53767	1.53592	1.53734	1.53745	0.12546
		30	1.52705	1.52434	1.52665	1.52432	0.11519
	(0.6, 0.4, 0.4)	10	1.33374	1.33483	1.33469	1.33434	0.15334
		20	1.47678	1.47657	1.47651	1.476	0.08584
		30	1.42795	1.42739	1.4285	1.42783	0.0785
	(0.7, 0.3, 0.3)	10	1.28044	1.28159	1.28113	1.28131	0.12913
		20	1.37302	1.37211	1.37277	1.37163	0.06897
		30	1.32126	1.32096	1.32084	1.32246	0.06163
0.7	(0.4, 0.6, 0.6)	10	1.37691	1.37451	1.37537	1.38091	0.21037
		20	1.34935	1.34606	1.34654	1.35982	0.11532
		30	1.37676	1.37731	1.37537	1.38336	0.10532
	(0.5, 0.5, 0.5)	10	1.38256	1.37971	1.38101	1.38703	0.19484
		20	1.38783	1.38338	1.38503	1.39552	0.10321
		30	1.41474	1.41508	1.41332	1.41958	0.09189
	(0.6, 0.4, 0.4)	10	1.36404	1.36296	1.3629	1.36921	0.14797
		20	1.3419	1.33821	1.33961	1.34733	0.06925
		30	1.29522	1.29565	1.29384	1.29796	0.05977
	(0.7, 0.3, 0.3)	10	1.3495	1.3496	1.34949	1.35044	0.12428
		20	1.34179	1.3423	1.34154	1.3432	0.05635
		30	1.30595	1.30702	1.30572	1.30736	0.0466
0.5	(0.4, 0.6, 0.6)	10	1.42088	1.42178	1.42094	1.4206	0.17919
		20	1.42398	1.42334	1.42322	1.42439	0.10337
		30	1.29656	1.29741	1.29588	1.29697	0.08518
	(0.5, 0.5, 0.5)	10	1.44423	1.44491	1.44411	1.44475	0.16627
		20	1.44042	1.44061	1.44018	1.44088	0.09198
		30	1.28709	1.288	1.28689	1.28756	0.07174
	(0.6, 0.4, 0.4)	10	1.40063	1.40079	1.40049	1.40101	0.13176
		20	1.37934	1.37978	1.37932	1.37921	0.06528
		30	1.17372	1.17402	1.17354	1.1735	0.04913
	(0.7, 0.3, 0.3)	10	1.36532	1.36552	1.3652	1.36579	0.11544
		20	1.32714	1.32737	1.327	1.32742	0.05661
		30	1.17834	1.17851	1.17825	1.17853	0.04386
Average value			1.372653	1.372154	1.372151	1.374407	0.112212

Table 7-2: Complete experimental results on MAE and Coverage when choosing different parameters