Anadolu Üniversitesi Bilim ve Teknoloji Dergisi A- Uygulamalı Bilimler ve Mühendislik
Anadolu University Journal of Science and Technology A- Applied Sciences and Engineering

Year: 2018
Volume: 19
Number: 2
Page: 303 - 315
DOI: 10.18038/aubtda.346407

DYNAMIC *k* NEIGHBOR SELECTION FOR COLLABORATIVE FILTERING

Halil ZEYBEK [1], Cihan KALELİ [1,*]

[1] Department of Computer Engineering, Faculty of Engineering, Anadolu University, Eskişehir, Turkey

ABSTRACT

Collaborative filtering is a commonly used method to reduce information overload. It is widely used in recommendation systems due to its simplicity. In traditional collaborative filtering, recommendations are produced based on similarities among users/items. In this approach, the most correlated *k* neighbors are determined, and a prediction is computed for each user/item by utilizing this neighborhood. During recommendation process, a predefined *k* value as a number of neighbors is usedfor prediction processes. In this paper, we analyze the effect of selecting different *k* values for each user or item. For this purpose, we generate a model that determines *k* values for each user or item at the off-line time. Empirical outcomes on movie based dataset show that using the dynamic *k* values during the *k*-nn algorithm leads to more favorable recommendations compared to a constant *k* value.

Keywords: *k*-nearest-neighbor, Collaborative filtering, Dynamic *k*, Accuracy

1. INTRODUCTION

We live in a digital age where data originate from many different sources. A large percentage of data arises from the Internet usage [1]. In recent years, people all over the world use the Internet to supply most of their needs such as shopping, reading newspapers, banking and planning their holidays. While people use the Internet, they leave enormous information behind them. Recommendation systems, which are simply the software tools and techniques, utilize this information to provide suggestions to the users [2]. Collaborative Filtering (CF) is a widely used recommendation method to produce predictions to the users based on their preferences [3]. The world's leading online service providers, e.g., Amazon, Spotify, TripAdvisor, etc. use the CF to meet the customer satisfaction and to increase the trading size by exploring the relevant products based on the history of user preferences [4].

Memory-based and model-based methods are two of the most common CF methods used in the literature. While neighborhood-based and heuristic-based methods are examples of memory-based CF methods, Bayesian Clustering, Singular Value Decomposition (SVD), and Support Vector Machines (SVM) are the implementations of the model-based CF methods [8, 11, 12, 13]. In the memory-based CF methods, a two-dimensional rating matrix, which has the users as rows and the items as columns, is persisted in the system and used to compute the prediction depending on the past habits. User-based and item-based methods both use the rating matrix and are instances of neighborhood-based CF methods. In user-based CF algorithm, neighbors of user *u* who have similar rating pattern with the user *u* are specified, and a prediction is computed according to the preferences of these users. Similarly, in item-based CF method, neighbors of a target item are specified, and then a prediction is computed. On the other hand, a predictive model is constructed from the rating matrix, and then the predictions are computed via this model in a type of CF method.

*Corresponding Author: ckaleli@anadolu.edu.tr
Receiving Date: 25 October 2017 Publishing Date: 29 June 2018

One of the most popular approaches to memory-based CF recommendation is the *k*-nearest-neighbor (*k*-NN) algorithm [2]. The *k*-NN algorithm depends on the similarities among users or items. CF systems persist the members' item ratings in a 2-dimensional user-item matrix [7, 8]. While predicting a rating value in a user-item matrix, top *k* closest neighbors are taken into account [5]. For instance, while estimating prediction value of a target movie for a user, firstly correlations between the user and all other users are calculated, and then the *k* closest neighbors are selected, and finally, a prediction value is estimated [6].

The most crucial step in the *k*-NN algorithms is determining an appropriate *k* value. If *k* is chosen to be a small number, the algorithm will be sensitive to irrelevant cases. On the other hand, if *k* is chosen to be a large number, estimations may be less consistent due to relatively distant neighbors [2]. Thus, the *k* value of the *k*-NN algorithm varies with the context (project, subject,etc.).

The purpose of a recommendation system is to provide the best recommendations to users. CF and *k*-NN algorithms also serve this purpose. However, it is considered that working with the same *k* for all users or items, may conflict with this purpose. Therefore, it has been considered that accuracy of estimations may increase using different *k* values for each user/item in user-based/item-based *k*-nn. In this paper, we motivate on showing the effect of dynamic *k* values during recommendation process. Therefore, we performed a set of experiments and results investigate that accuracy of a traditional user, or item-based CF systems' accuracy can be improved by utilizing dynamic *k* values.

