**A New Approach for Collaborative Filtering**

**based on Distributed Bayesian Network**

**Abstract**

**1. Introduction**

**2. A new CF algorithm based on distributed Bayesian network**

**3. An enhancement – Binary distributed Bayesian network**

**4. Evaluation**

There are four measures to evaluate our CF algorithm: time (*T*), precise (*P*), recall (*R*) and usefulness (*U*). We use database *Movielens* [1] for evaluation. Database Movielens has two versions such as *100*K and *1*M: version *100*K with *1.88* MB capacity including *100,000* ratings of *943* users on *1682* movies, version *1*M with *23.4* MB capacity including *1,000,209* ratings of *6040* users on *3900* movies. The system setting includes: Processor Pentium(R) Dual-Core CPU E5700 @ 3.00GHz, RAM 2GB, Available RAM 1GB, Microsoft Windows 7 Ultimate 2009 32-bit, Java 7 HotSpot (TM) Client VM. Our CF method is compared to three other methods: *simple method* – simplest memory-based CF algorithm, c*osine method* – memory-based CF algorithm in which the cosine measure is used, *Pearson method* – memory-based CF algorithm in which the Pearson measure is used.

**4.1. Time measure**

The time measure *T* tells us how fast CF method responses to the requirement of system for recommendation task. This measure represents the speed of algorithm and so it is the time in seconds which algorithm runs over rating database (rating matrix). Let *D* be the rating matrix (rating database) and let *ui* *D* be the rating vector as each row in *D*. Let *ti* be the time in seconds that the algorithm runs for row *ui*. So the measure *T* is the sum of all *ti (s)* when iterating over *D*.

Our method is *…* times faster than simple, cosine and Pearson methods but *…* times slower than Green Fall method.

**Table 5.1**: Time evaluation (in seconds)

|  |  |  |
| --- | --- | --- |
|  | 100K | 1M |
| Simple |  |  |
| Cosine |  |  |
| Pearson |  |  |
| Green Fall |  |  |
| Our method |  |  |

**Figure 5.1**: Time evaluation (in seconds)

**4.2. Precise measure**

Basically, the precise measure (*P*) is the inverse of normalized mean of error (*E*). In consequences, the error *E* is calculated as the mean of deviation. Let *ui* *D* and *vi* be the actual rating vector and the rating vector recommended by our CF algorithm, respectively. Each *ui* actually is a row in rating matrix. Each *ui* has respective *vi*. Suppose *ui =* (*ri1, ri2,…, rin*) and *vi =* (*r’i1, r’i2,…, r’in*), the basic idea is modeled by following formulation:

Where |*D*| is the size of database and *error*(*ui, vi*) is the deviation between *ui* and *vi*.

We recognized that the error *E* is the ratio of total deviation between predictive rating values and actual rating values to the size of database, and known as the number of rows of rating matrix.

The rating matrix *D* is browsed row by row, each row *ui* is a rating vector. The rating vector *vi* is derived from *ui* by removing randomly some values from *ui*. For example, if *ui* = (*1, 2,* ***4****,* ***5***) then the derived *vi* = (*1, 5,* ***?****,* ***?***), the third component and fourth component are removed. Missing values (?) in vector *vi* are predicted by CF method, for example *vi* = (*1, 5,* ***1****,* ***1***). After that the error is the deviation between *ui* and *vi*. Let *k* be the number of missing values, in this example, *k = 2*.

Suppose the range of rating value is {*min…max*}, so *min* and *max* are the possible minimum and maximum of each rating. In this example, *min = 1* and *max = 5*. The normalized error is computed as below:

The sum of errors is computed as the total error over database: . The mean of error: . The normalized mean of error: .It is written again:

So error *E* isthe normalized mean of error. The less this error is, the more precise CF method is. So precise measure is the inverse of normalized mean of error in percentage:

*Precise* (*P*) = (*1 –* *normalized\_mean\_of\_error*) *\* 100*

The precision of our method is a little bit less than simple, cosine and Pearson methods but higher than Green Fall method. Moreover it can tolerate sparse matrix. Following is the line chart of precise evaluation:

**Table 5.2**: Precise evaluation (%)

|  |  |  |
| --- | --- | --- |
|  | 100K | 1M |
| Simple |  |  |
| Cosine |  |  |
| Pearson |  |  |
| Green Fall |  |  |
| Our method |  |  |

**Figure 5.2**: Precise evaluation (%)

**4.3. Recall measure**

The recall measure tells us what percentage of missing value items which CF algorithm can predict over dataset. For example, there are *4* items whose values are missing and the CF algorithm predicts missing values of *2* items, so the recall is *2/4 \* 100% = 50%.* It also indicates the loss ratio. The more recall measure is, the less loss ratio is:

*Recall (R) =* (*Predicted value items / Missing value items*) *\* 100*

The recall calculated by the ratio of the number of predicted value items to the number of missing value items is not appropriate to recommendation context because predicted value items are not really meaningful in case that what user requires is an interesting item for her/him. This measure is merely used to evaluate the ability of prediction. So it is not as important as other measures.

