JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 1097-1118 (2018)
DOI: 10.6688/JISE.201809_34(5).0001

A Distributed Neural Filter for Finding Depth-*k* Skyline Friends in Social Networks

YI-CHUNG CHEN AND HENG-YI SU+
Department of Industrial Engineering and Management
National Yunlin University of Science and Technology
Yunlin, 640 Taiwan
+*Department of Electrical Engineering*
Feng Chia University
Taichung, 407 Taiwan
E-mail: chenyich@yuntech.edu.tw; hengyisu@fcu.edu.tw

Finding similar users in a social network is an essential element of personalized recommendation systems. Most traditional algorithms of this kind consider all the dimensions of user data as a whole and use this merged information to search for similar users. However, such a method has its flaws, as user information in various dimensions is generally independent. Recently multi-criteria skyline queries have been applied to search for similar users. However, these applications are only suited to centralized environments, which is not a common type of environment among social networks. Hence, this paper introduces a distributed neural filter to search for similar users in a distributed environment. By using the proposed filter, we can enhance the speed of identifying whether a user is similar to you in the distributed environment. That is, the operating speed of recommendation systems can be improved. Simulation results demonstrate the effectiveness and efficiency of the proposed identifier.

Keywords: social network, recommendation system, skyline query, neural network, distributed environment.

1. INTRODUCTION

In recent years, an increasing number of researchers have focused on personalized recommendation systems [14, 21], which tailor recommendations to users on subjects such as products, attractions, and travel itineraries. The first step of most personalized recommendation systems is the search for friends or users similar to the user in question. The online information that these individuals provide serves as the basis of these recommendations. Suppose, for instance, that User A wants to take a trip to an unfamiliar city, which is home to four attractions: science museum and art museum, department store, and central park. User A requests a recommendation from the system. The system receiving this request has access to the information presented in Table 1, which includes types of attractions, the cities they are located in, and the ratings that other users have given them. For example, the value 0.9 in (A, history museum) means that User A has been to history museum and given it a rating of 0.9. In contrast, the "–" in (A, science museum) means that User A has not been the science museum. Upon receipt of User A's request, the system first searches Table 1 for all the users that have given ratings similar to those given by User A. The table shows that Users B and D are more similar to User A than Users C, E, and F are because they both gave history museum high ratings and gave

Received August 26, 2017; revised October 7, 2017; accepted November 14, 2017.
Communicated by Nien-Lin Hsueh.

shopping mall and nature park extremely low ratings, as did User A. Users C, E, and F favor the opposite, having given shopping mall and nature park high ratings but given history museum lower ratings. After identifying Users B and D, the recommendation system then makes recommendations to User A based on the ratings given by Users B and D to science museum and art Museum, department store, and central park. As science museum received higher ratings from Users B and D than the other three attractions did, the system therefore recommends that User A go to science museum.

Table 1. An example of recommendation systems.

Attractions	Type	city	A	B	C	D	E	F
Science Museum	Intellectually-related	New York	–	0.95	0.14	0.97	0.07	0.2
Art Museum	Intellectually-related	New York	–	0.89	0.2	0.85	0.06	0.15
Department Store	Shopping-related	New York	–	0.13	0.95	0.11	0.98	0.93
Central Park	Nature-related	New York	–	0.08	0.97	0.13	0.99	0.84
History Museum	Intellectually-related	Houston	0.9	0.91	0.06	0.94	0	0.1
Shopping Mall	Shopping-related	Houston	0.1	0.2	0.88	0.15	0.99	0.95
Nature Park	Nature-related	Houston	0.05	0.01	0.87	0.1	1	0.98

Conventionally, when recommendation systems search for similar users, they use *k*-Nearest Neighbors [20, 23], cosine similarity [21] and the *k*-means algorithm [19, 30] to consider multiple user aspects simultaneously. However, this approach has its drawbacks because the information of users in different aspects is generally independent and uncorrelated [7]. Say that there is a User G in Table 1, who has given history museum, shopping mall, and nature park the ratings 0.93, 0.99, and 0.02, respectively. Clearly, the preferences of User G in intellectually-related attraction is similar to that of User A, so the ratings that User G has given to science museum and art museum should also be of value to User A. However, none of the *k*-Nearest Neighbors, the cosine similarity and the *k*-means algorithm would consider User G as similar to User A because they differ too greatly in one aspect. This presents a significant shortcoming in existing approaches.

Recently, researchers have begun to apply a well-known multi-criteria search algorithm, the skyline query [11, 16, 18, 25] to solve this problem. The most prominent feature of the skyline query is that it can consider various conditions (or dimensions) separately and assist users in retrieving the data points that they need based on all or some of the conditions. The first study to apply the concept of skylines to the relationships between users and their friends in social networks was that of Peng *et al.* [27]. They proposed that performing skyline queries based on various correlations between users and their friends can assist users in identifying those that are the most important to them. They furthermore designed a member promotion algorithm to guide users in increasing their importance to other users. Later, Peng *et al.* [28] presented the Infrasky algorithm which is a modification of their previous work to allow it to operate on equal-weighted social networks. With regard to the relationships between users and their social network friends, Emrich *et al.* [8] additionally considered geographical information and claimed that this approach can assist users in finding friends that they are closer to in both affection and distance. Also. Zheng *et al.* [34] proposed a method of friend recommendation that combines of skyline query and check-in information to find promising friends. But we should be noticed that despite the successful applications in these works, none of

them helped users identify other users with similar preferences. Recently, Chiu *et al.* [7] employed the depth-*k* skyline query to identify the depth-*k* sky line users of target User A and referred to them as the users similar to User A. Fig. 1 shows an example given by [7], where axes *X* and *Y* respectively measure the difference between User A and other users in their preferences for intellectually-related attractions and shopping-related attractions. For instance, B(0.1, 0.5) indicates that the difference between Users B and A is 0.1 point in intellectually-related attractions and 0.5 points in shopping-related attractions. In this example, we can see that a higher degree of similarity exists between Users B and A than between Users E and A because the preferences of User B are closer to those of User A than those of User E are in both types of attractions. In this case, we say that B dominates E. The depth-*k* skyline query proposed by [7] identifies the users that at most are dominated by *k* other users. The black dots in Fig. 1 display the results of a depth-2 skyline query, in which Users B, C, and D are not dominated by any other users, Users E, F, and G are dominated once, and Users H, and I are dominated twice. Clearly, these users all show preferences that are similar to those of User A in either intellectually-related or shopping-related attractions. Thus, they have reference value for User A and should be retrieved. This approach effectively solves the issue of only considering independent dimensions simultaneously. It also enables users to find a wider range of friends that exhibit similar preferences to them rather than only the users situated on the skyline. However, this approach requires a substantial number of dominance checks which slow down computation [13]. For this reason, [32] proposed an R-tree based algorithm to enhance the operation speed of depth-*k* skyline query. Also, as the number of similar friends returned by [7] can be quite unstable, which may cause some problems in recommendation systems, the algorithms in [32] also ensured that the number of similar friends they returned is fixed at a constant value. But we should be noticed that as the attribute values of users in social networks change frequently, it is quite difficult for us to store attribute values of users by R-tree. That is, the algorithms in [32] can be useless in real world applications. Hence, [13] proposed the concept of skyline regions. For each User A, their algorithm first identifies the skyline users and then draws a skyline region based on the threshold σ given by User A. All of the users within this skyline region are considered to be similar to User A. Fig. 2, for example, exhibits the skyline region of Fig. 1. Assuming that the target user is User A, we first identify the skyline users from among the friends of User A, who are Users B, C, and D as shown in Fig. 2. Next, we draw lines extending from points B, C, and D towards axes *X* and *Y* with a length of σ, which result in three virtual points B', C', and D'. The region formed by these three virtual points in the lower left corner is the skyline region, as shown in Fig. 2. In this manner, only the location of a point is needed to determine whether a user may be a skyline user, which negates the need for numerous dominance checks and therefore accelerates the processing speed of the algorithm. Nevertheless, we must understand that while this method can quickly identify users that are similar to User A, the algorithm only considers the absolute values in friend data but not the dominance relationships between friends. As a result, the number of similar users found can fluctuate considerably. It is even possible that the skyline region contains no similar users (such as with D and D' in Fig. 2) or too many similar users (such as with B and B' in Fig. 2). Such circumstances can lead to problems in the resulting recommendations. Furthermore, as mentioned above, a comprehensive review of past studies reveals that their proposed algorithms are only suitable

