CS229 COURSE PROJECT: RECOMMENDATION SYSTEMS

BAHMAN BAHMANI

1. Introduction

With the rapid growth of online data, it is constantly getting more difficult for people to find what they need or like. One aspect of this difficulty is addressed by search engines. But, search engines are only useful when the user exactly knows what he needs and can express it in terms of a search query. But, because of the sheer amount of data, more often than not the users do not even know what they may need or like to consume. This raises the need for recommendation systems; systems that can automatically find and expose to users the information that they may like. For instance, a user who wants to buy an interesting book, may not even know what options are available to her. Thus, we would desire to be able to automatically find the books that the user may enjoy and then recommend her to consider those books.

A recommendation system is desired to be:

(1) Personalized; that is, it should be able to tailor its recommendations to each single user's taste
(2) Scalable; that is, it should be able to handle the massive data sets usual in applications of interest

For instance, a very simple type of recommendation system is editorial recommendation. But, it does not have any of the above properties. Another very simple recommendation system is the Top 10 lists (or the list of most popular items). But, it does not have any personalization.

There are two main approaches to building a recommendation system having the mentioned desired properties. In the next secion, we will briefly explain each approach.

2. Approaches to building Recommendation Systems

2.1. **Content-based approach.** The basic idea in the content-based approach is that the system recommends items with similar content to the ones previously liked by the user. To achieve this goal, a profile is constructed for each item, and then items are recommende based on the similarity between their profiles and the profiles of the previously liked items. A similar idea can be used in the space of users instead of in the space of items. That is, one can construct a profile for each user, and then recommend to a user the items that users with similar profiles liked.

One can also adopt a more model-based approach in the content-based framework. For instance, one can design the system to learn a classifier for each user that classifies the items into rating classes for that user. All the usual classification methods (such as Bayesian methods, SVM, etc.) can be investigated in this approach. However, this approach usually has issues with scaling (e.g. with a huge number of users, it will be infeasible to learn a separate classifier for each user).

2.2. **Collaborative Filtering.** The basic idea in collaborative filtering is that the users who liked the similar items in the past are similar and are hence more likely to like the same items in the future. Therefore, the system recommends to a user items which are similar to the ones which were previously liked by similar users. Similarity in the items is also defined on past ratings of users. That is, the items which got rated similarly by users in the past are more likely to be rated similarly in future as well. Notice that this approach is solely based on the similarities of past ratings, and not, for instance, the similarity of contents.

In the next section, we will explain the problem that we considered in this project.

3. Problem Considered

In this project, we considered the problem of recommending friends on Twitter's social network. Twitter is a social network in which users can follow the feed of updates, called tweets, of other users. In this project, we would like to make a recommendation system that for each user comes up with a list of users that may be interesting for her to follow.

In this problem, since the tweets are very short (at most 140 characters), and Twitter does not ask for a lot of information from its users upon registeration, there are not a lot of content-based features about users that would be useful for a recommendation system. Also, the model-based approach (in the content-based framework) is doomed to failure, not only because of the mentioned lack of useful features, but also because of the huge number of users on the network.

Therefore, we decided to use a network-based (aka link-based) collaborative filtering approach for this problem. In the next section, we explain the exact algorithms that we studied for solving this problem.

4. Algorithms Studied

We will denote the network with $G = (V, E)$.

4.1. **HITS (aka Hubs and Authorities/ PCA/ SVD).** The original HITS algorithm is a ranking algorithm in which each node in the network is assigned two scores, called hub score and authority score [1]. These scores are related as follows:

$$h_v = \sum_{\{x|(v,x)\in E\}} a_x$$
$$a_x = \sum_{\{v|(v,x)\in E\}} h_v$$

To personalize this algorithm so it will be useful for the recommendation system, we adopted the random teleport idea of PageRank which, when personalizing for the user u, results in the following modification of above formulae:

$$h_v = (1-\epsilon)\Big(\sum_{\{x|(v,x)\in E\}} a_x\Big) + \epsilon\delta_{u,v}$$
$$a_x = \sum_{\{v|(v,x)\in E\}} h_v$$

Then, the hub score h_v represents the level of similarity of v to u, and the authority score a_x represents the perceived quality of x as a recommendation to u.

