Full Text Search Engine as Scalable k-Nearest Neighbor Recommendation System

Ján Suchal and Pavol Návrat

Slovak University of Technology
Faculty of Informatics and Information Technologies
Ilkovičova 3, 842 16 Bratislava, Slovakia
suchal@fiit.stuba.sk, navrat@fiit.stuba.sk

Abstract. In this paper we present a method that allows us to use a generic full text engine as a k-nearest neighbor-based recommendation system. Experiments on two real world datasets show that accuracy of recommendations yielded by such system are comparable to existing spreading activation recommendation techniques. Furthermore, our approach maintains linear scalability relative to dataset size. We also analyze scalability and quality properties of our proposed method for different parameters on two open-source full text engines (MySQL and SphinxSearch) used as recommendation engine back ends.

Keywords: full text search, recommendation systems.

1 Introduction

Recommendation systems are becoming very common in many domains, but their results can be mostly seen as recommendations in online shops (e.g. Amazon.com), news (e.g. Google news) or social networks sites (e.g. Facebook). In current era of web applications, to create a good and also a scalable recommendation system is not an easy task. Typically very specialized systems are developed to deal with the problem of high quality recommendations on large datasets.

In this paper we present a new method that allows to use a generic and generally available full text engine back ends (MySQL and SphinxSearch) to generate recommendations based on a k-nearest neighborhood approach. First we explain how data needs to be preprocessed into virtual documents to exploit full text engine capabilities for recommendation generation and present our k-nearest neighbor algorithm. Next, we discuss how neighborhood size affects precision of such recommendations. We show that long non-scalable queries are not present in real-world datasets by exploiting power-law distributions in datasets and by showing negligible precision gains from long queries. We also show that our method yields recommendations with comparable precision as spreading activation techniques normally used in recommendation systems. Furthermore we address scalability of our method showing that it yields linear scalability relative to

M. Bramer (Ed.): IFIP AI 2010, IFIP AICT 331, pp. 165–173, 2010.

dataset size. For our experiments two medium-sized real world datasets have been used.

2 Full Text Recommendation Algorithm

In order to be able to use a generic full text engine as recommendation system we need to transform data into special structures - virtual documents. We create two classes of such virtual documents - per-user and per-item virtual documents. Suppose we have data available from an e-shop consisting of records who bought which items. A per-user virtual document consists of user identifier and concatenation of item identifiers that this user is connected to. Per-item virtual document contains item identifier and space separated user ids. Tables 1 and 2 show examples of such virtual documents.

Table 1. Per-item virtual documents

Item identifier	*Users*
item1	user1 user2 user3
item2	user1 user2 user4
item3	user4

Table 2. Per-user virtual documents

User identifier	*Items*
user1	item1 item2
user2	item1
user3	item1 item2
user4	item3

Issuing specialized full text queries on such virtual documents enables a quick retrieval of similar items or users. For example finding top-k similar users to a given *user1* can be rewritten to a simple full text query: "item1|item2" where | represents the OR operator and *item1* and *item2* are items already seen by *user1*. Furthermore such full text query returns results ordered by relevance, which is in our case similarity. Finding similar items based on a list of users can be done by querying per-item virtual documents.

Since full text engines typically use weighting schemes[1] by using full text indices on per-user documents, lower weights are automatically given to items that are common for all users and higher weights for items that are connected to the same user several times (such as items bought by the same user on a regular basis). Similarly for per-item documents, users that tend to be connected to more items get lower weight since they are less discriminative.