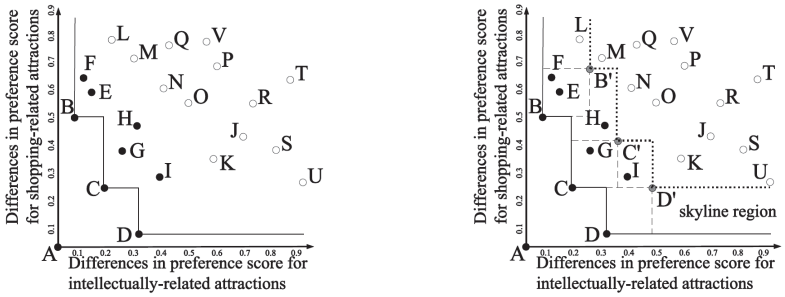
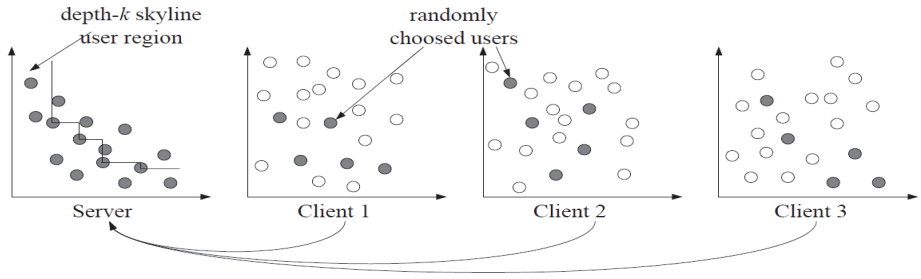


Fig. 1. An example of finding similar friends in [7]. Fig. 2. An example of finding similar friends in [13].

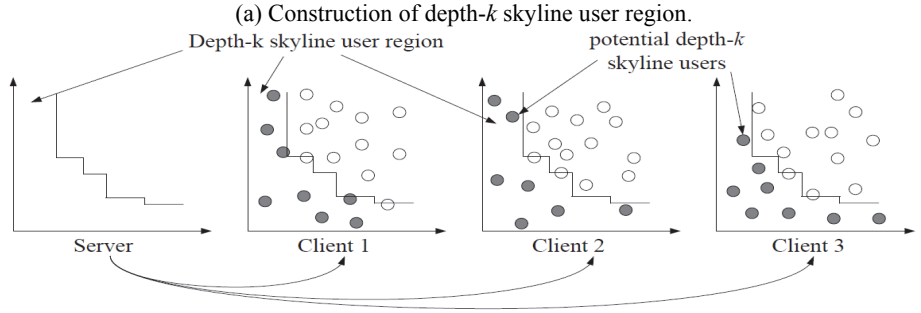
for centralized environments, yet most existing social networks are distributed environments. It is therefore necessary to develop an approach that overcomes these limitations.

This paper proposes the concept of depth-*k* skyline user regions to assist users in quickly identifying the depth-*k* skyline users of User A in distributed systems. These users are considered to be similar to User A. The concept of this paper is as shown in Fig. 3. Once initialized, the algorithm searches all of the datasets stored in the mainframe of the distributed system to derive the depth-*k* skyline user region. All of the users within this region are potential depth-*k* skyline users of User A, whereas those outside the region are not (Fig. 3 (a)). In the next step, this region is sent back to each distributed host to determine which users may be the depth-*k* skyline users of User A, as shown in Fig. 3 (b). The information of these users is then collected by the main server to confirm whether they are the depth-*k* skyline users of User A, as shown in Fig. 3 (c). This method of using depth-*k* skyline user regions is faster than conducting numerous dominance checks [13]. It also identifies the depth-*k* skyline users of the target user and is applicable to environments with distributed systems.

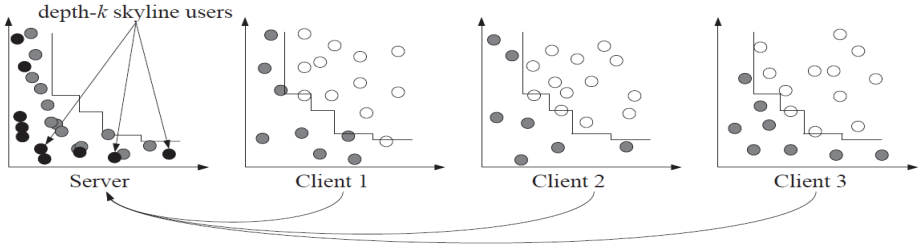
Although using depth-*k* skyline user regions to identify depth-*k* skyline users avoids numerous dominance checks, determining whether a user falls within the depth-*k* skyline user region of a particular user is still time-consuming. For instance, if we want to determine whether any of the users in Fig. 4 fall within the depth-*k* skyline user region of User A, each candidate must still undergo dominance checks with points X, Y, and Z. This method is therefore not highly scalable. The introduction of artificial neural networks reduces the computational burden of determining whether a user falls within the skyline-related region of User A. This approach has been used by Chen and Lee [6], but their algorithm can only determine whether a data point falls within a skyline candidate region (*i.e.*, data points that fall within this region may be skyline points) but not whether a data point falls within a depth-*k* skyline region. Moreover, their algorithm only used two features, min and sum, of a data point to reach their goal. This can be quite rough, because these two features cannot always be the best solution for identifying all kind of datasets. Other features should also be considered in the proposed problem. Last but not least, their method can only be used in centralized environments, not distributed environments; therefore, it is not suitable for this work. Chen and Lee [5] have developed another algorithm for finding depth-*k* skyline points in distributed environment. However, their work only proposed a naive algorithm, which inputs all attribute values of the data point into the neural network. In such a case, many redundant information will be inputted into the neural network and thus the precision of their algorithm can be quite



(a) Construction of depth-*k* skyline user region.



(b) Sifting possible depth-*k* skyline users in each client.



(c) Finding depth-*k* skyline users from possible depth-*k* skyline users.

Fig. 3. The concept of this paper.

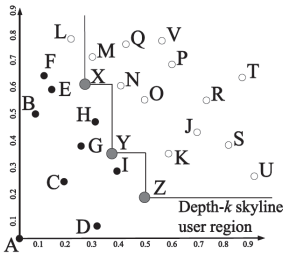


Fig. 4. The difficulty of examining if a user locates in the depth-*k* skyline user region.

low. That is, their algorithm cannot be apply to the real world applications. Moreover, the distributed environment they consider is different to that of social network. Hence, we cannot use their algorithm to solve our problem directly. Hou *et al.* [13] also used artificial neural networks to determine whether a user is located in the skyline-related region, but the precision of their method was not high, and it also could not be used in distributed systems. In view of this, we propose a novel model based on artificial neural

networks, called the distributed neural filter to swiftly and precisely determine whether a user falls within the depth-*k* skyline user region of the target user. The results of our simulation experiment demonstrate the efficiency and effectiveness of the proposed approach.

