Experiences from Implementing Collaborative Filtering in a Web 2.0 Application

Wolfgang Woerndl, Johannes Helminger, Vivian Prinz

TU Muenchen, Chair for Applied Informatics – Cooperative Systems
Boltzmannstr. 3, 85748 Garching, Germany
{woerndl, helminger, prinzv}@in.tum.de

Abstract. The goal of this paper is to report our experiences from integrating item-based collaborative filtering into the Web 2.0 site linkfun.net. We discuss the necessary steps to implement the selected Slope One algorithm in our real world application. It was necessary to conduct performance optimization to allow for recommendations without any delays in page generation on our site. Firstly, we significantly reduced the data model by including only items similarities for pairs of items where both items been rated by at least k users. Secondly, we precomputed recommended items for users. By analyzing the empirical results, we found out that user activity increased on the site after introducing the recommender. In addition, users rated recommended videos higher on average than others which indicates that the recommender allowed users to find preferred videos more effectively.

Keywords: recommender systems, collaborative filtering, slope one, performance optimization

1 Introduction

Web 2.0 applications such as MySpace, YouTube or FlickR have gained much interest in industry and academia in recent years. Users can easily upload content such as videos, photos or links in order to share them with others. However, most current sites lack structured intelligence and finding meaningful information can be difficult [1]. Recommender systems and collaborative filtering are techniques to deal with this problem as they filter information items according to a user's needs and taste.

In this project, we are investigating the integration of a collaborative recommendation system in the real world Web 2.0 site *www.linkfun.net* (Fig. 1)[1]. This site allows users to share links to funny content such as videos. While other sites like YouTube feature all kind of videos, linkfun.net focuses on humorous content. In addition, linkfun.net does not host the videos itself, but users provide links to content on various other sites. The overall goal of the work presented in this paper was to provide good recommendations to users and thus increase user interest in the site and expanding the community and value of linkfun.net. Notable requirements included the

[1] The user interface of linkfun.net is currently in German only

integration of the recommendation without decreasing the performance of the site, extra hardware needs or additional effort for the users.

Fig. 1. Screenshot www.linkfun.net

The rest of this paper is organized as follows. The next section gives some background on recommender systems and discusses which type of system appears well suited for our scenario in principle. In section 3, we explain the necessary steps to integrate the filtering functionality in linkfun.net. In section 4, we present the empirical results from the analysis of log files. Finally, we conclude the paper with a short summary and an outlook.

2 Recommender Systems and Collaborative Filtering

The basic idea of recommender systems is to recommend products like books and CDs and other items such as restaurants or videos to an active user. To do so, the system computes the chance that a user likes an item. This is based on information about the user the items and possibly other data such as contextual information. Characteristics of recommender algorithms include the quality of recommendations, storage and runtime complexities, anonymity and extensibility of the model.

2.1 Individual Recommender Systems

In general, we distinguish between individual and collaborative recommender systems. Individual recommenders determine fitting items based on the profile of the active user. Thereby, the system matches explicitly entered or implicitly observed user preferences and interests with items meta data. Hence, this type of recommender system is often called content-based recommender system. One way of implementing this kind of recommender is to use a rule system. However, the content-based approach is not well suited for Web 2.0 content because additional information about the items and/or users is required. Moreover, an individual recommender does not fit the social and collaborative nature of Web 2.0 applications.

2.2 Collaborative Recommender Systems

The second category of recommender systems are based on collaborative filtering (CF). CF utilizes the ratings of other users for items, for example a rating on a scale of 1 to 5. The vector of all ratings of a user for various items is called a user's rating vector. CF seems appropriate for Web 2.0 because it needs no information about items and implements the "word of mouth" idea that is also prevalent in Web 2.0. Users like to express their opinions on content and basic rating schemes already exist in some sites.

We differentiate between two variants of CF, user- and item-based collaborative filtering. The recommendation process of user-based CF basically consists of two steps. First, neighborhood creation: Determine a set of k users that have rated similarly to the active user in the past. Second, recommendation of new items for the active user. For neighborhood creation, the active user's rating vector is compared to the vectors of all other users. To do so, different metrics have been proposed in the literature, for example Euclidean distance, cosine similarity or Pearson-Spearman correlation [2]. Thus, user-based CF analyzes the available raw data, namely the user-item matrix of ratings. In the second step, the algorithm selects items, which the active user has not rated yet, but which have been rated positively in the neighborhood of the active user. User-based CF has proven very useful and accurate in applications such as Web shops. However, this type of collaborative recommender has several drawbacks. First of all, there is the new user problem. User-based CF cannot generate a suggestive recommendation if the active user has not rated any items. In a Web 2.0 site such as linkfun.net, new or occasional users may represent a high share of the user base. A second relevant problem of user-based CF is that the approach is computationally costly. The approach operates on the raw data of ratings, which have to be analyzed each time a prediction is computed.