2. RELATED WORK

Since interest in the *k*-NN-based CF increased in the nineties, researchers tried to effectively use nearest-neighbor algorithms in recommendation processes [14]. Herlocker et al. [15] had a comprehensive study on the neighbor selection for the first time in CF. They concluded that Pearson correlation coefficient is an effective way to compute the similarities between users and employing the most similar *k* users as neighbors increase the quality of the prediction. In the following years, researchers tried to improve the prediction quality by improving the neighbor selection. Sarwar et al. [19] had a similar study, but they worked on similarities between items instead of the users. They used threshold-based neighbor selection method, which was employed by Kim and Yang [16], to calculate the similarities between the items. Thanks to this threshold-based neighborhood selection method, neighbors are substituted for a user with an unusual rating pattern. Liang et al. [17] proposed calculating the similarities between users from a subset of the items, not from the whole dataset. Koren [18] proposed to specify the nearest neighbors of a user by optimizing a global cost function that improves the accuracy of estimations.

These studies aim to increase the accuracy of the predictions. To achieve this goal, some of them use different neighborhood selection methods while some of them vary in the similarity calculation algorithms. Although these studies differ from each other in similarity calculation or neighborhood selection method, ultimately they use the *k*-nearest-neighbor of the customers to compute the prediction. The *k* value of the algorithm is determined, and then the same *k* value is utilized for all users or items. In this study, the objective of the proposed method is to increase the accuracy of the predictions by using dynamic *k* values for each user or item in the system.

3. NEAREST NEIGHBOR-BASED CF ALGORITHMS

Nearest neighbor-based algorithms are based on the fact that similar users have similar patterns of rating behavior and users give similar ratings to similar items [20]. Recommender systems usually work on a user-item matrix that is composed by collected user preferences. This two-dimensional matrix consists of *m* users and *n* items which represent matrix's rows and columns. User for whom a prediction value will be estimated for a particular item is called active user (*a*). To produce a prediction for *a* for target item *q*, either user-based CF or item-based CF algorithms are used.

3.1. User-Based *k*-Nearest Neighbor Algorithm

User-based CF's first step is calculating correlations between a and other users in the system by utilizing correlation metrics such as Pearson Correlation Coefficient (PCC). After similarities between users are computed, recommender system selects top k users who are the most similar to a. Finally, a prediction value is estimated for a for q by taking into account only k users selected before. This process is repeated for each item that a prediction value will be estimated for. User-based k-nn algorithm's basic scheme is shown in Figure 1.

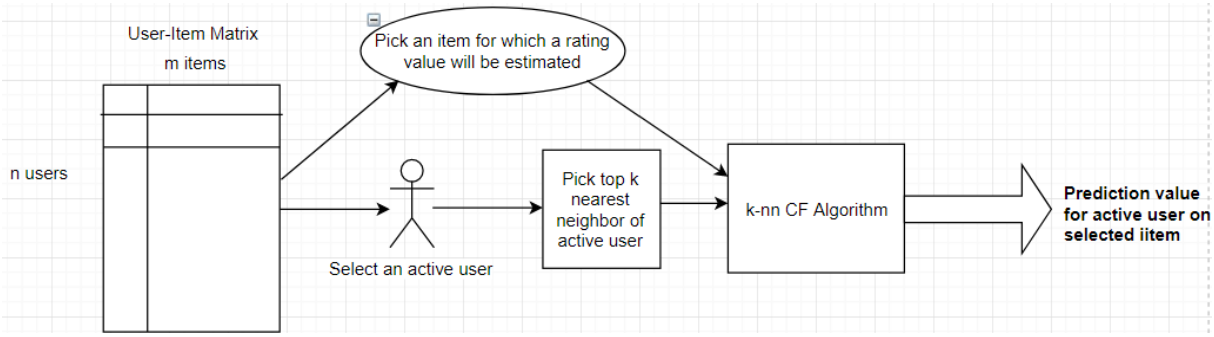


Figure 1. Basic scheme of *k*-nn algorithm

3.1.1. Similarity calculation

CF is based on calculating similarities among users or items. In order to calculate similarities between two users, different methods can be employed. One of these methods is PCC which is widely used in recommender systems. The similarities between a and one of user in the rating matrix can be computed via PCC as in the following equation.

$$w(a,u) = \frac{n * (\sum_{i \in I(a,u)}(r_{a,i} * r_{u,i})) - (\sum_{i \in I(a,u)}(r_{a,i})) * (\sum_{i \in I(a,u)}(r_{u,i}))}{\sqrt{[n * \sum_{i \in I(a,u)}({r_{a,i}}^2) - (\sum_{i \in I(a,u)}(r_{a,i}))^2] * [n * \sum_{i \in I(a,u)}({r_{u,i}}^2) - (\sum_{i \in I(a,u)}(r_{u,i}))^2]}} \quad (1)$$

Here, I represent set of items which are rated both a and user u, $r(a,i)$and $r(u,i)$ stands for ratings for item i that is an element of I rated by user a and user u, respectively. n is the number of co-rated items between a and user u.

The similarity coefficient value calculated by Eq.1 must be in the range -1 and +1. The greater value of similarity coefficient ($w(a,u)$) means the higher correlation between user a and u and vice versa.