The remainder of this paper is organized as follows. Chapter 2 presents related work. Chapter 3 introduces the algorithm and model of the distributed neural filter, and Chapter 4 outlines the experiment simulations. Chapter 5 presents our conclusions.

2. RELATED WORKS

2.1 Skyline Query

Four skyline query algorithms are in common use: block nested-loop algorithms, divide-and-conquer algorithms, the Sort and Limit Skyline algorithm, and branch-and-bound skyline algorithms. These algorithms are introduced in detail in the following.

2.1.1 Block Nested Loops Algorithm (BNL)

BNL algorithms [4] are the most intuitive processing methods for skyline queries. The primary concept behind them is to compare each data point with all of the other data points in the database and check whether they are dominated by any other data points. If not, the data point is placed in the skyline candidate region; if it is, then it is eliminated immediately. Once all of the comparisons have been made, the points remaining in the skyline query candidate region are the final results of the skyline query. A limitation of this method is that while it is intuitive, it is inefficient; the time that it requires to process a query is proportional to the amount of data. The substantial amount of data that we processed in this study discounted this type of algorithm as a candidate for this study.

2.1.2 Divide and Conquer Algorithm (DAC)

The primary concept behind DAC algorithms [4] is to first divide large amounts of data into groups, search each for skyline data points, combine these skyline data points, and then conduct another skyline query to derive the final skyline query results. This method filters out unlikely candidates beforehand, significantly reducing the number of data points that must be considered during the query. In Fig. 5, we can clearly see that the data points in region *B* must be dominated by the data points in region *C*. For this reason, we do not have to check the data points in *B* for the skyline query. However, the improvement in computational burden made by this approach, while positive, was not sufficient, prompting researchers to seek alternatives.

2.1.3 Sort and Limit Skyline algorithm (SaLSa)

The main idea of the SaLSa algorithm [2] is to use threshold values to search for potential skyline data points. The threshold values are derived from feature values of the data, which may be the minimum, total, or product of data points in each dimension. The threshold values record the upper limits of the feature values of current skyline candidates. For each data point, its feature values are first calculated and compared with the

threshold values. If the feature values are less than the threshold values, then the data point may be a skyline data point. If not, then it cannot be a skyline data point. This algorithm also uses feature values to rank the data points in ascending order. In this manner, if the feature values of a data point surpass the threshold values, then all of the later data points do not need to be checked because their feature values will certainly be greater than the threshold values.

2.1.4 Branch and Bound Skyline algorithm (BBS)

BBS algorithms are currently the most widely applied type of skyline query algorithm [24] because they can significantly reduce the number of data points that need to be checked during skyline queries. This is achieved by indexing and contrasting them with BNL algorithms, which use all of the data points, and DAC algorithms, which use most of the data points. Below, we give an example to illustrate this advantage of BBS algorithms. Fig. 6 (a) displays the first step of a BBS algorithm. The root node M_0 must contain skyline data points and is thus divided into M_1 and M_2. Next, Fig. 6 (b) shows that only M_1 contains skyline data points and therefore must be divided, while M_2 does not contain skyline data points and therefore does not have to be divided. Finally, Fig. 6 (c) indicates that both M_3 and M_4 contain skyline data points, necessitating further checking. This procedure shows that because M_2 is never divided, data points B, C, H, I, and J in M_2 will not be processed by the BBS algorithm. In other words, BBS algorithms checks fewer data points than conventional algorithms and thus require less time to complete skyline queries.

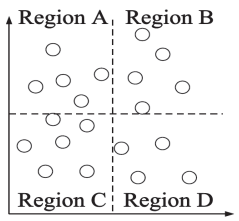
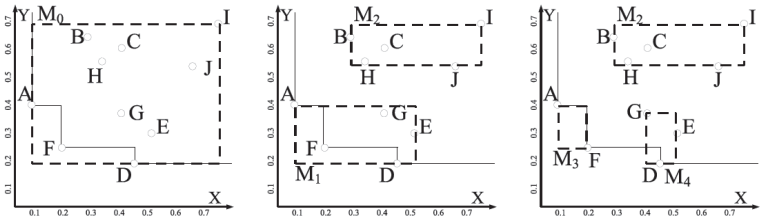


Fig. 5. An example of divide and conquer algorithm.



(a) First step. (b) Second step. (c) Third step.

Fig. 6. An example of branch and bound skyline algorithm.

2.2 Recommendation System

The existing recommendation systems can be classified into three types, the Collaborative filtering, the content-based recommendation and the knowledge-based recommendation. The detail of these three types of recommendation system are introduced in the following.

2.2.1 Collaborative filtering

Collaborative filtering classifies users into groups according to similarities in inter-

ests, and then recommends items to these users as members of that group [10, 29]. There are two types of collaborative filtering: memory-based and model-based [17]. The memory-based approach finds users with similar interests within a pre-established database [12], whereas the model-based approach uses clustering [17]. The memory-based approach is the most-used recommendation method and thus it is also the main recommendation method this work focus on.

2.2.2 Content-based recommendation

Content-based recommendation identifies items in accordance with the profile or preferences of the user [1, 10]. This method considers only existing users of the system and can be adapted to users with special interests. The content-based recommendation process includes two steps [3]: (1) acquisition of user profiles, including logs, cookies, and records of previous interactions obtained via data capture applications; (2) finding similar friends for the recommended users by using the information retrieved in the first step.

2.2.3 Knowledge-based recommendation

Knowledge-based recommendation involves the correlation of user profiles with the characteristics of candidate items selected by the users. This can be implemented using two types of classification: case-based and rule-based reasoning [15]. The case-based approach applies empirical rules to previously obtained data in order to find solutions to new problems [31]. In contrast, the rule-based approach uses rules defined by experts without reference to the historical records of specific users [22].

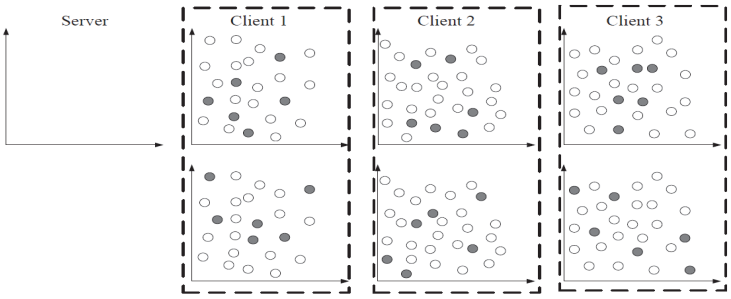
3. THE ALGORITHMS AND MODEL OF THE DISTRIBUTED NEURAL FILTER

Supposing that the target user is A; the process of the distributed neural filter actually consists of four parts; (1) It retrieves information from the clients and then constructs a depth-k skyline user region for A; (2) It constructs the distributed neural filter, which uses neural network to learn and approximate the range of this depth-k skyline user region; (3) It transmits the distributed neural filter to each client, filters out those users that may possibly become A's depth-k skyline user and feeds back these users to the server; (4) The server will find A's depth-k users out of these fed back users. These four parts are explained respectively as follows.