**Table 5.3**: Recall evaluation (%)

|  |  |  |
| --- | --- | --- |
|  | 100K | 1M |
| Simple |  |  |
| Cosine |  |  |
| Pearson |  |  |
| Green Fall |  |  |
| Our method |  |  |

**Figure 5.3**: Recall evaluation (%)

We propose another measure so-called *recommend recall* which is the ratio of the number of recommended items to the number of total potential items. For example, there are *100* items in dataset which can recommend to users but algorithm only gives *90* items to users. The number total potential items are 100 and the number of recommended items is *90*, so the recommend recall is *90%*.

*Recommend Recall (R2) =* (*Recommended items / Potential items*) *\* 100*

Comment that all memory-based CF (s) get 100% recommend recall because they computed directly on dataset and don’t build up any model and there is no loss of items. Following is the line chart of recommend recall (*R2*) evaluation:

**Table 5.4**: Recommend recall evaluation (%)

|  |  |  |
| --- | --- | --- |
|  | 100K | 1M |
| Simple |  |  |
| Cosine |  |  |
| Pearson |  |  |
| Green Fall |  |  |
| Our method |  |  |

**Figure 5.4**: Recommend recall evaluation (%)

**4.4. Usefulness measure**

The precise measure is based on statistical data, it doesn’t reflect how much user really like recommended items and so it is very rigid. There is in case that some items are given to user and they are predicted with high consistency and high precision but they are not interesting items for user. The usefulness indicator measures the level of user interest on such recommended items. In other words, it tells us how useful such recommended items are and so this measure is the most important one. Usefulness measure (*U*) is the ratio of the number of interesting items to the number of best items. Best item is defined as the item which algorithm recommends to user and its rating value is highest. Interesting item is the item on which user rates with highest value.

*Usefulness* (*U*) *=* (*number of interesting items / number of best items*) *\* 100*

For example, algorithm gives user *item1, item3, item5* whose values are *1, 5, 5*, respectively and user rates on *item2, item3, item4* with values *2, 5, 3,* respectively. Best items are *item3* and *item5*. Interesting item is *item3*. So the usefulness measure is *1 / 2 = 50%*. Our method gains a highest usefulness because it discovers the patterns of user interests based on Bayesian inference mechanism via her/his ratings or purchases as evidences . Moreover it gets more precise than Green Fall method. So all recommended items conform to user interests. Following is the line chart of usefulness evaluation:

**Table 5.5**: Usefulness evaluation (%)

|  |  |  |
| --- | --- | --- |
|  | 100K | 1M |
| Simple |  |  |
| Cosine |  |  |
| Pearson |  |  |
| Green Fall |  |  |
| Our method |  |  |

**Figure 5.5**: Usefulness evaluation (%)

**5. Conclusion**

**Reference**

1. [Su, Khoshgoftaar 2009]. Xiaoyuan Su and Taghi M. Khoshgoftaar. A Survey of Collaborative Filtering Techniques. Hindawi Publishing Corporation, Advances in Artificial Intelligence, Volume 2009, Article ID 421425, 19 pages, doi:10.1155/2009/421425.
2. [Movielens dataset 2011]. Home page is http://www.movielens.org. Download dataset from http://www.grouplens.org/node/12.
3. Richard E. Neapolitan. Learning Bayesian Networks. Northeastern Illinois University Chicago, Illinois 2003.
4. [Ungar, Foster 1998]. L. H. Ungar and D. P. Foster. Clustering Methods for Collaborative Filtering. Proceedings of the Workshop on Recommendation Systems, AAAI Press, 1998.
5. [Han, Kamber 2006]. Jiawei Han and Michelline Kamber. Data Mining: Concepts and Techniques. Second Edition. © 2006 by Elsevier Inc.
6. [Breese, Heckerman, Kadie 1998]. John S. Breese, David Heckerman and Carl Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. Technical Report MSR-TR-1998. Microsoft Research, Microsoft Corporation, One Microsoft Way, Redmon, WA 98052.
7. [Heckerman, Chickering, Meek, Rougnthwaite 2000]. David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rougnthwaite and Carl Kadie. Dependecy Networks for Inference, Collaborative Filtering and Data Visualization. Journal of Machine Learning Research 1 (2000) 49-75.
8. [Hofmann 2004]. Thomas Hofmann. Latent Semantic Models for Collaborative Filtering. ACM Transactionson Information Systems, Vol.22, No.1, January 2004, Pages 89-115.
9. [Shani, Heckerman, Brafman 2005]. G. Shani, D. Heckerman, and R. I. Brafman. An MDP-based Recommender System. Journal of Machine Learning Research, vol. 6, pp. 1265-1295, 2005.
10. [Rennie 2004]. Jason D. M. Rennie (jrennie@csail.mit.edu). A Short Tutorial on Using Expectation-Maximization with Mixture Models. MIT University, March 3, 2004

We propose another measure so-called *recommend recall* which is the ratio of the number of recommended items to the number of total potential items. For example, there are *100* items in dataset which can recommend to users but algorithm only gives *90* items to users. The number total potential items is 100 and the number of recommended items is *90*, so the recommend recall is *90%*.

*Recommend Recall (R2) =* (*Recommended items / Potential items*) *\* 100*

Comment that all memory-based CF (s) get 100% recommend recall because they computed directly on dataset and don’t build up any model and there is no loss of items. Following is the line chart of recommend recall (*R2*) evaluation: