SUGGEST[*]
Top-N Recommendation Engine
Version 1.0

George Karypis

University of Minnesota, Department of Computer Science / Army HPC Research Center
Minneapolis, MN 55455

karypis@cs.umn.edu

Last updated on November 7, 2000 at 3:05pm

Contents

1 Overview

SUGGEST is a *Top-N* recommendation engine that implements a variety of recommendation algorithms. *Top-N* recommender systems is a personalized information filtering technology, used to identify a set of N items that will be of interest to a certain user. In recent years, recommender systems have been used in a number of different applications such as to recommend products a customer will most likely buy; recommend movies, TV programs, or music a user will find enjoyable; identify web-pages that will be of interest; or even suggest alternate ways of searching for information.

The algorithms in SUGGEST are based on collaborative filtering (CF) that is probably the most successful and widely used framework for building recommender systems. Collaborative filtering-based *Top-N* recommendation algorithms derive their recommendations by analyzing information on what people have done in the past to identify pieces of information (*e.g.*, web-pages, movies, products, *etc*.) that will be of value to a particular user. One of the advantages of collaborative filtering recommender systems is that they are content independent and can be used in many diverse areas.

SUGGEST implements two classes of CF-based *Top-N* recommendation algorithms, called *user-based* and *item-based*. User-based algorithms rely on the fact that each person belongs to a larger group of similarly-behaving individuals. Consequently, items (*i.e.*, products) frequently liked by the various members of the group can be used to form the basis of the recommended items. On the other hand, item-based algorithms rely on the fact that a person will more likely like items that are similar or related to the items that he/she has liked in the past. In both classes of algorithms, information about what the different individuals have done in the past are used to discover either the groups of like-minded individuals (as in the case of user-based algorithms), or the similar items (as in the case of the item-based algorithms). The details of the different algorithms are described in [2, 1].

Our primary goals in designing SUGGEST was to develop a *Top-N* recommendation engine that satisfied the following requirements:

- Produce high-quality recommendations.
- Achieve low recommendation latency.
- Scale to large datasets.

The item-based *Top-N* recommendation algorithms provided by SUGGEST meet all three of these design objectives. The experiments reported in [1], have shown that SUGGEST's item-based *Top-N* recommendation algorithms produce recommendations that are often substantially better than those produced by the user-based algorithms, can compute each set of *Top-N* recommendations in less than 50us on a 366MHz Pentium II, and can scale to large datasets.

The rest of this manual is organized as follows. Section 2 provides a brief introduction on how to use the different routines in SUGGEST. Section 3 describes the calling sequence of SUGGEST's routines. Section 4 provides some experiments comparing the performance of SUGGEST's recommendation algorithm. Finally, Section 5 describes SUGGEST's hardware requirements and provides contact information.

2 Using SUGGEST

The different recommendation algorithms implemented in SUGGEST are provided in the form of a library that can be directly linked into your application. SUGGEST's library provides the following three routines for accessing the recommendation engine.

- **SUGGEST_Init()**
- **SUGGEST_TopN()**
- **SUGGEST_Clean()**

SUGGEST_Init() is used to initialize the recommendation engine and set up the internal data structures to be used during the recommendation. The input to SUGGEST_Init() is the set of historical user-item transactions that depending on the application can represent different things. For instance, in e-commerce type of applications, the historical transactions will correspond to the set of products the different customers have purchased in the past. Similarly, in the case of web-page recommendations, the transactions will correspond to the set of pages the various users have accessed the past. The SUGGEST_Init() routine must be called before any recommendations can be computed, and returns a pointer to the internal data structures (referred to as the *RcmdHandle*) that must be passed as a parameter to SUGGEST_TopN() and SUGGEST_Clean().

The actual recommendations are computed using the SUGGEST_TopN() routine. This routine takes as input the *RcmdHandle* returned by SUGGEST_Init() and the user's current *basket* of items, and returns the *Top-N* recommended items. Again, depending on the application the basket can represent different things. In e-commerce applications the basket can represent either the products that have been purchased by the user or the products that have been inserted in his/her *shopping basket*. On the other hand, in web-page recommendation applications, the basket can correspond to the set of pages that have already been accessed by the user. The SUGGEST_TopN() routine can be called as many times as needed, once for each of the users for which we need to compute their *Top-N* recommendations.

Once all the required recommendations have been computed, the SUGGEST_Clean() routine must be called to free the internal data-structures created by SUGGEST_Init(). The input to this routine is the *RcmdHandle*, and once it has been called, the handle becomes invalid. In order to compute any additional recommendations, the engine needs to be initialized again.

Finally, SUGGEST provides the SUGGEST_EstimateAlpha() routine that is used to estimate the optimal value for one of the parameters for the item-based *Top-N* recommendation algorithm.