3.1 The Construction of Target User's Depth-k Skyline User Region

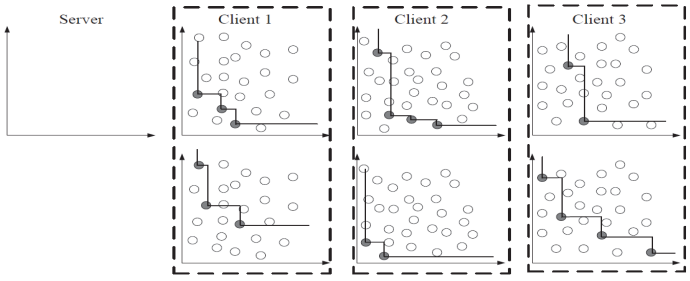
To construct a depth-k skyline user region for target user A in the distributed system, two steps are required. The first step is to retrieve the users who may become A's depth-k user, and to transmit them back to the server. In the second step, the server uses these fed back users to derive A's depth-k skyline user region. These two steps are described respectively as follows.

It is assumed that a specific user intends to find his user with depth-*k*. There are m groups of clients in the community network; and in each group of clients, there are *u* users. To find A's possible depth-*k* users in step 1, *n* user groups (there will be *p* users in a user group) will first be randomly chosen in each client. Hence, $m \times n$ user groups will be selected from the whole community network. Furthermore, skyline query will be conducted for each user group; and skyline results are derived. At this stage, the most commonly used skyline query algorithm BBS algorithm [24] is not used for query. This is due to the fact that when the BBS algorithm is employed, all users must use the R-tree [9] index. However, in this step, each user group was randomly chosen, therefore the retrieved users cannot possibly exist in the R-tree. Hence it is inconvenient to use the BBS algorithm for querying. Of course, the R-tree can first be established for each user group prior to query but this method will consume extra time and will slow down the algorithm speed and hence it is not considered. Another skyline query algorithm, the SFS algorithm [2], is employed here to find the skyline results for each user group. It is the fastest way to find the skyline results without using any index structure. Eventually, the skyline results of each user group will be fed back to the server. Note that when the system decides the *n* value in step 1, it assumes $m \times n \geqq k$. This is because in some special cases, if the relationship between A and other users are in correlated distribution, the user A might merely find a single skyline user [2, 4, 24]. Meanwhile, if $m \times n$ is less than *k*, and all $m \times n$ groups can only find a single skyline user, the skyline results received by the server will be less than *k*; consequently, there is no way to form a depth-*k* skyline user region. Besides, the *p* range in each user group is recommended to be between $[0.01u, 0.1u]$. The actual value should be based on client performance and the size of u. In general, a larger *p* will result in less non-depth-*k* skyline users in the depth-*k* skyline user region established in this step. However, the time required for skyline query grows exponentially with the size of the data set [2, 4, 24]. A bigger *p* implies each client will need more query time; hence the speed of the overall algorithm is further reduced. In other words, this is a trade-off. The system designer can set a suitable *p* according to the conditions and needs of his own system. Fig. 7 is used to explain this step. In this figure, there are three sets of clients and one set of server. It is supposed that user A intends to take out the user in Depth-4. Based on target user A, the system will select two user groups from each client (the gray points in Fig. 7 (a)). Next, the system will find the skyline users for six user groups, as shown in Fig. 7 (b). These skyline users will be transmitted back to the server.



(a) Randomly choosing some users in each client.

Fig. 7. An example of retrieving depth-*k*-skyline-user-region-related information from clients.



(b) Finding skyline users in each client.

Fig. 7. (Cont'd) An example of retrieving depth-*k*-skyline-user-region-related information from clients.

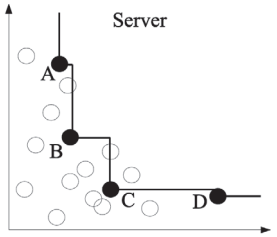


Fig. 8. An example of constructing the depth-*k* skyline user region from the retrieved information.

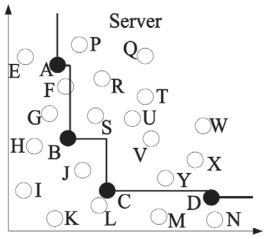
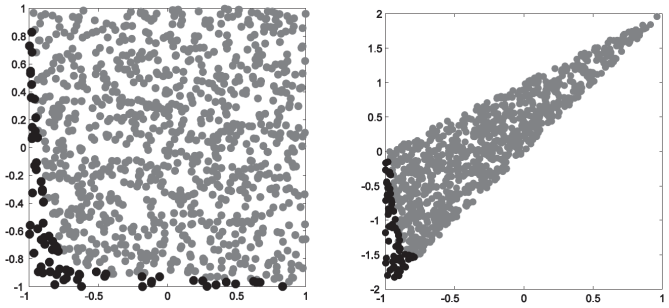


Fig. 9. An example of generating training data for the neural network.

In step 2 of this part, the server uses the users being fed back from each client to obtain target user A's depth-*k* skyline user region. It is assumed that the fed back users from each client are q in total. The system will perform dominating comparison for each user against the other $q-1$ users in this data set; and will calculate the number of times each user was being dominated. The users that were dominated k number of times will be retrieved. The range enclosed by the retrieved users in the lower left corner is A's depth-*k* skyline user region. All users in this region have the potential to become A's depth-*k* skyline user; and those falling outside this region are unlikely to become A's depth-*k* skyline users. Fig. 8 is used to explain this procedure; users in this figure are the users being fed back by each client in Fig. 7. In Fig. 8, A, B, C, and D are being dominated four times and are specifically picked out. The enclosed lower left corner area is A's depth-*k* skyline user region.



(a) (b)

Fig. 10. An example of mapping users from original coordinates to featured coordinates; (a) before mapping; (b) after mapping.

3.2 The Construction of the Distributed Neural Filter

This section describes how to construct the distributed neural filter (*i.e.*, using neural networks to learn and approximate the target user A's depth-*k* skyline user region). To attain this target, the following three steps are required; (1) From A's depth-*k* skyline user region, find out proper training data to train the neural networks; (2) Retrieve the characteristic values of these training data; (3) Construct proper neural network architecture; and conduct training via the usage of the characteristic values of the training data. These three steps are described respectively as follows.

To accelerate the speed of the overall algorithm, step 1 in this part employs the simplest way to generate the neural network training data. It is to say, using a random manner to produce equivalent virtual users in depth-*k* skyline user region and non-depth-*k* skyline user region. These virtual users are regarded as the input data of the neural network. Furthermore, every one of these virtual users is also given an additional target value. This value represents the region that an individual virtual user is located in; and is regarded as an output of the neural network. If this value is 1, the user falls in the depth-*k* skyline user region. Conversely, if the value is −1, the user falls in the non-depth-*k* skyline user region. Note that there are two reasons that virtual users (not real users) are used here. The first reason owes to the fact that it is quite time-consuming to find a real user in the two regions. It is impractical to use such an approach. The second reason is because these virtual users are used only to train the neural network and are irrelevant to the final results. Hence it is not necessary to use real users. Besides, it must also be noted that the number of virtual users generated in these two regions must be equal, so as to help the neural network to more accurately approximate the depth-*k* skyline user region. Fig. 9 provides the example in this step. Because E to N fall into the depth-*k* skyline user region, their target values are set to 1. Conversely, the target values of *P* to *Y* are set to −1

Step 2 of this part is to retrieve the characteristic values of training data. This step is expected to significantly improve the problem of insufficient judging accuracy in previous related neural network papers [5, 13]. Past methods mostly directly input the user values to the neural network for training. In this situation, all the depth-*k* skyline users will fall on the edges of the data sets. This is unfavorable to neural network training. As shown in Fig. 10 (a), the gray dots are general users; the black dots are depth-*k* skyline users. However, the characteristic value method proposed in this paper will reflect users from the original coordinate axes to another coordinate using characteristic values as coordinate axes. As shown in Fig. 10 (b), the data sets in Fig. 10 (a) have been reflected to the coordinate using minimum and summation as coordinate axes. Through this step, it

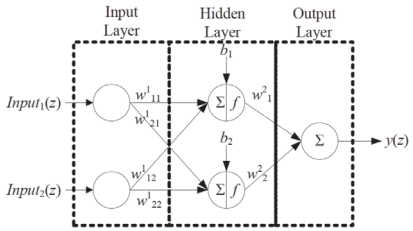


Fig. 11. The model of the proposed distributed neural filter.

can be observed that the depth-k skyline users originally were scattered at the edges of data sets are now concentrated together. In this situation, the range of the depth-k skyline user region will be shrunk and concentrated; and now the neural network more accurately approaches the depth-k skyline user region.