2.3 Item-based Collaborative Filtering

The alternative approach is item-based CF. Item-based CF does not consider the similarity of users, but of items [3]. Thus, the user-item matrix is not analyzed line by line, but column by column. One significant difference to user-based CF is the

independence from who the active user is: the item similarities can be precomputed to build an item-item matrix. The item-item matrix is the model of the algorithm. Therefore, this type of recommendation algorithm is also called model-based collaborative filtering. One element $S_{i;j}$ of the item-item matrix expresses the similarity between items i and j, determined from the users' ratings. The ratings vector of the active user is then used to recommend items that are similar to items the active user has rated positively in the past.

It is important to note that item-based CF has little in common with individual, content-based filtering. This is because the users' ratings are solely used for computing the item similarity. Meta data of items is irrelevant. Item-based CF has an advantage over user-based CF with regard to the complexity of the computation. The item-item matrix can be calculated as an intermediate result, independently from the active user. The main advantage of item-based CF is performance, because generating recommendations from the model instead of the raw data of ratings is much more efficient.

Slope One [4] is an example of an item-based CF algorithm. The main idea of the approach is to use differentials of ratings and store them in the item-item matrix. Slope One uses predictors of the form $f(x) = x + b$, which precompute the average difference between the ratings of two items for users who rated both items [4]. Consequently, the model can be updated on the fly, without the need to recalculate the model when a rating is made. In addition, it is not demanding as much information from new users. One rating is enough to be able to generate recommendation for a user. Finally, [5] describes a straightforward implementation in PHP using a SQL database. Linkfun.net was also implemented in PHP and SQL, so we decided to base our recommender on Slope One. More details on Slope One and our implementation are given in the next section.

3 Data Model, Recommendations and Optimization

In this third section of the paper we explain the design decisions when integrating Slope One into linkfun.net and implementing the collaborative filtering method.

3.1 Data Model

The initial situation in linkfun.net was that users were able to give ratings on a scale from 1 to 5 with 5 being the best grade. However, the application did only save the aggregated average value for each item. Information about the individual ratings of users was not kept. Slope One and all other CF algorithms do need the detailed user-item matrix though.

Hence, the existing ratings had to be discarded and a new data model had to be designed. This new model consists of three database tables to store the necessary information:

- Table `rating` to store the user ratings for items, with one rating corresponding to one row in this table.

- Table `dev` for the average deviations of ratings for an item-item pair [5]. This table implements the item-item matrix or, in other words, the model of Slope One.
- Table `rating_recom` to log when a video was recommended to a particular user and optionally how the user rated it after the recommendation.

Every time a user *u* rates an item *i*, our system updates `rating` and calculates the impact on `dev`. To do so, the algorithm first determines all items u has rated, computes their rating differentials to *i* and updates the affected entries in `dev`.

3.2 Generating Recommendations

The model can then be used to generate and display the top five recommended items – primarily videos – when a user accesses linkfun.net. Slope One distinguishes between non-personalized and personalized recommendations [5]. Non-personalized recommendations are not based on collaborative filtering in the strict sense. After a user rates just one item *i*, the algorithm searches for items, which have the highest rating differential to *i*, i.e. were rated best in comparison to *i*.

To predict the rating for a particular item *i* for an active user *u*, the algorithm first selects the set of items *rs*, which were rated by *u* and also by at least one other user. In the second step, the differentials between the ratings of *i* and the items in *rs* are determined using the item-item matrix, respectively our database table `dev`. Finally, the differentials are summed up and divided by the number of items in *rs*. This results in a predicted rating for *i*. Note that the predicted rating is possibly higher than the highest grade, for instance 5 on a 1-5 scale. This is because Slope One works on rating differences [4]. However, it is only important to compare the predicted values of items to each other to be able to generate a ranked list of items for the personalized recommendations.

3.3 Optimization

The number of elements in the item-item matrix – i.e. the table `dev` – soon grew to over 3.4 million entries with about 2000 items (Table 1). This is due to the fact that with more ratings of users, more and more items receive at least one rating and more and more similarities between item pairs can be calculated. This led to delays in handling ratings and computing recommendations. It hurt the overall user experience on linkfun.net because page generation was noticeably slowed. This necessitated performance optimization.