3.1.2. Producing prediction

In user-based CF, in order to produce a prediction for target item q for a, firstly neighbors of user a are determined according to the weights between user a and all other users in the system and then aggregate ratings of users in the neighborhood as in Eq.2.

$$\boldsymbol{p_{a,q} = \overline{r_a} + \frac{\sum_{u \in U}(r_{u,q} - \overline{r_u}) * w(a,u)}{\sum_{u \in U} w(a,u)}} \quad (2)$$

Here, U represents set of users are neighbors of a, $\overline{r_a}$ is the mean of rating values of user a, and $w(a,u)$ corresponds to the similarity coefficient between user a and u calculated via Eq.1. While

producing prediction, it should be taken into account that prediction value must be in the range of minimum and maximum rating value of dataset is worked on.

3.2. Item-Based *k*-Nearest Neighbor Algorithm

Contrary to user-based CF, item-based CF's first step is calculating similarities between items, not users. After similarities between items are computed, recommender system determines top *k* items that are the most similar to the active item *q*. A prediction value is estimated for an *a* for a particular item by taking into account only *k* items which are most similar to this item.

3.2.1. Similarity calculation

Item-based CF has also some different weighting algorithms, but in this work PCC method as in user-based CF to provide a controlled experiment.

$$w(i,j) = \frac{n * (\sum_{u \in U(i,j)}(r_{u,j} * r_{u,i})) - (\sum_{u \in U(i,j)}(r_{u,j})) * (\sum_{u \in U(i,j)}(r_{u,i}))}{\sqrt{[n * \sum_{u \in U(i,j)}(r_{u,j}{}^2) - (\sum_{u \in U(i,j)} r_{u,j})^2] * [n * \sum_{u \in U(i,j)}(r_{u,i}{}^2) - (\sum_{u \in U(i,j)} r_{u,i})^2]}} \quad (3)$$

Here, *U* represents set of users which rated both active item *j* and item *i*, *r*(*u*,*j*)and *r(u,i)* stands for ratings for user *u* that is an element of *U* rates both item *j* and item *i*, respectively. n is the number of users who rated both *j* and *i*.
The similarity coefficient value calculated by Eq.3 must be in the range -1 and +1. The greater value of similarity coefficient ($w(i,j)$) means the higher correlation between item *j* and *i* and vice versa.

3.2.2. Producing prediction

In the item-based CF, to produce prediction estimation value for target item *i* for user *a*, the neighbors of item *i* are specified and then weighted average of neighbor items which are rated by the user *a* is calculated as in Eq.4.

$$\boldsymbol{p_{a,i}} = \frac{\sum_{j \in NN}(\boldsymbol{r_{a,j}} * \boldsymbol{w(i,j)})}{\sum_{j \in NN} \boldsymbol{w(i,j)}} \quad (4)$$

4. DETERMINING DYNAMIC *k* VALUE FOR *k*-NN BASED ALGORITHMS

The major problem in the *k*-nn algorithm is specifying the *k* value. In the traditional *k*-nn algorithm, after similarities are calculated between active user or item and all other users or items in the system top *k* neighbor are selected, and prediction is generated for a particular item. This procedure is repeated for all items which are to be estimated. In the *k*-nn algorithm, the different neighborhood size can affect the quality of the prediction indicated by MAE. For example, if 15 is set as *k* value lower MAE value is obtained compared to 20 is selected as *k* value, but if *k* is set about 30 the MAE value is less than MAE that is obtained when *k* equals to 15 [6].

In this work, we tried to specify *k* distinctly for all users or items in order to get higher accuracy. For user-based *k*-nn, we firstly assign user-specific *k* values in the training phase of our method and then use them for estimating prediction in the test phase. After we completed our work for a user-based *k*-nn algorithm, we adapted this work for an item-based *k*-nn algorithm by assigning specific *k* values for each item in the training phase and using them in the test phase. We also generated a prediction using the *k*-nn algorithm with a constant *k* for all users and items that is 50 to compare the results in terms of accuracy of the predictions.

After calculating weights between active user or item and all other users and items in the system separately, the weights are sorted in descending order, and the first *k* neighbor is selected to estimate prediction for a specific item. In the traditional *k*-nn algorithm the *k* value of the algorithm is set a predefined value. In other words, a *k* value is selected for the system, and all predictions for all users are estimated with the same *k* value.

In our study, we tried to give different *k* values for different users in order to give the best estimate. For this purpose, we generated estimation for all items that user rated by using leave-one-out cross-validation method. While generating estimations, we tested 10 different *k* values which spans from 5 to 50 with interval 5 for each user in the training phase of our algorithm. In other words, we produced 10 rating prediction for an item that is rated by *a* with 10 different *k* values. We performed it for all items that user rated. We also repeated this process for item-based CF.