The characteristic values used in this paper are minimum, summation, and volume. All these characteristic values had been proven by Bartolini *et al.* [2] that they can be used to identify whether a data point is a skyline point. It is assumed that a virtual user A with coordinate values (a_1, a_2, ..., a_d) is used to train the neural network, where d is the dimensional quantity. The equations for these three characteristic values can be written respectively as:

$$\text{Min(A)} = \text{minimum}(a_1, a_2, ..., a_d), \tag{1}$$

$$\text{Sum(A)} = \sum_{i=1}^{d} a_i, \tag{2}$$

$$\text{Vol(A)} = \prod_{i=1}^{d} a_i, \tag{3}$$

where Min (A), Sum (A), and Vol (A) represent the minimum, summation, and volume of virtual user A. The experimental portion of this paper will explore the characteristic value combinations among (Min (A), Sum (A)), (Min (A), Vol (A)), (Sum (A), Vol (A)), (Min (A), Sum (A), and Vol (A)); and will derive the combination which can be most helpful for the neural network to approach the target depth-k skyline user region.

Step 3 of this part is to construct a neural network to approximate the depth-k skyline user region of the target. The neural network expected to be used in this paper is shown in Fig. 11. It contains input layer, hidden layer, and output layer. The node number of input layer is 2 or 3, which is determined by the input characteristic value. The node number of hidden layer is equal to the node number of input layer. For example, if the characteristic value entered is (Min (A), Sum (A)), the input layer and the hidden layer of the neural network should contain two nodes. The output layer will have only one node and is responsible for determining whether the user falls in the approximate depth-k skyline user region (between −1 to 1). If the value is closer to 1, it indicates that the point has a higher chance of falling to the depth-k skyline user region. If the value is closer to −1, it indicates that the point is almost impossible to be in the depth-k skyline user region. The Fig. 11 neural network expected network function is shown as follows. The first part is the input layer. It is assumed that the input user is the z-th user. The input characteristic value is then $input(z)$. Meanwhile, since the input layer does not have any action, the output will directly be designated as input:

$$out_i^{(1)}(z) = input_i(z). \tag{4}$$

Additionally, in the hidden layer, the input will go through tangent sigmoid operation, and is written as:

$$f_i^{(2)}(z) = \sum_{j=1}^{r} w^1_{ij} out_j^{(1)}(z) + b_i, \tag{5}$$

$$out_i^{(2)}(z) = \frac{\exp(f_i^{(2)}(z)) - \exp(-f_i^{(2)}(z))}{\exp(f_i^{(2)}(z)) + \exp(-f_i^{(2)}(z))}, \tag{6}$$

where $f_i^2(z)$ indicates the result of the summing the outputs of input layer, w^1 is the weight value between input layer and hidden layer; b is the bias value; r is the node number of input layer; $\exp(\cdot)$ is the exponential function. The equation for the final output layer is

$$y(z) = \sum_{j=1}^{s} w^2{}_j out_j^{(2)}(z), \tag{7}$$

where w^2 is the weight value between hidden layer and output layer; s is the node number of hidden layer. After determining the target neural network architecture, the well-known back propagation algorithm [17] is now used to train the neural network. It is assumed that the training network required to attain target function is

$$Error(\mathbf{w}, z) = ½(y_d(z) - y(z)) = ½error(z)^2, \tag{8}$$

where w can be w^1, b, and w^2; $error(z)$ is the error value between ideal output $y_d(z)$ and network output $y(z)$ of user z.

$$w(z) = w(z-1) + \xi(-\frac{\partial Error(z-1)}{\partial w(z-1)}) \tag{9}$$

Through using error Eqs. (8) and (9), the updated equation for w^2 can be derived as follows:

$$w^2(z) = w^2(z-1) + \xi(-\frac{\partial Error(z-1)}{\partial w^2(z-1)}), \tag{10}$$

where

$$\frac{\partial Error(z-1)}{\partial w^2(z-1)} = \frac{\partial Error(z-1)}{\partial error(z-1)} \frac{\partial error(z-1)}{\partial y(z-1)} \frac{\partial y(z-1)}{\partial w^2(z-1)} = error(z-1) \cdot -1 \cdot out^{(2)}(z-1). \tag{11}$$

Hence,

$$w^2(z) = w^2(z-1) + \xi(error(z-1) \frac{\exp(f^{(2)}(z-1)) - \exp(-f^{(2)}(z-1))}{\exp(f^{(2)}(z-1)) + \exp(-f^{(2)}(z-1))}). \tag{12}$$

Next, by using error Eqs. (8) and (9), we can obtain the updated equation for b as follows,

$$b(z) = b(z-1) + \xi(-\frac{\partial Error(z-1)}{\partial b(z-1)}), \tag{13}$$

where

$$\frac{\partial Error(z-1)}{\partial b(z-1)} = \frac{\partial Error(z-1)}{\partial error(z-1)} \frac{\partial error(z-1)}{\partial y(z-1)} \frac{\partial y(z-1)}{\partial out^{(2)}(z-1)} \frac{\partial out^{(2)}(z-1)}{\partial f^{(2)}(z-1)} \frac{\partial f^{(2)}(z-1)}{\partial b(z-1)}$$
$$= error(z-1) \cdot -1 \cdot w^2(z-1) \frac{4}{(\exp(f^{(2)}(z-1)) + \exp(-f^{(2)}(z-1)))^2} \cdot 1. \quad (14)$$

Hence,

$$b(z) = b(z-1) + \xi(error(z-1) w^2(z-1) \frac{4}{(\exp(f^{(2)}(z-1)) + \exp(-f^{(2)}(z-1)))^2}). \quad (15)$$

Finally, according to the error Eqs. (8) and (9), the updated equation for w^1, can be derived as follows:

$$w^1(z) = w^1(z-1) + \xi(-\frac{\partial Error(z-1)}{\partial w^1(z-1)}), \quad (16)$$

where

$$\frac{\partial Error(z-1)}{\partial w^1(z-1)} = \frac{\partial Error(z-1)}{\partial error(z-1)} \frac{\partial error(z-1)}{\partial y(z-1)} \frac{\partial y(z-1)}{\partial out^{(2)}(z-1)} \frac{\partial out^{(2)}(z-1)}{\partial f^{(2)}(z-1)} \frac{\partial f^{(2)}(z-1)}{\partial w^1(z-1)}$$
$$= error(z-1) \cdot -1 \cdot w^2(z-1) \frac{4}{(\exp(f^{(2)}(z-1)) + \exp(-f^{(2)}(z-1)))^2} input(z-1). \quad (17)$$

Hence,

$$w^1(z) = w^1(z-1)$$
$$+\xi(error(z-1) w^2(z-1) \frac{4}{(\exp(f^{(2)}(z-1)) + \exp(-f^{(2)}(z-1)))^2} input(z-1)). \quad (18)$$

Through the above three formulas, the target neural network can be trained.