By using these virtual documents and similarity queries we can now formulate our top-k nearest neighbor recommendation algorithm using pseudo code as follows:

```
Inputs: user_id, k, neighborhood size N
Output: top-k recommended items for user user_id

functionrecommend_with_fulltext(user_id, k, N)
   items = find_items_connected_to(user_id)
   items_query = create_query(items.ids)
   n_similar_users = query_per_user(items_query, N)
   n_similar_users.remove(user_id) # remove current user
   user_query = create_query(n_similar_users.ids)
   # find items not already seen, based on similar users
   similar_items = query_per_item(user_query, k, items)
   returnsimilar_items
end
```

At first, user history is retrieved by finding all items connected to a given user. Next a "full text" query is constructed by concatenating itemidentificators with OR-operator. This query is used to retrieve top N similar users (with similar history) from per-user virtual documents table. Next a specialized "full text" query is again constructed by concatenating user identificators with OR-operator. Finally this query is used to retrieve top k items, but excluding items already in user history, since we do not want to recommend items that user already is connected to. In short, this algorithm work in such a way that it first finds similar users based on user's history and then finds top-k items that these similar users have in their history.

This algorithm however makes an important and hidden assumption that needs to be addressed. Since full text engines are heavily optimized for short query strings (in our case low number of item/user identifiers in a query) and generally do not scale well for very long query strings, we need to make sure that *find_items_connected_to*does return only a limited number of items. This is a problem in theory, but in practice and thanks to nature of real-world datasets, we are mostly dealing with power-law distributions where most users are connected only to a very limited subset of items. In those rare cases where users are connected to a large number of items, a filtering based on a secondary heuristic (e.g. information gain or time-based weighting) should be considered. For the sake of simplicity, we ignore this detail in our further evaluation since implicitly used full text tf-idf weighting already addresses this problem at an intermediate level.

3 Evaluation

In order to test recommendation quality, speed and scalability we picked two real datasets for evaluation. Githubcontest data published by github.com and 15 days of server access logs from top Slovak online news publisher www.sme.sk.

Github.com is an online social collaborative repository hosting service based on git source control management, where users can create and share source code. Users can even fork and watch repositories of other users. A subset of github.com database was released as a challenge to build a recommendation system that recommends users

which repositories they should also watch based on repositories they are watching already, on their programming language preferences and other attributes. Github.com evaluated recommendation systems by querying top10 repository recommendations for 4788 users giving one point for each correctly guessed repository. In our evaluation we use the same approach, but for clarity we use ratio of correct guesses to all guesses and refer to this value as precision on top 10 – P10.

As a part of collaboration with largest Slovak online news publisher we have access to server logs of their online news portal www.sme.sk having roughly one million page views a day (excluding discussion forums). To evaluate our recommender system we are recommending articles to users based on their reading history. For evaluation we have used a 15 day window, splitting it into "train" and test sets. Similarly as in github.com contest, we generate top10 recommendations for each user in test set and add a point for each article that this user has actually visited in this test set. Again we evaluate precision on top 10 – P10. Table 3 shows size characteristics of these two datasets.

In evaluation we silently ignore the fact that such an evaluation cannot be used for testing the real precision or quality of any recommendation system since we are unable to simulate how users would actually react given such top-k recommendations. For example in evaluation using a train/test dataset in domain of news articles, a sole fact that a user has not seen a recommended article does not make it a bad recommendation, it could simply mean that the user has not found this article. On the other hand a sole fact that a user has seen a recommended article does not make this recommendation a good one. This could also mean that this recommendation was not necessary since the user found this article even without our recommendation system[2] [3].

However for our purposes (scalability analysis and parameter sensitivity) such train/test dataset evaluation is sufficient.

Table 3. Dataset size characteristics

Dataset	*github.com*	*sme.sk*
# of users	56 519	1 023 407
# of items	120 867 (repositories)	162 455 (articles)
# of entries	440 237 (followings)	11 996 530 (pageviews)

For all experiments we used two generic and freely available full text engines (MySQL 5.0 full text and SphinxSearch 0.9.8) as recommendation backends.

3.1 Recommendation Quality

To evaluate the precision of our top-k nearest neighborhood based method we use precision on top-10 metric. Since generic full text engines typically offer different tf-idf weighting schemes we also experimented with possible configurations of engines.

Figure 1 shows recommendation precision as a function of neighborhood size (N) and different engines and their configurations. The big drop for *Sphinx default* at N=10 is caused by additional Sphinx's full text word proximity weighting that gives more weight if words in query are close to each other. This is crucial for w counter-productive for our purpose, since we are not interested in which order of users/items appear in virtual documents.