Procedure 1 Rating Prediction Procedure in the Training Phase (user-based)
Require: User rating dataset for *n* users and *m* items ($\boldsymbol{U_{nxm}}$)
for all users **in***U*(*i*← 1 to *n*) **do**
  for all items **in***U*(*i*← 1 to *m*) **do**
    Calculate weights for each user between other users and specify nearest neighbors by sorting weights in descending order
    if $\boldsymbol{U_{n,m}}$is not nan **then**
      activeUserSimilarities← calculate $\boldsymbol{w(a,)}$ between *a* and all other users
      sortedUserSimilarities← sort(activeUserSimilarities)
      For all k values in range 5 to 50 estimate rating value by using *k*-nn algorithm
      for k **in***(5,10,15….,50)***do**
        $\boldsymbol{P_{k,n,m}}$←$\mathbf{p_{a,i}}$(calculate by using Equation 2)
      end for
    end if
  end for
end for

We hold our rating predictions in a three-dimensional matrix for 10 *k* values, 943 users, and 1,682 items. Rating predictions can be calculated as shown in Procedure 1. Rating estimation matrix has not a number (Nan) values for items which are not rated by user or whose rating prediction cannot be generated. We also adapted Procedure 1 for item-based CF by calculating similarities between items instead of users, by using Eq.4. instead of Eq.2.

Procedure 2 Rating Prediction Procedure in the Training Phase (item-based)
Require: User rating dataset for *n* users and *m* items ($\boldsymbol{U_{nxm}}$)
for all items **in***U*(*i*← 1 to *m*) **do**
  for all users **in***U*(*i*← 1 to *n***) do**
    Calculate weights for each item among other items and specify nearest neighbors by sorting weights in descending order
    if $\boldsymbol{U_{i,j}}$is not nan **then**
      activeItemSimilarities← calculate weights between active item and all other items
      sortedItemSimilarities← sort(activeItemSimilarities)
      For all k values in range 5 to 50 estimate rating value by using knn algorithm
      for k **in***(5,10,15….,50)***do**
        $\boldsymbol{P_{k,n,m}}$←$\mathbf{p_{a,i}}$(calculate by using Equation 4)
      end for
    end if
  end for
end for

5. EXPERIMENTS

5.1. Dataset & Preprocessing

In order to compare the effects of different *k* values on users, we used MovieLens dataset which has three columns; userId, movieId and rating value as the dataset. Since we worked with MATLAB that is a high-performance software written especially for scientific and numerical calculations and programming, we converted the original MovieLens dataset to MATLAB matrix. While doing the conversion process, we put Nan value which is an abbreviation for not-a-number if the dataset does not contain rating for some userId and movieId pair. In this way, we were able to use some special MATLAB functions such as *nanmean* function that calculates the user rating mean by omitting the Nan values. We also used *corr* function to calculate similarities between users and items by giving 'Pearson' as a parameter.

The dataset consists of 100.000 ratings from 943 users and 1682 movies. Each user has at least 20 ratings. This means only 6% of the available items are rated. The rating values in the dataset are in the range 1-5.

5.2 Metrics

Coverage and accuracy are vital dimensions which assess the quality of a prediction algorithm. Coverage specifies the percentage of items for which a recommendation system can provide prediction [6]. Neighborhood sizes, dataset quality, and size of the dataset can affect the coverage metric dramatically. We compute coverage by dividing the number of the items for which recommender system can produce prediction to a total number of items for which recommender system intended to produce prediction.

Accuracy of a system is divided into two categories as statistical accuracy metrics and decision-support accuracy metrics [6]. Statistical accuracy metrics evaluate accuracy of a recommender system by comparing the prediction values against the real user ratings. Mean Absolute Error (MAE) has been used previously to measure that kind of recommender systems' accuracy performance by Shardand & Maes[22] and Sarwar[21] et al.

5.3. Experimentation Methodology

We used leave-one-out cross-validation method, during specifying dynamic *k* values for each user phase. In this way, each user in the dataset is active user once, and rest of users populates the training data [9, 10]. For each element in the MATLAB matrix which has not a NAN value, a prediction is produced by excluding one rating at a time, and predicting the value of rating by using the similarity and prediction calculation formulas given in section 3.1 and 3.2. We also selected 5 items for each user among rated by that user, and produced the prediction for those items using *k* values which are user-specific.

5.4. Experimental Results and Discussion

For user-based CF, we produced estimations for each item that was rated by any user using Procedure-1 and we obtained a matrix having MAE values after subtracting our prediction values from the real ratings. For example, while first row and the first column of the matrix represent the MAE value of the first user for *k=50*, second row and the tenth column of the MAE matrix represent the MAE value of the second user for *k=5*.MAE values of the first ten users in the dataset for varying *k* values are shown in Table 1.