3.3 Finding Possible Depth-*k* Skyline Users in Clients by Using the Distributed Neural Filter

After the construction of the distributed neural filter, step 3 of the algorithm is to filter out target user A's possible depth-*k* skyline users in each client. The algorithm of this part is to mainly input the users' data of each client one at a time to the neural network; and calculate the possibility of becoming depth-*k* skyline user for each user via the three equations (4) to (6). If characteristic value of a user B is input to the neural network and the neural network output is greater than or equal to 0, user B is regarded as a possible depth-*k* skyline user; and the data of this user is fed back to the server. Conversely, if the output is less than 0, then user B is not a possible depth-*k* skyline user, and the data of this user is not fed back to the server. Note that using this method to check whether a user can possibly become a depth-*k* skyline user is much faster than the traditional method of using dominance check [6, 13]. This is because the processing time of neural networks will only grow linearly with the amount of data, while the processing time of dominate check will grow exponentially with the amount of data.

3.4 Finding Target User's Depth-*k* Users in the Server

The last step of this algorithm is to find the depth-*k* skyline users from the possible depth-*k* skyline users fed back by clients using the existing depth-*k* skyline algorithm and sorted-loop-checking algorithm [5]. The algorithm used is divided into two parts. First of all, the algorithm will sort all the users that were fed back based on individual user's summation of coordinate values. At this point, since a user with a smaller summation will never be dominated by a user with larger summation [2, 5], it is not necessary to check the domination of a larger summation user over a smaller summation user. Only the domination of a user with a smaller summation user over a user with a larger summation user has to be checked, hence a large amount of domination checks are eliminated. Besides, since smaller summation users might dominate more users, the checks are started from smaller summation users. This step also reduces the to-be-checked users drastically in the very beginning; and enhances the execution speed of depth-*k* skyline query.

After the fed back users have completed the sorting, user A with the smallest summation will conduct domination checks with all the users behind him one by one (as user B). After user A has completed all his domination checks, the user with the second smallest summation becomes user A and all the remaining users become a user B one at a time. Every time during the checking process, between the checking user A and to-be-checked user B, one of three relationships must exist. The three relationships and their corresponding handling methods are depicted as follows:

- A dominates B and the dominated time of B does not exceed *k*. In this case, the dominated time of B is added by 1.
- A dominates B and the dominated time of B exceeds *k*. In this case, B is eliminated and is not considered in future steps.
- A is incomparable with B. In this case, both the dominated time of A and B should not be updated.

Eventually, after the sorted-loop-checking algorithm has checked the largest summation user, the algorithm stops. Meanwhile, the users that have yet to be eliminated are the depth-*k* skyline users for target use. This completes the algorithm.

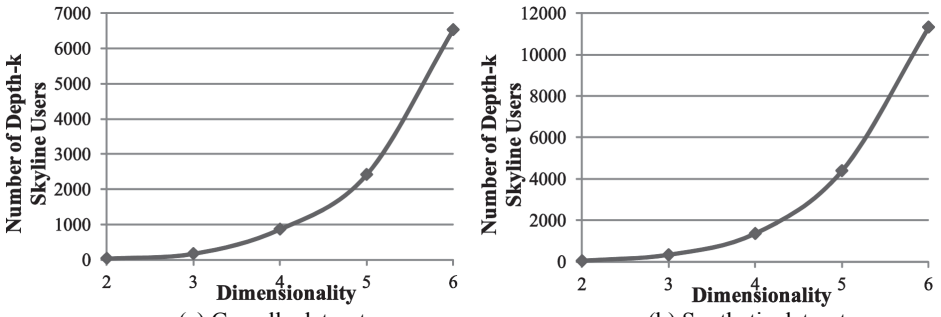
4. SIMULATIONS

In this section, we present a comprehensive set of computer simulations used to investigate the performance of the proposed algorithm, employing various combinations of features and evaluating their effectiveness. We adopted the Gowalla dataset [21, 26] and a synthetic dataset for the experiments. The Gowalla dataset is a benchmark commonly used in systems for the recommendation of tourist attractions, containing 196,591 users, 1,280,969 check-in points, and 6,442,890 check-ins. The preferences reported by each user in the Gowalla dataset are evaluate according to the type of check-in points. For example, if the check-in times at locations A, B, C, and D are 5, 1, 10, and 4, wherein A and D are museums, B is a park, and C is a shopping mall, then we can say the preferences of this user include knowledge-intensive attractions (5+4)/(5+1+10+4), natural att-

ractions 1/(5+1+10+4), and shopping attractions 10/(5+1+10+4). However, the types of check-in points (*e.g.*, restaurant, park, shop) are not specified in this dataset; therefore, we randomly generated location types. The number of users in the Gowalla dataset is generally too small to enable a comparison with actual social networks; we therefore employed a synthetic dataset to test the performance of the proposed algorithm in a more realistic setting. The proposed dataset includes 5,000,000 users in which the preferences of each user are given directly, without further evaluation. Moreover, as the time costs of distributed environments can be quite different under different cases, this work only discusses the sifting rate (*i.e.*, the ratio of data that transfer from the clients to the server) of simulations, which is the decisive factor that affect the time costs of simulations. Table 2 summarizes the parameter settings of this dataset with the default values written in boldface. Each performance curve in the figures represents an average of the experimental results obtained from 30 datasets generated using the benchmark. All of the experiments were performed on an Intel Core I7-4790 CPU at 3.60GHz with 4GB main memory, running Microsoft Windows 7. All of the programs were written in MATLAB®.

Table 2. Experimental parameters.

Parameter	Values
Dimensionality of datasets	2, 3, **4**, 5, 6
Depth, *k*	**3**, 5, 8, 10, 15, 20
Number of users in the Gowalla dataset	196,591
Number of users in the synthetic dataset	5,000,000



(a) Gowalla dataset. (b) Synthetic dataset.

Fig. 12. The number of depth-*k* skyline users in different datasets with different dimensionality.

4.1 The Numbers of Depth-*k* Skyline Users Found for Datasets of Various Dimensions and *k*

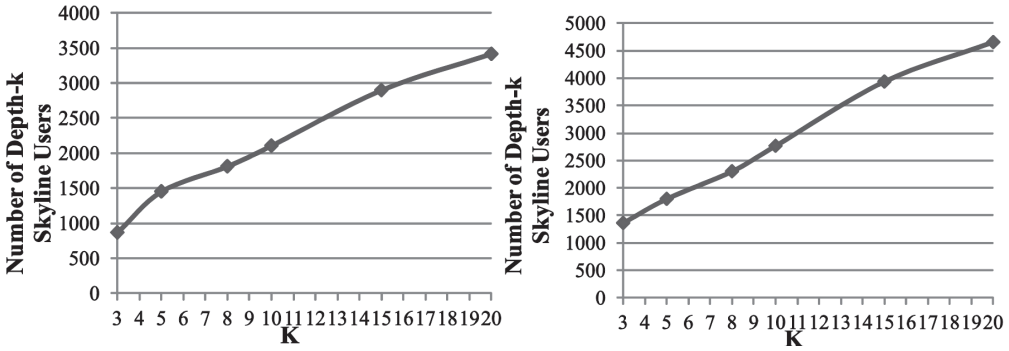
The numbers of depth-*k* skyline users found for datasets of various dimensions (two to six) are presented in Fig. 12, where Fig. 12 (a) presents the results of the Gowalla dataset and Fig. 12 (b) presents the results of the synthetic dataset. In these two figures, we can see that the number of depth-*k* skyline users in the Gowalla dataset (196,591) is much smaller than that of the synthetic dataset (5,000,000). As shown in Figs. 12 (a) and (b), the number of depth-*k* skyline users increases exponentially with the number of di-

mensions, because the number of skyline points increases exponentially with the number of dimensions [2, 24].