We can see that for N > 40 precision gain starts to be negligible, showing that we do not need long queries for recommendation generation. Also Sphinx's *bm25* precision is superior to MySQL for lower N, but for higher N MySQL starts to be slightly more precise. This might seem as an advantage for MySQL, but quite the contrary is true because higher N means longer queries and scalability problems for large datasets as we show in experiments in section Scalability.

Figure 1 also shows that difference between weighting schemes *wordcount* and *bm25*(Sphinx bm25 and MySQL) is 3-4%. Here, wordcount refers to precision that could be yielded by creating basic inverted indices on user/item data and using our k-nearest neighborhood algorithm.

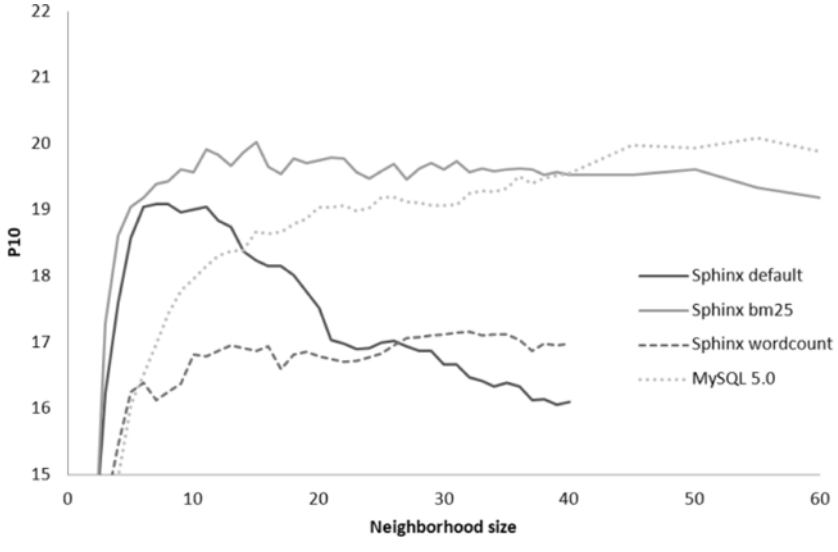

Fig. 1. Recommendation precision as function of neighborhood size and full text backend engine for github.com dataset

Since this evaluation only compares different configurations of the same algorithm we compared our method also with an existing spreading activation based recommendation method. Spreading activation is a recursive energy distribution algorithm with a threshold stopping criterion [4] [5]. Our results show that spreading activation on

github.com dataset yields best results (P10 = 19.6%) for starting energy = 500 and threshold = 0.1 which is slightly worse than best result of our k-nearest neighborhood based method (P10 = 20.0%).

3.2 Scalability

We have shown that our recommendation method yields comparable results as an existing spreading activation based approach. Figure 2 shows a scatterplot of time needed for recommendation generation versus precision on top10 for such recommendation. Since these are very different implementations to compare in an absolute manner (e.g. recommendation speed) we are only interested in scalability characteristics. Results show that while recommendation time for spreading activation grows exponentially due to neighborhood traversal explosion, k-nearest neighborhood method maintains a low footprint.