Table1. MAE values of the first ten users in the dataset for varying k values

	k=50	***k=45***	***k=40***	***k=35***	***K=30***	***k=25***	***k=20***	***k=15***	***k=10***	***k=5***
User1	1,0349	1,0702	1,1098	1,1336	1,2000	1,1997	1,6657	1,7890	1,5321	1,3688
User2	0,7597	0,7596	0,7928	0,8497	0,9027	0,9176	0,8055	0,8556	1,0106	0,9179
User3	1,0496	0,9801	0,9250	0,9572	0,9719	1,0189	1,1142	1,1079	1,1242	1,2876
User4	0,8617	0,8866	0,9144	0,8719	0,8275	0,7895	0,7860	0,7730	0,9155	1,0270
User5	1,0978	1,1285	1,1663	1,1143	1,1032	1,1282	1,2259	1,3125	1,3356	1,4249
User6	0,9388	0,9372	0,9734	0,9585	0,9033	0,8708	0,9046	0,8407	1,0558	1,1355
User7	1,0466	1,0409	1,0675	1,2055	1,2451	1,2666	1,0447	0,8382	0,8407	0,8189
User8	0,9467	0,9594	0,9576	1,0051	1,0101	1,0473	0,9184	0,9517	1,1437	1,1353
User9	0,9184	0,9292	0,8643	0,8991	0,8854	0,9415	0,9719	1,0195	1,1485	1,0996
User10	0,6708	0,5638	0,5564	0,6054	0,6448	0,6404	0,6073	0,6110	0,6489	0,7187

After we obtained MAE values for users for varying *k* values, we achieved to find the best *k* values for each user in the dataset. For example, according to the MAE values in Table 1 the best *k* values for each user are formed as shown in Table 2. Note that Herlocker et al. [6] set static *k* value to 50. Therefore, result of *k* equals 50 can used as baseline for comparison.

Table2. Dynamic *k* value look-up table for users

	The best *k* value for user	**MAE for specifying *k* value**
User1	50	1,0349
User2	45	0,7596
User3	40	0,9250
User4	15	0,7730
User5	50	1,0978
User6	15	0,8407
User7	5	0,8189
User8	20	0,9184
User9	40	0,8643
User10	40	0,5638

For item-based CF, we obtained the MAE matrix which has 1.682 rows which is the number of the movies in the user-item matrix and ten columns by subtracting our prediction values from the real ratings. MAE values of the first ten items in the dataset for varying *k* values are shown in Table 3.

Table 3. MAE values of the first ten items in the dataset for varying *k* values

	k=50	***k=45***	***k=40***	***k=35***	***k=30***	***k=25***	***k=20***	***k=15***	***k=10***	***k=5***
Item1	1,3478	1,2666	1,2069	1,2271	1,2415	1,2609	1,2975	1,2389	1,2976	1,4861
Item2	0,9906	0,9879	1,0108	0,9933	1,0368	0,9977	1,0054	0,9103	1,2130	1,3333
Item3	1,1624	1,0891	1,0894	1,0963	1,0771	1,0501	0,9876	0,9538	1,0000	1,0714
Item4	0,9863	1,0108	1,0992	1,0735	1,0643	1,1279	1,1646	1,1515	1,1914	1,2857
Item5	0,8732	0,8645	0,8214	0,7752	0,8744	0,8990	0,9108	0,8631	0,9420	1,1000
Item6	1,1008	1,1874	1,1099	1,1085	1,1137	1,0426	0,9461	1,0000	0,9100	0,6133
Item7	1,2427	1,3031	1,3184	1,3024	1,3497	1,2971	1,1925	1,2647	1,2800	1,7143
Item8	1,4337	1,4368	1,4114	1,4132	1,3337	1,3344	1,2773	1,3705	1,3824	1,2917
Item9	1,0134	0,9871	0,9712	0,9805	0,9801	0,9111	0,9477	1,1250	1,0810	1,0781
Item10	0,8465	0,8934	1,0103	1,0301	1,0611	1,0356	1,0900	1,1377	0,9537	1,0000

After we obtained the MAE values for items for varying *k* values, we achieved to find the best *k* values for each item in the dataset. The best *k* values are shown in Table 4 for the first ten items.

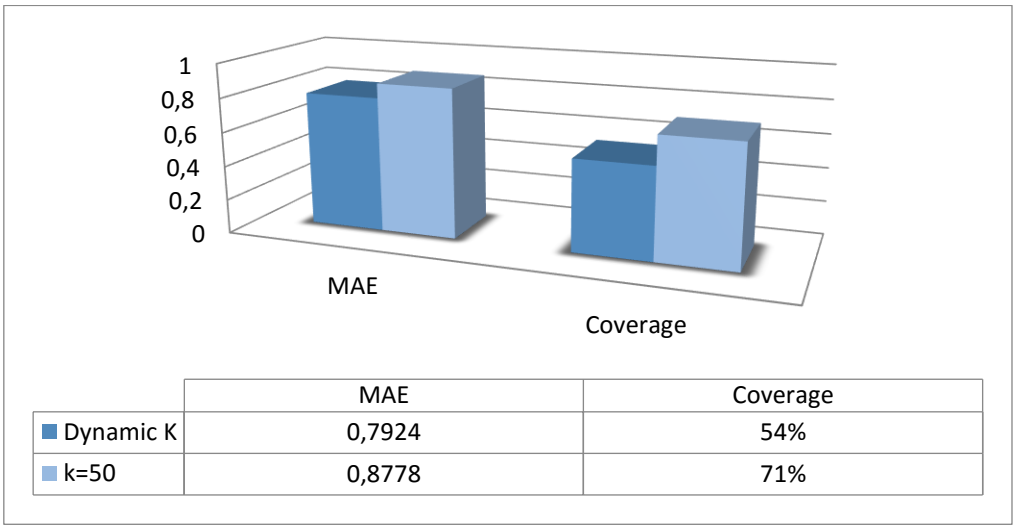
Table 4. Dynamic *k* value look-up table for items