Fig. 13 presents the number of depth-*k* skyline users obtained from datasets of various *k* (3 to 20), where Fig. 13 (a) presents the results of the Gowalla dataset and Fig. 13 (b) presents the results of the synthetic dataset. Clearly, the number of depth-*k* skyline users increases linearly with *k*, due to fact that the number of users in each depth is nearly the same.

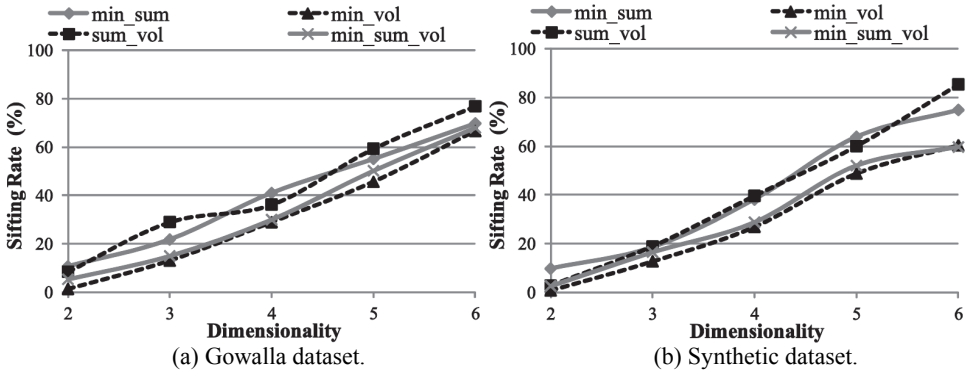
4.2 Comparisons of Feature Combinations in the Proposed Algorithm With Various Dimensions

In the following, we examine the performance of the proposed algorithm with various combinations of features under various numbers of dimensions within a distributed system of five clients where *k* is fixed at 3. In the Gowalla dataset, each client stores the information related to nearly 39,000 users. In the synthetic dataset, each client stores the information of 1,000,000 users. Without a loss of generality, we can assume that the data distributions of these objects in all clients are the same. We employed the average sifting in Fig. 14, where Fig. 14 (a) presents the results of the Gowalla dataset and Fig. 14 (b)



(a) Gowalla dataset. (b) Synthetic dataset.

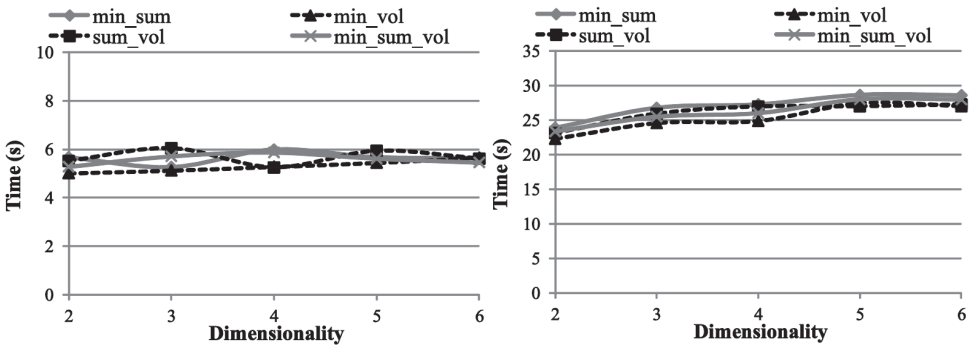
Fig. 13. The number of depth-*k* skyline users in different datasets with different *k*.



(a) Gowalla dataset. (b) Synthetic dataset.

Fig. 14. The sifting rates of the proposed algorithm under various data dimensions.

presents the results of the synthetic dataset. In most cases, fewer than 60% of the objects need to be transferred, which represents a saving of 40%. The only exception is found in the dataset with six dimensions, in which the number of depth-*k* skyline results increases exponentially with the number of dimensions. Thus, the number of possible depth-*k* skyline objects and the average sifting rate increase accordingly. In Figs. 14 (a) and (b), we can see that using only the combination of minimum and volume (*i.e.*, min_vol) in the algorithm results in the lowest sifting rate. This minimizes the number of users that need to be examined by the sorted-loop-checking algorithm, thereby enabling the algorithm to find all depth-*k* skyline users with similarities to a specific user in the social network more quickly. This can be explained by the fact that projecting users on the axis of minimum and the axis of volume enables a greater separation of depth-*k* skyline users and non-depth-*k* skyline users than could be achieved by projecting users on the axis of summation. Thus, the neural network is better able to differentiate between these two types of users. The second lowest sifting rate is achieved by using all three features in the neural network for the classification of depth-*k* skyline users and non-depth-*k* skyline users. The fact that this combination is a bit worse than that of the combination minimum and volume (*i.e.*, min_vol) can be attributed to two factors. First, feature summation provides no benefits with regard to the differentiation of users into depth-*k* skyline users and non-depth-*k* skyline, which has a direct influence on the resulting sifting rate. Second, a greater number of inputs in the neural network can reduce performance in the approximation of regions. Thus, the use of all three features can compromise the performance of the neural network, compared to using just two features. The other two combinations (min_sum and sum_vol) produce the highest sifting rates, due to the fact that at least one useful feature is absent and summation is taken into account.



(a) Gowalla dataset. (b) Synthetic dataset.

Fig. 15. The sifting rates of the proposed algorithm under different *k*.

4.3 Comparisons of the Proposed Algorithm with Various Feature Combinations and *k* Values

The performance of the proposed algorithm was evaluated using various *k* values in a distributed system with five clients, where *k* was varied between 3 and 20 and the number of dimensions was fixed at 4. Fig. 15 (a) presents the results of the Gowalla of

the figures, the sifting percentage of all combinations increased only slightly with k. This can be explained by the fact that the increase in the number of depth-k skyline users (with an increase in k) is far lower than the increase in the total number of users. As a result, the proposed algorithm is not required to sift through a larger number of users from the social network in order to find all of the depth-k skyline users. Fig. 15 shows that the combination min_vol achieves the lowest sifting percentage, due to the fact that these features are able to provide a more distinct differentiation between depth-k skyline users and non-depth-k skyline users than can be achieved using summation. The other combinations are unable to achieve sifting percentages lower than this due to the fact that they take the feature 'summation' into consideration.

5. CONCLUSIONS

This paper introduces a distributed neural filter to facilitate the search for users with similarities to a specific user in the social network. This new approach overcomes some of the shortcomings of the conventional depth-k skyline algorithm. The contributions of this work include the following: (1) We introduce the notion of using a skyline query to find friends with similar preferences in social networks; (2) We map the information of users within a feature space to facilitate the differentiation of the depth-k skyline users from non-depth-k skyline users; (3) We introduce the concept of using neural networks to accelerate computation; and (4) We provide a realistic simulation of features suitable for the differentiation of depth-k skyline users from non-depth-k skyline users. Our simulation results verify the effectiveness and efficiency of the proposed approach. In future work, we will continue to improve the proposed algorithm by using or developing other neural-network-related model, such as fuzzy neural networks or wavelet neural networks.

ACKNOWLEDGMENTS

This work was supported in part by the Ministry of Science and Technology of Taiwan, under Contracts MOST 106-2119-M-224-003 and MOST 106-2221-E-035-064.