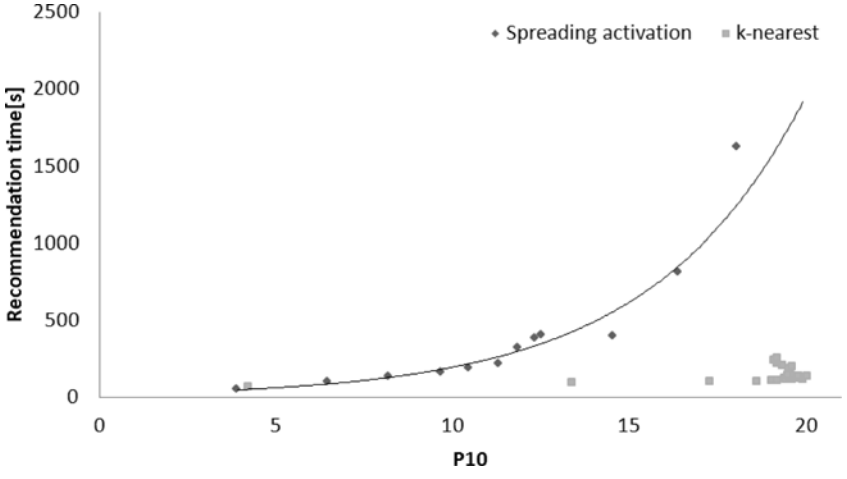

Fig. 2. Recommendation time versus precision on top-10 for spreading activation and k-nearest neigborhood based method

Figure 3 shows scalability analysis of recommendation generation time as a function of neigborhood size on github.com dataset for both recommendation engine backends and Sphinx variants. All configurations maintain linear scalability for varying neighborhood size, with *MySQL* being superior to *Sphinx bm25*.

Figure 4 shows scalability as a function of dataset size for larger sme.sk dataset showing that both algorithms maintain linear scalability relative to dataset size. However this analysis also shows that MySQL scales much worse, being superior to Sphinx only for smaller datasets and having a steeper linear characteristic beyond 250 thousand entries mark.

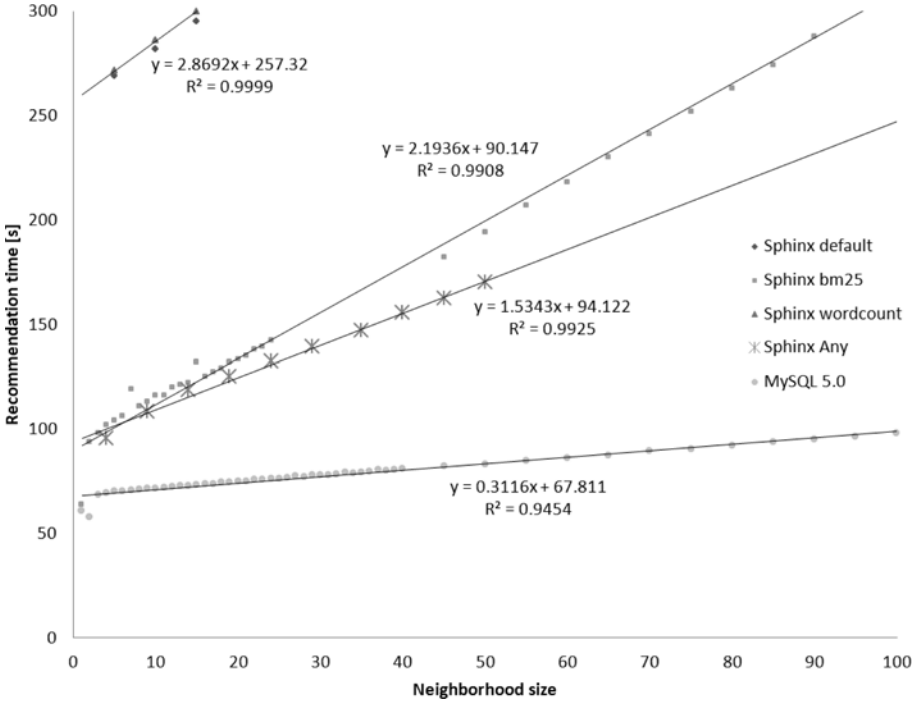

Fig. 3. Scalability analysis showing time needed for recommendation generation as a function of neighborhood size for various full text backend engines and settings on github.com dataset