	The best *k* value for item	**MAE for specifying *k* value**
Item1	40	1,2069
Item2	15	0,9103
Item3	15	0,9538
Item4	50	0,9863
Item5	35	0,7752
Item6	5	0,6133
Item7	20	1,1925
Item8	20	1,2773
Item9	25	0,9111
Item10	50	0,8465

In order to observe the effects of using dynamic *k* value for each user in prediction in user-based CF, we randomly selected five items for each user to be estimated and we ran our algorithm for 100 times with both dynamic *k* value for each user and a constant *k* value that is 50. The results showed that using dynamic *k* value in *k*-nn algorithm for rating prediction dramatically increases the accuracy of the estimations. The higher accuracy means, the lower MAE. On the other hand, using dynamic *k* value in *k*-nn decreases the coverage. Our procedure which was used in the test phase in our study is shown in Procedure 3.

Procedure 3 Rating Prediction Procedure in the Test Phase (user-based)
Require: $\boldsymbol{U_{n*5}}$ test set contains 5 test item for each user, dynamic *k* value for each user(k look-up table)
1:**for all** users **in***U*(*i*← 1 to *n*) **do**
2: k← read the best k value of the user from look-up table
3: **for all** test items **in***U*(*i*← 1 to 5**) do**
Calculate weights for each user between other users and specify k nearest neighbors by sorting weights in descending order
4: $\boldsymbol{P_{n,m}}$←$\mathbf{p_{a,i}}$ (calculate by using Equation 2)
5: **end for**
6:**end for**
Calculate MAE value and the coverage of the system by substracting the prediction matrix by the real rating values of the items given by the users.
7:mae←$\boldsymbol{P_{nxm}}$- $\boldsymbol{R_{nxm}}$
8:**end for**

The MAE results and the coverage that is a measure of the percentage of generating estimation on given test set is given comparatively in Figure 2.



	MAE	Coverage
Dynamic K	0,7924	54%
k=50	0,8778	71%

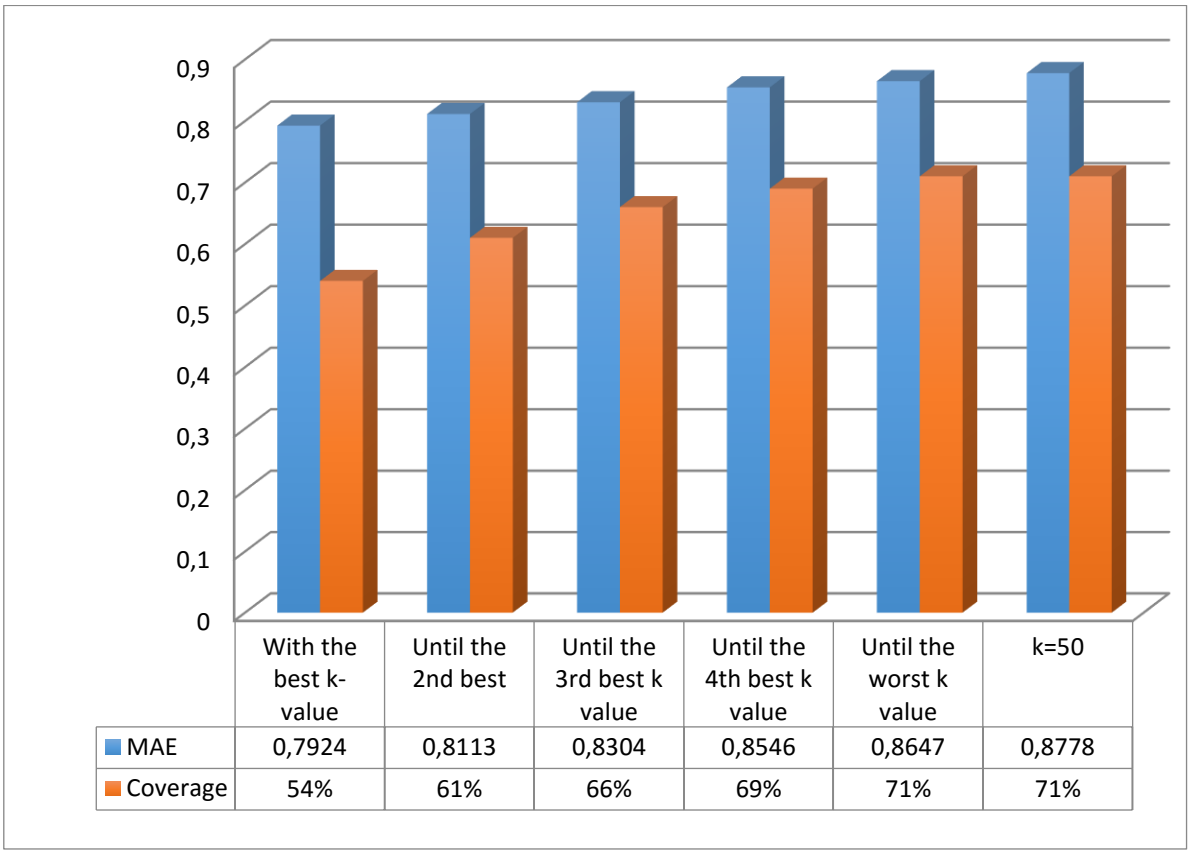
Figure 2. MAE and Coverage results for dynamic *k* values and *k*=50