3 Calling Sequence of the Routines in SUGGEST

The calling sequences of the routines provided by SUGGEST are described in the rest of this section.

int ***SUGGEST_Init** (int nusers, int nitems, int ntrans, int *userid, int *itemid, int RType, int NNbr, float Alpha)

Description

It is used to initialize the *Top-N* recommendation engine.

Parameters

nusers The total number of distinct users in the database of historical transactions.

nitems The total number of distinct items available in the database. This is essentially the universe of items that the recommendation engine can recommend.

ntrans The total number of historical user-item transactions that will be used to base the recommendations upon.

userid, itemid

The actual user-item transactions. These are two integer arrays of size *ntrans* such that each (*userid*[i], *itemid*[i]) pair represents the transaction that the user stored in *userid*[i] has purchased/accessed the item stored in *itemid*[i]. Note that the users and items must be numbered consecutively from 0 to (*nusers*-1) and from 0 to (*nitems* -1), respectively. That is, the entries in the *userid*[] array can only take values from 0 to (*nusers*-1), and the entries in the *itemid*[] array can only take values from 0 to (*nitems*-1).

RType Used to select the type of the *Top-N* recommendation algorithm that should be used. *RType* can take the following values:

1 Item-based *Top-N* recommendation algorithm with cosine-based similarity function (*ItemCos*).

2 Item-based *Top-N* recommendation algorithm with probability-based similarity function (*Item-Prob*). This algorithms tends to outperform the rest.

3 User-based *Top-N* recommendation algorithm with cosine-based similarity function (*UserCos*).

NNbr Used to indicate the size of the neighborhood to be used during recommendation. In the case of item-based recommendation algorithms this corresponds to item-neighborhoods, whereas in the case of user-based algorithms it corresponds to user neighborhoods. Our experiments have shown that a value in the range of 15-30 is sufficient for item-based algorithms, and a value in the range of 40-80 is sufficient for user-based algorithms. Note that selecting very small values for *NNbr* can potentially lead to poor recommendations, and selecting large values for *NNbr* can reduce the recommendation speed and, in the case of user-based algorithms, it can also reduce the quality of the recommendations.

Alpha It is used only when *RType* = 2, and specifies how the frequently occurring items will be de-emphasized during the computation of the *Top-N* recommendations. *Alpha* ranges from (0.0, 1.0], and our experimental evaluation in [1] has shown that a value in the range of [0.3, 0.5] provides reasonably good performance. In general, a smaller value of *Alpha* should be preferred when the historical information is quite sparse, and a larger value should be used in denser datasets. However, the optimal value is dataset dependent, and some experimentation may be required. Note that the right value for *Alpha* can dramatically improve the performance of the algorithm.

SUGGEST provides the SUGGEST_EstimateAlpha() routine that can be used to automatically estimate the optimal value for *Alpha*, using a cross-validation approach.

Returned Value

The routine returns a pointer to the internal data structures that were setup by the recommendation engine. We will refer to this pointer as the *RcmdHandle* and it must be passed to both the SUGGEST_TopN() and SUGGEST_Clean() routines.

int **SUGGEST_TopN** (int *RcmdHandle, int bsize, int *itemids, int NRcmd, int *rcmds)

Description

It is used to compute the *Top-N* recommendations based on the current set of items in the user's basket.

Parameters

RcmdHandle
This is the pointer returned by the SUGGEST_Init() routine.

bsize This is the number of items in the user's current basket.

itemids This is an array of size *bsize* that stores the items-ids in the user's basket. Note that each of these item-ids should be in the range of $[0, nitems - 1]$, where *nitems* is the total number of distinct items specified in the SUGGEST_Init() routine, and that the item numbering should be consistent with the item numbering used in SUGGEST_Init().

NRcmd This is the number of recommended items that are required; *i.e.*, it is the value of N in the *Top-N* recommendation algorithm.

rcmds This is an array of size *NRcmd* that will store the item-ids of the recommended items.

Returned Value

The routine returns the actual number of items that were recommended. Note that this number can be smaller than the requested number of recommended items *NRcmd*. If this is the case, the entries in the *rcmds* array corresponding to the missing items will be undefined.

Also, if any of the values in the *itemids* array is invalid (*i.e.*, not in the $[0, nitems-1]$ range), SUGGEST_TopN() returns -1.

void **SUGGEST_Clean** (int *RcmdHandle)

Description

It is used to free the memory allocated for the internal data-structures of the *Top-N* recommendation engine.

Parameters

RcmdHandle

This is the pointer returned by the SUGGEST_Init() routine.

float **SUGGEST_EstimateAlpha** (int nusers, int nitems, int ntrans, int *userid, int *itemid, int NNbr, int NRcmd)

Description

It is used to estimate the optimal value of *Alpha* for the probability-based *Top-N* recommendation algorithm (*i.e.*, *RType*=2).