Notice that the HITS algorithm basically corresponds to doing PCA on the adjacency matrix of the network[1].

4.2. **SALSA.** The SALSA algorithm has the same flavor as HITS, except it distributes the scores uniformly through the edges [2]. In other words, it mixes the idea of the HITS algorithm with the idea of random walks. The equations governing SALSA are as follows:

$$h_v = \sum_{\{x|(v,x)\in E\}} a_x/\mathrm{indeg}(x)$$

$$a_x = \sum_{\{v|(v,x)\in E\}} h_v/\mathrm{outdeg}(v)$$

Again, using the same trick as above to personalize this algorithm for the recommendation setting, we get the following equations:

$$h_v = (1-\epsilon)\Big(\sum_{\{x|(v,x)\in E\}} a_x/\mathrm{indeg}(x)\Big) + \epsilon\delta_{u,v}$$

$$a_x = \sum_{\{v|(v,x)\in E\}} h_v/\mathrm{outdeg}(v)$$

4.3. **COSINE SIMILARITY.** We represent the set of friends of each user u, as a $0-1$ vector, denoted f_u. Then, we define the hub/similarity scores as:

$$h_v = cos(\angle(f_u, f_v))$$

and the authority scores using:

$$a_x = \sum_{\{v|(v,x)\in E\}} h_v$$

4.4. **PageRank.** Denoting the personalized PageRank of user v by π_v, we have [3]:

$$\pi_v = \epsilon\delta_{u,v} + (1-\epsilon)\Big(\sum_{\{x|(x,v)\in E\}} \pi_x/\mathrm{outdeg}(x)\Big)$$

In the next section, we explain how we compared these different algorithms for our problem.

5. Performance Comparison Methodology

We picked 100 random users from the network. We used the network data from two different dates. The earlier date network was used to find the recommendation list for each of the selected users using each of the above algorithms. Then, we used the network data from the later date to find out how many (in average) of the actual new friends of each user were predicted in the Top 100 and Top 1000 recommendations of each algorithm.

Notice that the basic idea in this comparison method is that we assume giving enough time to the users to explore the network and add new friends, they will be

	HITS	COSINE	PageRank	SALSA
Top 100	0.25	4.93	5.07	6.29
Top 1000	0.86	11.69	12.71	13.58

Table 1. Collaborative filters performances

able to find the users they would like to follow, and hence it is desirable for our recommendation system to be able to predict the actual friendships made. This will give us an unbiased characterization of the performance of the system.

Of course, notice that the resulting numbers will be found for the case when the users were not exposed to their recommendation lists at all. Hence, we do not expect the numbers to be high. Also, the actual performance of each recommendation system will be actually much better when the users are exposed to its results. But, still our method gives an objective comparison of the predictive ability of different recommendation systems.

In the next section, we will present the results that we obtained, and we will also discuss a very interesting observation.

6. Results

The results of our experiments are summarized in Table 1. Again, we emphasize that the actual (absolute) numbers in this table are not indicative of each recommender system's actual performance, but the comparison and the directionality in the results are significant.

The very interesting result that we can observe from Table 1 is that HITS, which is considered as the standard algorithm and is also very closely tied to PCA and SVD [1], actually achieves a very poor performance compared to the other algorithms considered, while the SALSA algorithm, which basically mixes the idea of HITS and the idea of PageRank, achieves the best performance. We believe this is a very interesting result. Notice that there is already some literature [4] showing that SALSA is much more effective than HITS in the ranking application. Here, we arrive at a similar result in the recommendation systems application.

References

[1] J. M. Kleignberg, "Authorative sources in a hyperlinked environment", J. ACM 1999.

[2] R. Lempel and S. Moran, "SALSA: the stochastic approach for link-structure analysis", ACM Trans. Inf. Syst. 2001.

[3] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web", Tech. Rep., 1999.

[4] M. A. Najork, "Comparing the effectiveness of HITS and SALSA", CIKM'07.