**Method Article**

|  |  |
| --- | --- |
| Title |  |
| Authors |  |
| Affiliations |  |
| Corresponding Author’s email address |  |
| Keywords |  |
| Direct Submission or Co-Submission |  |

**Abstract**

**Specifications table**

**Method details**

There are two main approaches for recommendation such as content-based filtering (CBF) and collaborative filtering (CF). CF recommends an item to a user if her/his neighbors (other users like her/him) are interested in such item. One of popular algorithms in CF is *k*-nearest neighbors (KNN) algorithm. The essence of traditional KNN algorithm (Torres Júnior, 2004, pp. 16-18) is to find out *k* nearest neighbors of a regarded user (called active user) and then to recommend active user items that these neighbors may like. Let ***U*** = {*u*1, *u*2,…, *um*} be the set of users and let ***V*** = {*v*1, *v*2,…, *vn*} be the set of items. User-based rating matrix is the matrix in which rows indicate users and columns indicate items and each cell is a rating which a user gave to an item. In other words, each row in user-based rating matrix is a rating vector of a specified user. Rating vector of active user is called active user vector. As a convention, rating matrix implies user-based rating matrix if there is no additional explanation. In general, KNN algorithm includes two steps (Torres Júnior, 2004, pp. 17-18):

1. Find out *k* nearest neighbors of the active user by calculating similarities between active vector and other vectors. The more the similarity is, the nearer two users are. Given a threshold, *k* users whose similarities between them and active user are equal to or larger than a threshold are considered as *k* nearest neighbors of active user. Alternately, *k* users whose similarities between them and active user are in the top of *k* large values in descending ordering are *k* nearest neighbors of active user.
2. Compute (estimate) predictive values for missing ratings of active vector. The computation is based on ratings of *k* nearest neighbors and similarities calculated in step 1.

The main point of step 1 is to calculate similarities between active vector and other vectors. The most popular similarity measures are cosine and Pearson. Given two rating vectors *u*1 = (*r*11, *r*12,…, *r*1*n*) and *u*2 = (*r*21, *r*22,…, *r*2*n*) of user 1 and user 2, in which user 1 is considered as active user and some rating values *rij* can be missing (empty). Let *I*1 and *I*2 be set of indices of items that user 1 and user 2 rated, respectively. Let denote intersection set of *I*1 and *I*2 and let denotes union set of *I*1 and *I*2. All items whose indices belong to are rated by both user 1 and user 2. In other words, all items whose indices belong to co-exist in vectors *u*1 and *u*2. All items whose indices belong to are rated by user 1 or user 2. Notation |x| indicates absolute value of number, length of vector, length of geometric segment, or cardinality of set, which depends on context. Please pay attention to these denotations.

Let sim(*u*1, *u*2) denote the similarity of *u*1 and *u*2. For instance, the cosine measure of *u*1 and *u*2 is defined as follows (Torres Júnior, 2004, p. 17):

Where |*u*1| and |*u*2| are lengths of *u*1 and *u*2, respectively whereas *u*1•*u*2 is dot product (scalar product) of *u*1 and *u*2, respectively.

The main point of step 2 in KNN algorithm is to estimate missing values based on *k* nearest neighbors. Suppose NN algorithm finds out *k* neighbors of *u*1 from step 1, let *N* be set of indices of *k* neighbors of *u*1. Of course, we have |*N*| = *k*. A missing value *r*1*j* of *u*1 is estimated (computed or predicted) based on ratings of nearest neighbors and similarities according to step 2 of NN algorithm (Torres Júnior, 2004, p. 18).

Where and are mean values of *u*1 and *ui*, respectively.

Which items whose estimates of missing values are higher are recommended to active user. Such recommended items are also called relevant items.

Setting the predefined parameter *k* is not the drawback of KNN algorithm because the larger the parameter *k* is, the better the estimates are. The true drawback of KNN algorithm is that there are many cases in which *k* nearest neighbors does not rate the considered missing value *r*1*j*. In other words, determination of *k* nearest neighbors does not concern missing value *r*1*j* when *r*1*j* is only concerned in step 2. Consequently, there are not enough neighbors to calculate a good enough estimate for *r*1*j*. In order to alleviate this drawback, I proposed a so-called nearest neighbors (NN) algorithm in which determining neighbors of active user concerns missing value *r*1*j*. In other words, NN algorithm mixes or integrates step 1 and step 2 together and so there is no separation between step 1 and step 2. The two steps of NN algorithm are modified as follows:

1. For each given missing value *r*1*j*, similarities between active user and other users who rate on *r*1*j* are calculated. Note, nearest neighbors of active user are users who rate on *r*1*j* (item *j*).
2. Compute predictive values for *r*1*j*. The computation is based on ratings of nearest neighbors and similarities.

Thus, the main idea of NN algorithm is integration of neighbor determination and missing value estimation. Note, calculation of similarity measure and predicted values are kept intact. Following is the pseudo-code of NN algorithm.

There may be a question that why NN algorithm does not predefine the parameter *k* for reducing the number of nearest neighbors when the predefined *k* can speed up the algorithm. Indeed, setting *k* is reasonable for KNN algorithm but unreasonable for NN algorithm because NN algorithm determinates repeatedly neighbors of active user at each time that a missing value *r*1*j* is estimated whereas KNN algorithm determinates such neighbors only one time at step 1 before estimating all missing values at step 2. As a result, KNN is faster and NN is more accurate. It is easy to recognize that there is no “really nearest” neighbors in NN algorithm because all users different from active user contribute to the estimate of given missing value. It can be said that in NN, there is no nearest neighbors but neighbors are all users who rate on missing value *r*1*j* (item *j*). Therefore, restriction of *k* neighbors in NN algorithm is insignificant because the cost of making few arithmetic operators is not so expensive than the cost of sorting similarity measures in descending ordering or the cost of threshold comparison for taking *k* nearest neighbors. Moreover, restriction of *k* neighbors will decrease accuracy of NN algorithm. Anyhow, NN algorithm always browse all users for determining neighbors at each time that a missing value *r*1*j* is estimated. Recall that the main idea of NN algorithm is integration of neighbor determination and missing value estimation. Such idea can be stated in equivalent manner that the nearest neighbors of active user in NN algorithm are all users who rate on a given missing value. Thus, the concept of nearest neighbors in NN algorithm is different from the one in KNN algorithm.

**Acknowledgements**

**Declaration of interests**

**References**