**Method Article**

|  |  |
| --- | --- |
| Title | Nearest neighbors (NN) algorithm for collaborative filtering in recommendation application |
| Authors | Loc Nguyen1, Ali A. Amer2 |
| Affiliations | 1 Independent scholar, Vietnam  2 TAIZ University, Yemen |
| Corresponding Author’s email address | Loc Nguyen (ngphloc@yahoo.com)  Ali A. Amer (ali.amer@zu.ac.ae) |
| Keywords | *k*-nearest neighbors (KNN) algorithm, nearest neighbors (NN) algorithm, collaborative filtering, recommendation system |
| Co-Submission | The direct submission is the manuscript “Knowledge-Based Systems Enhancing Recommendation Systems Performance Using Highly-Effective Similarity Measures”. |

**Abstract**

The *k*-nearest neighbors (KNN) algorithm is a popular algorithm for collaborative filtering in recommendation application. The essence of KNN algorithm is to find out *k* nearest neighbors of an active user and then to recommend active user items that these nearest neighbors may like. The drawback of KNN algorithm is that determination of *k* nearest neighbors does not concern missing values, which in turn causes the weak consequence that there are not enough neighbors to calculate a good enough estimate for missing values. For alleviating this drawback, I proposed a so-called nearest neighbors (NN) algorithm in which determining neighbors of active user concerns missing values. In other words, given an item which active user does not rate, nearest neighbors of active users are all other users who rate on such item. The predefined parameter *k* is not set for KNN. As a result, KNN is faster but NN is more accurate.

**Specifications table**

|  |  |
| --- | --- |
| Subject Area | Data Science |
| More specific subject area | Data Science |
| Method name | Nearest neighbors (NN) algorithm |
| Name and reference of original method | *k*-nearest neighbors (KNN) algorithm |
| Resource availability | http://www.locnguyen.net/st/products/sim |

**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 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 the 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. Note, *r*1*j* and *r*2*j* are rating values that user *u*1 and user *u*2 rate on item *j*, 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 item *j* that active user (*u*1) does not rate (so that *r*1*j* is missing), similarities between active user and other users who rate on item *j* are calculated. Note, nearest neighbors of active user are users who rate on 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 like programming language *C* of NN algorithm.

|  |
| --- |
| Input: active user *u*1 = (*r*11, *r*12,…, *r*1*n*), item *j* that *u*1 does not rate (*r*1*j* is missing).  Output: the estimate *r*1*j* for item *j*.  *accum* = 0  *simTotal* = 0  For each user *u*2 = (*r*21, *r*22,…, *r*2*n*) in rating matrix  If *u*2 does not rate on item *j* then ignore.  Calculating the similarity measure sim(*u*1, *u*2) between *u*1 and *u*2.  *accum* += sim(*u*1, *u*2) \* (*r*2*j* – )  *simTotal* += |sim(*u*1, *u*2)|  End For  *r*1*j* = + *accum* / *simTotal* |

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” neighbor 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 item *j* (*r*1*j* is missing for active user). 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 the items that active user does not rate. Thus, the concept of nearest neighbors in NN algorithm is different from the one in KNN algorithm.

Given active user *u*1 who rates a few of items in the set of items ***V***, a recommendation application uses NN algorithm to estimate other items that *u*1 does not rate yet. Estimated ratings which are larger than a threshold (often average rating) are sorted in descending ordering and then, items whose estimated ratings are first in the ordering will be recommended to user *u*1. In other words, recommended items are the ones whose estimated ratings are both relevant and large enough. Note, an item is called relevant item if its rating value is larger than a threshold (often average rating). Let *C* be the number of recommended items, which is called recommendation count. A new problem is raised that how to restrict *C* to be large enough so that user can receive a plentiful enough number of favorite items, but it is still acceptable for algorithm speed. In general, *C* is neither too large nor too small. We proposed a method to determine *C* based on dataset with purpose that *C* will be more accurate and objective. The proposed method is dynamic and takes advantages of a so-called sparse-relevant ratio. This ratio is the ratio of the count of relevant ratings to the count of cells with note that the count of cells is product of user number and item number, which is size of rating matrix. Recall that a relevant rating is larger than average rating and the count of cells is sum of the count of rating values and the count of missing values. Following equation specifies sparse-relevant ratio denoted *sr*.

*sr* = the-count-of-relevant-ratings / (|***U***| \* |***V***|)

Note, |***U***| is the number of users and |***V***| is the number of items. We calculate recommendation count *C* dynamically according to both dataset and each rating vector *ui*. Let *C*(*ui*) be the recommendation count for user *i*, which means that NN algorithms will recommend at least *C*(*ui*) items to user *i*. Following equation specifies *C*(*ui*).

Where *T* is the number of items with note that every item included in *T* is rated by at least one user. Of course, *T* is smaller than or equal to the number of users |***U***|. Note, |*Ii*| is the number of items rated by user *i*. The quantity |*Ii*| is not redundant because real recommendation systems always recommend a user items that she/he do not either know or rate yet. If |*Ii*| is too much smaller than *T* (|*Ii*| << T), *C*(*ui*) can be calculated as follows:

In general, the purpose of the estimation equation is to establish *C* to be more accurate and objective. As a result, the tests for NN algorithm and similarity measures are more solid and fair. By improving fairness of the tests, Hudup dataset is divided into *k* folders (*k* = 5, 20, 50) and each folder includes training set and testing set. Training set and testing set in the same folder are disjoint sets. The ratio of testing set over the whole dataset depends on the testing parameter *r*. For instance, if *r* = 0.1, the testing set covers 10% the dataset, which means that the testing set has 10,000 = 10%\*100,000 ratings and of course the training set has 90,000 ratings. In our experimental design, parameter *r* has nine values 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, and 0.9. The smaller *r* is, the more accurate measures are because training set gets large if *r* gets small with note that NN algorithm is executed on training set. In general, the *k* parameter implies randomness of data and the *r* parameter implies spareness of data.

It is necessary to make an example about how to calculate *C* in detail. Recall that the dataset is divided into 5 folders and each folder has one training set and one testing set. In equation above, for each folder, *T* and sparse-relevant ratio *sr* are calculated on training set but |*Ii*| is determined on testing set, of course. Suppose one among 5 folders divided from Movielens has training set *d*1 and testing set *t*1. The number of users in *d*1 is 943 and the number of items in *d*1 is 1,584. Because every item in *d*1 is rated by at least one user, we have *T* = 1,584. Training set *d*1 has 50,000 rating values but only 27,712 rating values are relevant (larger than a threshold). So sparse-relevant ratio is *sr* = 27,712 / (943\*1584) ≈ 1.86%. Suppose it is necessary to make recommendation for user *u*1 (in testing set *t*1) who rates 23 items. Hence, recommendation count for user 12 is *C*(*u*12) = 1.86% \* (1,584 – 23) ≈ 29.

**Acknowledgements**

We express our deep gratitude to Ensha ALLAH who gave us sponsorship to do this research.

**Declaration of interests**

The authors declare that they have no known competing financial interests or personal relationships which have, or could be perceived to have, influenced the work reported in this article.

**References**

Torres Júnior, R. (2004). *Combining Collaborative and Content-based Filtering to Recommend Research Paper.* Universidade Federal do Rio Grande do Sul, Programa de Pós Graduação em Educação. Porto Alegre: Universidade Federal do Rio Grande do Sul. Retrieved from http://www.lume.ufrgs.br/bitstream/handle/10183/5887/000432990.pdf;sequence=1