We implemented two solutions to improve performance:

1. Reducing the data model and the number of entries of the table `dev`
2. Precomputing recommended items for users

Table 1. Reducing the data model

	Number of entries in `dev`	Available videos for recommendation
No threshold	3404049	1961
k=2	2436997	1885
k=3	1397523	1759
k=4	*605561*	*1555*
k=5	198889	1278
k=10	43	19

The basic idea to minimize the size of the item-item matrix is to only compute and store item similarities for pairs of items where both items been rated by at least *k* users [5]. The obvious drawback is that some items, which have received few ratings, are not represented in the item-item matrix anymore. Thus, these items are no longer available for recommendation. Table 1 shows the number of entries in `dev` and the number of available videos for certain values of the threshold k of necessary ratings. At that point of time, a reasonable value for *k* was *k=4*. This value reduces the numbers of entries in `dev` significantly from 3404049 to 605561 while keeping about 80% of items available for recommendations.

Despite this reduction, generating recommendations when a user hits the corresponding link took still too much time. Hence, the recommendation had to be precomputed. To do so, we created a new database table `precomp_recom` that stores five recommended items for every user. This table is updated according to the following schedule:

- Once per day, the recommended items for all users are recalculated. This procedure takes about 15 minutes. It is performed during the night when less users access the site.
- The recommendations are updated every 5 minutes for users that have rated items since the last update.

The second condition ensures that recommendations are updated promptly and regularly for active users and reflect their latest ratings. In any case, the model, i.e. the item-item matrix as basis for the recommendations, may be slightly out of date. Yet this fact has to be accepted to allow for instant recommendations by precomputing them.

In general, CF algorithms may suffer from cold start problems with new users or new items. For example, new items cannot be recommended until they receive at least one rating. This was not a problem in our case. Most of our items were rated within hours and thus were potentially considered for recommendations reasonably soon. As far as new users are concerned, newly registered users to linkfun.net were shown a message that they need to rate at least one video to obtain recommendations.

4 Empirical Results

In this section, we analyze the log files to evaluate the effect of the CF function on linkfun.net. We were in particular interested in two questions:

- User activity: what were the effects of the recommender system on user activity?
- Quality of recommendations: were recommended videos rated higher on average than other items?

4.1 User Activity

At time of research, there were 490 registered users of linkfun.net, although some of the registered users frequented the site rather seldom. 150 users rated at least one item and 50 users actively used the recommendation function. Overall, there were 10500 ratings and 100000 times a video was played by users. About 60% of the video playbacks occurred after using the recommendation function.

Table 2. User activity with regard to rating items

Total number of users	**490**	**100%**
Users with more than 1 rating	150	30,6%
Users with more than 5 ratings	87	17,8%
Users with more than 10 ratings	67	13,7%
Users with more than 20 ratings	54	11,0%
Users with more than 50 ratings	30	6,1%
Users with more than 200 ratings	12	2,4%

Table 2 shows the distribution of the number of user ratings. The table shows, that a rather small user base ("early adopters") contributed to most of the ratings. Twelve users have accounted for about 8600 ratings or roughly 75% of all ratings.

For overall user activity we looked at the visitor logs after the recommendation function was introduced. Fig. 2 illustrates the sessions and page visits per month over the course of the research period. While the number of sessions increased only gradually, we noticed a far bigger increase in page impressions. This means that the average time users spent on the site grew considerably. Although we are not able to measure the exact impact of the recommender on site activity, the overall goal of increasing user activity was met.

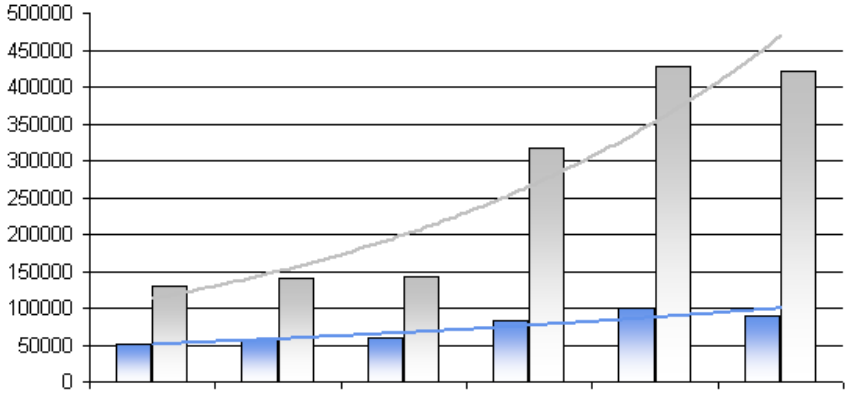


Fig. 2. Sessions (left bars) and page visits (right bars) per month

4.2 Quality of Recommendations

As far as the quality of recommendations is concerned, we investigated the ratings the users gave to the videos. Overall, the videos were rated high on average with many videos receiving the best grade. This may be due to the fact that linkfun.net is a specialized community where users only provide links to funny content that may cater to a similar taste. We noticed a trend that this high rating average increased even further after the introduction of the recommender function (Fig. 3). Our assumption is that the rating and recommendation scheme allowed users to find preferred videos more effectively.