Parameters

nusers, nitems, ntrans, userid, itemid, NNbr
These parameters are identical to the corresponding parameters of the SUGGEST_Init() routine.

NRcmd This parameter is identical to the corresponding parameter of the SUGGEST_TopN() routine.

Returned Value

The routine returns the estimated optimal value for *Alpha*.

Notes

The algorithm used by SUGGEST_EstimateAlpha() to estimate the optimal value of *Alpha* is based on the cross-validation paradigm. As a result, for accurate estimations, its various parameters should be identical to the corresponding parameters passed in SUGGEST_Init() and SUGGEST_TopN(). The optimal value of *Alpha* does not change dramatically as the set of historical transactions are updated. For this reason, the value of *Alpha* needs to be only estimated from scratch, when the historical transactions change substantially.

4 Experimental Comparison of SUGGEST's Recommendation Algorithms

In this section we present some experimental results comparing the performance of the different *Top-N* recommendation algorithms implemented in SUGGEST. A more detailed experimental evaluation can be found in [1]. All experiments were performed on a Pentium II based workstation running at 366MHz, 256MBytes of memory, and Linux-based operating system.

Data Sets SUGGEST's performance was evaluated using five different datasets whose characteristics are shown in Table 1. For each dataset, the columns labeled "*nusers*", "*nitems*", and "*ntrans*" correspond to the number of users, number of items, and total number of transactions, respectively.

Name	*nusers*	*nitems*	*ntrans*
ecommerce	6667	17491	91222
catalog	50918	39080	435524
ccard	42629	68793	398619
skills	4374	2125	82612
movielens	943	1682	100000

Table 1: The characteristics of the various datasets used in evaluating the *Top-N* recommendation algorithms.

The *ecommerce* dataset corresponds to web-based purchasing transactions of an e-commerce site. The *catalog* dataset corresponds to the catalog purchasing transactions of a major mail-order catalog retailer. The *ccard* dataset corresponds to the store-branded credit card purchasing transactions of a major department store. The *skills* dataset corresponds to the IT-related skills that are present in the resumes of various individuals and were obtained from a major online job portal. Finally, the *movielens* dataset corresponds to movie ratings and were obtained from the *MovieLens* research project. Note that in our experiments, we ignored the actual ratings in the *movielens* dataset.

Experimental Design and Metrics In order to evaluate the quality of the *Top-N* recommendations we split each of the datasets into a *training* and *test* set, by randomly selecting one of the transactions for each user to be part of the test set, and used the remaining transactions for training. The transactions in the training set were used to initialize SUGGEST's recommendation engine (*i.e.*, were the input to SUGGEST_Init()).

The quality was measured by looking at the number of *hits*; *i.e.*, the number of items in the test set that where also present in the *Top-N* recommended items returned for each user. In particular, if n is the total number of users, we computed the *recall* of the recommended system as:

$$\textit{recall} = \frac{\textit{Number of hits}}{n}.$$

A recall value of 1.0 indicates that the recommendation algorithm was able to always recommend the hidden item, whereas a recall value of 0.0 indicates that the recommendation algorithm was not able to recommend any of the hidden items.

In order to ensure that our results were statistically accurate, for each of the experiments we performed ten different runs, each time using a different random partitioning into training and test. The results reported in the rest of this section are the averages over these ten trials. Finally, in all of experiments we used $N = 10$, as the number of items top be recommended by the *Top-N* recommendation algorithms.

Comparison of the Recommendation Algorithm The performance of the different *Top-N* recommendation algorithms is shown in Figure 1. This figure includes four different set of results. The results labeled "User" correspond to the user-based results obtained by using *RType=3*. The results labeled "ItemCos" correspond to the cosine-based results obtained by using *RType=1*. The results labeled "ItemProb-a=0.5" correspond to the probability-based algorithm in which *Alpha* was set to 0.5. The results labeled "ItemProb-a=Opt" correspond to the probability-based algorithm that uses for each dataset the value of *Alpha* that achieved the highest performance [1]. These results were obtained by using *RType = 2*. Finally, Figure 1 also includes the *Top-N* recommendation quality achieve by the naive algorithm, labeled "Frequent", that recommends the N most frequent items not already present in the active user's set of items. The item-based recommendations were obtained using item-neighborhoods of size 20 (*i.e.*, *NNbr* = 20), and the user-based recommendations were obtained using user-neighborhoods of size 50 (*i.e.*, *NNbr*=50).