While the accuracy of the prediction is crucial for some systems or web sites, the coverage can be so important for others. After we obtained the results and got there is a tradeoff between accuracy and coverage, we decided to try extending the *k* look-up table for the user and putting the values for the second best *k* value for the user and then the third best *k* value for the user and so on. By using the MAE values in Table 1, the extended dynamic *k* look-up table composed as shown in Table 5 for the first ten users.

Table 5. *k* values in ordered will be used in *k*-nn for each user

	1st best *k*	2nd best *k*	3rd	4th	5th	6th	7th	8th	9th	10th
User1	50	45	40	35	25	30	5	10	20	15
User2	45	50	40	20	35	15	30	25	5	10
User3	40	35	30	45	25	50	15	20	10	5
User4	15	20	25	30	50	35	45	40	10	5
User5	50	30	35	25	45	40	20	15	10	5
User6	15	25	30	20	45	50	35	40	10	5
User7	5	15	10	45	20	50	40	35	30	25
User8	20	50	15	40	45	35	30	25	5	10
User9	40	30	35	50	45	25	20	15	5	10
User10	40	45	35	20	15	25	30	10	50	5

For the systems where the coverage is crucial, we tried to design a new algorithm. According to our algorithm, if the system cannot produce estimation for an item for a specific user by using dynamic *k* value for this user, it will retry to produce prediction by using the second best *k* value for this user. If it cannot produce again, it will retry to produce by using third and so on. As we tried more *k* values to generate a prediction, the percentage of the prediction generation increased but the accuracy of the estimation decreased as shown in Figure 3.



	With the best k-value	Until the 2nd best	Until the 3rd best k value	Until the 4th best k value	Until the worst k value	k=50
MAE	0,7924	0,8113	0,8304	0,8546	0,8647	0,8778
Coverage	54%	61%	66%	69%	71%	71%

Figure 3. MAE and Coverage results according to the users' top-n *k* values

We also tried to use dynamic *k* value in the item-based CF. Like user-based CF, we also constituted *k* look-up table for each item in the dataset, and we used these *k* values while estimating prediction value for a specific item. In order to observe the effects of the *k* values in item-based *k*-nn algorithm, we employed *k*-nn algorithm with both dynamic *k* value and a constant *k* value that is 50. While producing prediction, we used Procedure 4. We again randomly selected five items for each user in the dataset and tried to produce a prediction value. Using dynamic *k* value also served the purpose in item-based CF as shown in Table 4.

Procedure 4 Rating Prediction Procedure in the Test Phase (item-based)
Require: $\boldsymbol{U_{n*5}}$ test set contains 5 test item for each user, dynamic *k* value for each item (k look-up table)
1:**for all** users **in***U*(*i*← 1 to *n*) **do**
s2: **for all** test items **in***U*(*i*← 1 to 5**) do**
3: k← read the best k value of the item from look-up table
Calculate weights for active item between items and specify k nearest neighbors by sorting weights in descending order
4: $\boldsymbol{P_{n,m}}$←$\mathbf{p_{a,i}}$ (calculate by using Equation 4)
5: **end for**
6 :**end for**
Calculate MAE value and the coverage of the system by substracting the prediction matrix by the real rating values of the items given by the users.
7:mae←$\boldsymbol{P_{nxm}}$- $\boldsymbol{R_{nxm}}$
8:**end for**

Table 4. MAE values for dynamic *k* value and *k*=50 in item-based *k*-nn

	MAE value
Dynamic *k* value	0.9722
k=50	1.1045

6. CONCLUSIONS AND FUTURE WORK

Collaborative filtering and *k*-nearest neighbor algorithms are frequently used in classification algorithms as well as for rating prediction and recommender systems. Especially online booking, movie and shopping sites where there are many users trying to offer the best recommendations to their users. Today's trend in recommender systems is to be able to find the point of the shot while finding suggestions towards the user. For this purpose, new technologies such as big data emerged in recommender systems. Although some recommendations can be instantly made to the users by bringing together different data sets using the used technologies to analyze, the basic stones used by the recommender systems are algorithms like relatively same used in the past such as CF, Support Vector Machines, etc. As a result, besides the development of the technologies, the improvement of the basically used algorithms will increase the success rate of the recommendations presented to the user.