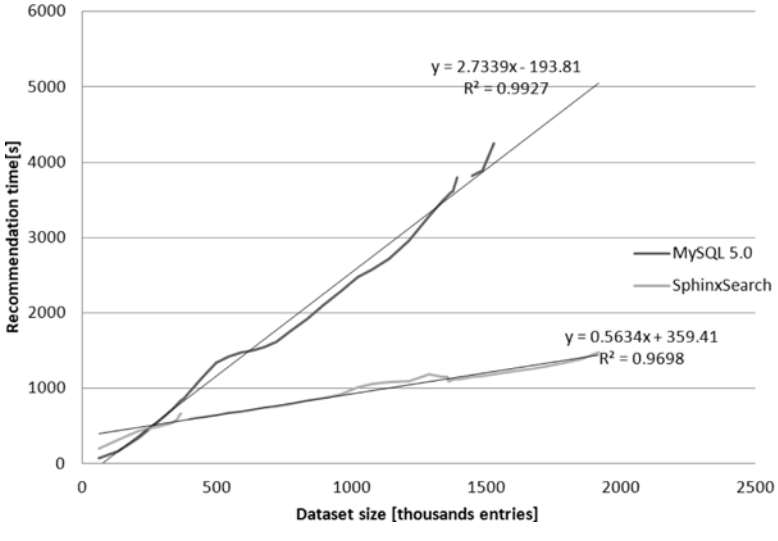

Fig. 4. Scalability analysis showing time needed for recommendation generation for different backend full text engines as a function of dataset size on sme.sk dataset (k = 35)

4 Related Work

Our nearest neighborhood recommendation method uses standard tf/idf weighting scheme typically used for document vectorizationin content-based recommendation methods [6], but we do not use cosine similarity to find similar vectors.

There are two major approaches to scalability problems when dealing with nearest-neighborhood-based recommendation systems. One approach is to reduce the number of item similarity comparisons required to retrieve most similar items. This can be done by pre-clustering similar items intosets, which are used for pruning possible item candidates [7]. The second, but more challenging approach is to rewrite existing algorithms in such a way that scalable frameworks (e.g. map-reduce) can be effectively utilized [8].

5 Conclusions

In this paper we have proposed a k-nearest neighborhood based recommendation method that exploits generic and generally available full text engines as back ends for quick similarity search. Using experiments on two real world datasets we have also shown that this method not only yields comparable results to spreading activation based recommendation methods, but is superior in means of linear scalability relative to dataset size. We have also addressed the following drawback of full text engines: they are optimized only for short queries. We have shown that a) in real world datasets power-law distributions are present and most users will be connected only to limited subset of items, thus finding similar users quickly is possible, and b) by showing that recommendation precision gains are negligible for high neighborhood sizes that cause long queries.

6 Future work

Our work is planned to be extended by

- adding additional columns to virtual documents, we would be able to match similar users/items based on more than one attribute. For example on github.com dataset we could also use preferred programming languages as an additional weighted similarity metric, or similarly on sme.sk dataset article categories.
- evaluating scalability and precision characteristics for additional backed engines such as PostgreSQL fulltext and Apache Solr.
- evaluating scalability in distributed environments on large datasets.

Acknowledgments. This work was partially supported by the grants VEGA 1/0508/09, KEGA 345-032STU-4/2010 and it is the partial result of the Research & Development Operational Programme for the project Support of Center of Excellence for Smart Technologies, Systems and Services, ITMS 26240120029, co-funded by the ERDF.

References

1. Buckley, C.: The importance of proper weighting methods. In: Human Language Technology Conference (1993)
2. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems (TOIS) 22, 1 (2004)
3. Karypis, G.: Evaluation of Item-Based Top-N Recommendation Algorithms. In: Conference on Information and Knowledge Management (2001)
4. Crestani, F.: Application of Spreading Activation Techniques in Information Retrieval. Artificial Intelligence Review 11(6), 453–482 (1997)
5. Salton, G., Buckley, C.: On the use of spreading activation methods in automatic information. In: Annual ACM Conference on Research and Development in Information Retrieval (1988)
6. Pazzani, M., Billsus, D.: Content-Based Recommendation Systems. The Adaptive Web, 325–341 (2007)
7. Sarwar, B.M., Karypis, G., Konstan, J., Riedl, J.: Recommender Systems for Large-scale E-Commerce: Scalable Neighborhood Formation Using Clustering (2002)
8. Das, A.S., Datar, M., Garg, A., Rajaram, S.: Google news personalization: scalable online collaborative filtering. In: International World Wide Web Conference (2007)