REFERENCES

1. M. J. Barranco and L. Martínez, "A method for weighting multi-value features in content-based filtering," in *Proceedings of International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems*, 2010, pp. 411-418.
2. I. Bartolini, P. Ciaccia, and M. Patella, "SaLSa: Computing the skyline without scanning the whole sky," in *Proceedings of ACM International Conference on Information and Knowledge Management*, 2006, pp. 405-414.
3. T. Berka and M. Plößnig, "Designing recommender systems for tourism," in *Proceedings of ENTER*, 2004, pp. 26-28.
4. S. Borzsonyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedings of IEEE International Conference on Data Engineering*, 2001, pp. 235-254.

5. Y. C. Chen and C. Lee, "Depth-*k* skyline query for unquantifiable attributes in distributed Systems," in *Proceedings of International Conference on Artificial Intelligence and Soft Computing*, 2011, pp. 315-322.
6. Y. C. Chen and C. Lee, "A neural skyline filter for accelerating the skyline search algorithms," *Expert Systems*, Vol. 32, 2015, pp. 108-131.
7. S. M. Chiu, Y. C. Chen, H. Y. Su, and Y. L. Hsu, "Finding similar users in social networks by using the depth-*k* skyline query," in *Proceedings of IEEE Conference on Consumer Electronics*, 2015, pp. 162-163.
8. T. Emrich, M. Franzke, N. Mamoulis, M. Renz, and A. Züfle, "Geo-social skyline queries," in *Proceedings of International Conference on Database Systems for Advanced Applications*, 2014, pp. 77-91.
9. A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proceedings on ACM Special Interest Group on Management of Data*, 1984, pp. 47-57.
10. A. Hanze and S. Junmanee, "Travel recommendations in a mobile tourist information system," in *Proceedings of Information Systems and its Application*, 2005, pp. 86-99.
11. G. He, L. Chen, C. Zeng, Q. Zheng, and G. Zhou, "Probabilistic skyline queries on uncertain time series," *Neurocomputing*, Vol. 191, 2016, pp. 224-237.
12. T. Horozov, N. Narasimhan, and V. Vasudevan, "Using location for personalized POI recommendations in mobile," in *Proceedings of International Symposium on Applications on Internet*, 2006, pp. 1-6.
13. C. C. Hou, C. K. Chang, Y. C. Chen, H. Y. Su, and Y. L. Hsu, "Finding similar users in social networks by using the neural-based skyline region," in *Proceedings of International Conference on Artificial Intelligence for Engineering*, 2015, pp. 292-299.
14. H. P. Hsieh, C. T. Li, and S. D. Lin, "Exploiting large-scale check-in data to recommend time-sensitive routes," in *Proceedings of ACM SIGKDD International Workshop on Urban Computing*, 2012, pp. 55-62.
15. A. Jadhav and R. Sonar, "An integrated rule-based and case-based reasoning approach for selection of the software packages," *Information Systems*, Vol. 31, 2009, pp. 280-291.
16. C. Kalyvas, T. Tzouramanis, and Y. Manolopoulos, "Processing skyline queries in temporal databases," in *Proceedings of Symposium on Applied Computing*, 2017, pp. 893-899.
17. X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in Artificial Intelligence*, Vol. 2009, 2009.
18. T. M. N. Le, J. Cao, and Z. He, "Answering skyline queries on probabilistic data using the dominance of probabilistic skyline tuples," *Information Sciences*, Vol. 340-341, 2016, pp. 58-85.
19. M. J. Li, M. K. Ng, Y. M. Cheung, and J. Z. Huang, "Agglomerative fuzzy *k*-Means clustering algorithm with selection of number of clusters," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 20, 2008, pp. 1519-1534.
20. W. Liang, G. Lu, X. Ji, J. Li, and D. Yuan, "Difference factor' KNN collaborative filtering recommendation algorithm," in *Proceedings of International Conference on Advanced Data Mining and Applications*, 2014, pp. 175-184.

21. E. H. C. Lu, C. Y. Chen, and V. S. Tseng, "Personalized trip recommendation with multiple constraints by mining user check-in behaviors," in *Proceedings of International Conference on Advances in Geographic Information Systems*, 2012, pp. 209-218.
22. M. Okabe, M. Yanagisawa, H. Yamazaki, K. Kobayashi, A. Yoshioka, and T. Yamaguchi, "Organizational knowledge transfer of intelligence skill using ontologies and a rule-based system," *Practical Aspects of Knowledge Management*, Vol. 5345, 2008, pp. 207-218.
23. R. Pan, P. Dolog, and G. Xu, "KNN-based clustering for improving social recommender systems," in *Proceedings of International Workshop on Agents and Data Mining Interaction*, 2012, pp. 115-125.
24. D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2003, pp. 467-478.
25. Y. Park, J. K. Min, and K. Shim, "Efficient processing of skyline queries using MapReduce," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 29, 2017, pp. 1031-1044.
26. K. Pelechrinis and P. Krishnamurthy, "Location affiliation networks: Bonding social and spatial information," in *Proceedings of European Conference of Learning and Knowledge Discovery in Databases*, 2012, pp. 531-547.
27. Z. Peng, C. Wang, F. Tao, and L. Han, "SkyBoundary: An improved approach to member promotion in social networks," in *Proceedings of IEEE International Conference on Dependable*, *Autonomic and Secure Computing*, 2011, pp. 838-845.
28. Z. Peng and C. Wang, "Member promotion in social networks via skyline," *World Wide Web*, Vol. 17, 2014, pp. 457-492.
29. J. Salter and N. Antonopoulos, "CinemaScreen recommender agent: combining collaborative and content-based filtering," *IEEE Intelligent Systems*, 2006, pp. 35-41.
30. O. Shamir and N. Tishby, "Stability and model selection in *k*-means clustering," *Machine Learning*, Vol. 80, 2010, pp. 213-243.
31. G.S. J. Gong, "Joining case-based reasoning and item-based collaborative filtering in recommender systems," in *Proceedings of International Symposium on Electronic Commerce and Security*, Vol. 1, 2009, pp. 40-42.
32. K. C. Ting, R. P. Wang, Y. C. Chen, D. L. Yang, and H. M. Chen, "Finding *m*-similar users in social networks using the *m*-representative skyline query," *Information Discovery and Delivery*, Vol. 45, 2017, pp. 121-129.
33. J. S. Wang and Y. C. Chen, "A Hammerstein-Wiener recurrent neural network with universal approximation capability," in *Proceedings of IEEE International Conference on Systems*, *Man and Cybernetics*, 2008, pp. 1832-1837.
34. S. Zheng, A. Zaman, and Y. Morimoto, "Friend recommendation by using skyline query and location information," *Bulletin of Networking*, *Computing*, *Systems*, *and Software*, Vol. 5, 2016, pp. 68-72.

Yi-Chung Chen (陳奕中) received the B.S. and M.S. degrees in Electrical Engineering from National Cheng Kung University, Tainan, Taiwan, in 2007 and 2008, and the Ph.D. degrees in Department of Computer Science and Information Engineering from National Cheng Kung University, Tainan, Taiwan, in 2014. He joined the faculty of Department of Information Engineering and Computer Science, Feng Chia University in 2014 and participated in some projects related to AI techniques. He is currently an Assistant Professor in the Department of Industrial Engineering and Management, National Yunlin University of Science and Technology. His research interests include spatio-temporal databases, recommendation systems, social network analyses, artificial intelligences, and techniques of Industry 4.0.

Heng-Yi Su (蘇恆毅) was born in Taipei, Taiwan, in 1980. He received the M.S. and Ph.D. degrees in Electrical Engineering from National Taiwan University, Taipei, Taiwan in 2005 and 2014, respectively. Currently, he is an Associate Professor of Electrical Engineering at Feng Chia University. His research interests include techniques of artificial intelligence, and applications of PMUs to power system voltage stability monitoring and control.