The improvement of the conventional algorithms in collaborative filtering, *k*-nearest neighbor-based method will make it easier to reach the goal of making a shot at the suggestions presented to the user. For this reason, in this study, we decided to give different *k* values for each user and item in the *k*-nn algorithm to get better estimations, and we succeeded in this. We tried to determine the best k-values for each user and item in the dataset by trying many *k* values during the training phase of the algorithm, and then we used these *k* values in the *k*-nn algorithm to predict rating users gave to the items. The results showed that using user or item specific *k* values instead of using static *k* value in the *k*-nn algorithm dramatically increase the success of prediction estimations. For some users or items, a higher k value allows us to achieve lower MAE values, while for some users or items smaller *k* values increase the estimation success rate.

So far, we have tried to determine the best *k* value for each user and item by trying different *k* values on that user, and we use the *k* values we determined during the test phase. Especially in the systems which have many users and many items, it will be a long process to try *k* values for each new user and item to determine the value of *k* to be used in recommendations for specific user, so it will not be possible to use *k*-nn algorithm instantly for new coming users by using the users' best *k* value especially while using user-based *k*-nn algorithm. We plan to be able to determine the best *k* values online by looking at specific attributes of the new user in the system, without having to go through any training phase after the new user provides a certain number of rating values.

REFERENCES

[1] White T. Hadoop the definitive guide. O'Reilly US, 2015.

[2] Ricci F, Rokach L, Shapira B. Recommender Systems Handbook. Springer US, 2011.

[3] 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, 2005; 17.6: pp. 734-749.

[4] Bilge A, Yargıç A. Improving accuracy of muti-criteria collaborative filtering by normalizing user ratings. Anadolu University Journal of Science and Technology a- applied science and engineering, 2017; 18.1: pp. 225-237

[5] Keller JM, Gray MR, Givens JA. A fuzzy k-nearest neighbor algorithm. IEEE transactions on systems, man and cybernetics, 1985; 15.4. pp. 580-585

[6] Herlocker JL, Konstan JA, Borchers A, Riedl J. An algorithmic framework for performing collaborative filtering. In: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, 1999, pp. 230-237.

[7] Su X, Khoshgoftaar TM. A survey of collaborative filtering techniques. Advances in Artificial Intelligence, 2009; 4.

[8] Adomavicius G, Manouselis N, Kwon Y. Multi-criteria recommender systems. In: Recommender Systems Handbook. Springer US, 2011; pp.769-803.

[9] Jannach D, Karakaya Z, Gedikli F. Accuracy improvements for multi-criteria recommender systems. In: Proceedings of the 13th ACM Conference on Electronic Commerce. ACM, 2012; pp. 674-689.

[10] Resnick P, Iacovou N, Suchak M, Bergstrom P, Riedl J. GroupLens: an open architecture for collaborative filtering of netnews. In: Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work. ACM, 1994; pp. 175-186.

[11] Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: Proc. of the 14th Annual Conf. on Uncertainty in Artificial Intelligence, 1998; pp. 43–52.

[12] Delgado, J., Ishii, N.: Memory-based weighted majority prediction for recommender systems. In: Proc. of the ACM SIGIR'99 Workshop on Recommender Systems (1999)

[13] Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: KDD'08: Proceeding of the 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp. 426–434. ACM, New York, NY, USA, 2008.

[14] Kaleli C. An entropy-based neighbor selection approach for collaborative filtering. Knowledge-based Systems, 2014; pp. 273-280.

[15] Herlocker J, Konstan JA, Riedl J. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms, Inform Retrieval, 2002; 5 (4) 287–310.

[16] Kim TH, Yang SB. An effective threshold-based neighbor selection in collaborative filtering, in: Proceedings of the 29th European conference on IR Research. ECIR'07, Springer-Verlag, Berlin, Heidelberg, 2007; pp. 712–715.

[17] Liang ZX, Jun Bo G, An approach of selecting right neighbors for collaborative filtering, in: Proceedings of the Innovative Computing, Information, and Control (ICICIC), 2009 Fourth International Conference on, 2009; pp. 1057–1060.

[18] Koren, Y. Factor in the neighbors: scalable and accurate collaborative filtering, ACM Trans Knowl Discovery Data, 2010; 4 (1) 1–24.

[19] Sarwar G, Karypis B, Konstan J, Riedl J. Item-based collaborative filteringrecommendation algorithms, in: Proceedings of the 10th InternationalConference on World Wide Web, 2001, pp. 285–295.

[20] Aggarwal, CC. Recommender Systems: The Textbook. Springer, 2016.

[21] Sarwar B, Konstan J, Borchers A, Herlocker J, Miller B, Riedl J. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering systems, Proceedings of 1998 Conference on Computer Supported Collaborative Work, 1998.

[22] Shardanand U, Maes P. Social information filtering, Proceedings of ACH CHI'95 Conference on Human Factors in Computing Systems, 1995; pp. 210-217