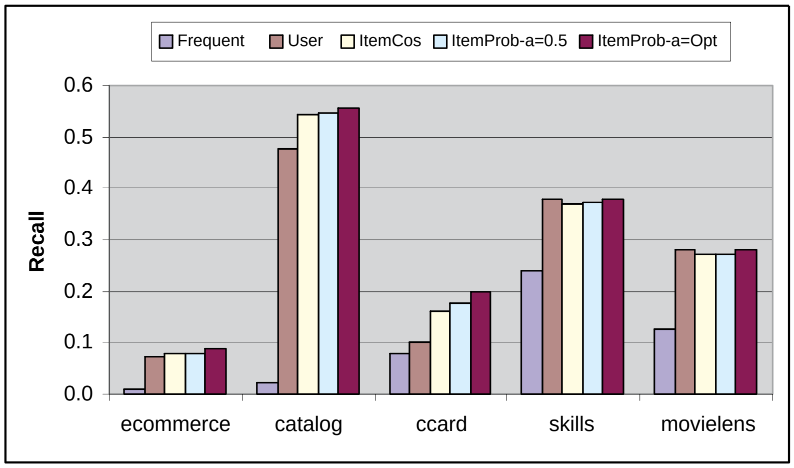

Figure 1: The quality of the recommendations obtained by the naive, the item-based and the user-based recommendation algorithms.

From the results in Figure 1 we can see that both the "ItemCos" and the "ItemProb-a=0.5" algorithms outperform the user-based algorithm in three out of the five datasets, whereas "ItemProb-a=Opt" outperforms the user-based scheme in all five datasets. It is interesting to note that the first two item-based algorithms perform substantially better for the first three datasets and only marginally worse for the remaining two. In fact, the average improvement achieved over all five datasets is a significant 15.7% and 18.8% for "ItemCos" and "ItemProb-a=0.5", respectively. The item-based algorithm that uses the optimal values of *Alpha* performs even better, achieving an average improvement of 27%. Also note that both the user- and item-based algorithms produce recommendations whose quality is substantially better than the recommendations produced by the naive "Frequent" algorithm.

One of the advantages of the item-based algorithm is that it has much smaller computational requirements than the user-based *Top-N* recommendation algorithm. Table 2 shows the amount of time required by the two algorithms to compute the *Top-N* recommendations for each one of the five datasets. The columns labeled "RcmdTime" shows the amount of time required to compute all the recommendations for each one of the dataset, and the columns labeled "RcmdRate" shows the rate at which the *Top-N* recommendations were computed in terms of *recommendations/second*. All the times in Table 2 are in seconds.

As we can see from these results, the recommendation rates achieved by the item-based algorithm are 12 to 28 times higher than those achieved by the user-based algorithm. If we add the various "RcmdTime" for all five data sets we can see that the overall recommendation rate for the item-based algorithm is 19579 recommendations/second compared to only 1157 recommendations/second achieved by the user-based algorithm. This translates to one recommendation every 50us for the item-based algorithm, versus 864us for the user-based algorithm.

In summary, the item-based *Top-N* recommendation algorithms improve the recommendations produced by the user-based algorithms by up to 27% in terms of recommendation accuracy, and it is up to 28 times faster.

[1]These values of *Alpha* where determined by performing a set of experiments in which we varied *Alpha* in 0.1 increments.

	User-based		Item-based	
Name	RcmdTime	RcmdRate	RcmdTime	RcmdRate
ecommerce	4.05	1646	0.33	20203
catalog	27.20	1848	2.20	22817
ccard	50.04	851	2.43	17542
skills	6.50	672	0.23	19017
movielens	3.38	278	0.20	4715

Table 2: The computational requirements for computing the *Top-N* recommendations for both the user- and item-based algorithms.

5 Hardware Requirements, and Contact Information

Currently, SUGGEST is an *in-memory* recommendation engine. The memory requirements of SUGGEST's internal data-structures (*i.e.*, where *RcmdHandle* points) depend on whether or not the item- or user-based algorithms are used. In the case of the item-based algorithms, the internal data-structures require approximately

$$2 * (nitems * NNbr) + 4 * nitems \quad \text{words},$$

and in the case of the user-based algorithms, they require approximately

$$4(nrows + ncols + ntrans) \quad \text{words}.$$

Since in most typical applications, *nitems* will be much smaller than *ntrans*, the item-based *Top-N* recommendation algorithms require the least amount of memory.

SUGGEST is written in ANSI C and has been extensively tested under Linux and Solaris. At this point SUGGEST's distribution is only in a binary format. However, the source can be made available to government and educational institutions for research and educational purposes. A simple driver program along with a small dataset is included for testing. Please look at the driver program for an example of how to use SUGGEST.

If you find any problems or have any questions on how to use SUGGEST, please contact George Karypis via email at *karypis@cs.umn.edu*.

References

[1] George Karypis. Experimental evaluation of item-based top-*n* recommendation algorithms. Technical Report TR-00-046, Department of Computer Science, University of Minnesota, Minneapolis, 2000. Available on the WWW at URL *http://www.cs.umn.edu/˜karypis*.

[2] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of ACM E-Commerce*, 2000.