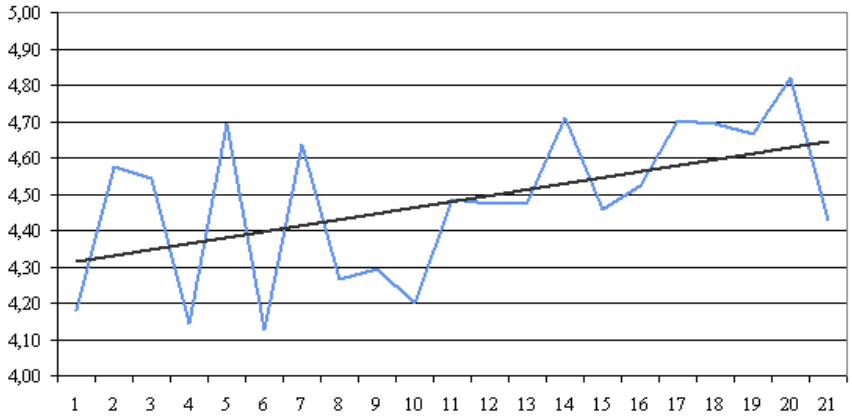
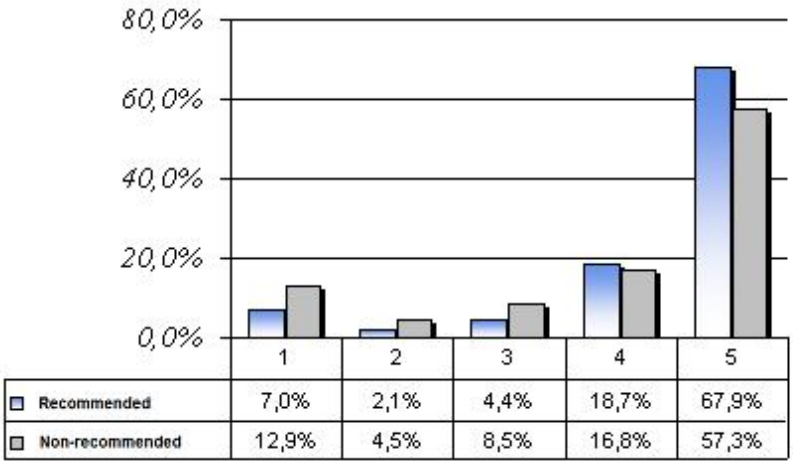


Fig. 3. Trend of average ratings per week

To evaluate the quality of recommendations in more detail, we compared the ratings of videos that were recommended to a user with the ratings of videos that were not in the list of recommendations. The latter category consists of videos accessed from the homepage of the site or from a "newest videos" section. We found out that recommended videos received higher grades: the average rating was 4.4 in comparison to 4.0 for non-recommended videos.



	1	2	3	4	5
Recommended	7,0%	2,1%	4,4%	18,7%	67,9%
Non-recommended	12,9%	4,5%	8,5%	16,8%	57,3%

Fig. 4. Distribution of ratings: recommended videos (left bars) vs. non-recommended videos (right bars)

Fig. 4 shows the distribution of ratings for the videos in more detail. For example, 67,9% of recommended videos were rated with the top grade ("5"), while the percentage of top-graded non-recommended videos is significantly lower (57,3%).

5 Conclusion

In this paper, we have described our experiences from integrating the item-based collaborative filtering algorithm Slope One into the Web 2.0 site linkfun.net. We explained the necessary steps including the data model. After we have done some performance optimization, the recommender function ran smoothly on the site, without any delays in user experience or additional hardware requirements. The performance optimization included reducing the data model of item similarities and precomputing recommended items for users. Overall, the Slope One algorithm proved to be very practicable and fitting for the examined site. By evaluating our implementation, we found out that user activity increased on the site after introducing

the recommender. In addition, users rated recommended videos higher on average than others.

As far as related work is concerned, there are several applications which also use SlopeOne in practice. For example, InDiscover (http://www.indiscover.net) is a site for promoting independent musician and recommending new music to interested customers. Their system uses RACOFI which is a framework for rule-based collaborative filtering partly based on Slope One [6]. However, there is no published information about performance optimization or empirical results. There is plenty of research in improving recommender systems, mostly with a focus on prediction quality [7], but there are few reports on experiences from applying recommender algorithms in practical Web 2.0 applications. Leimstoll and Stormer discuss in [8] how collaborative filtering can be integrated in online shops in principle. Their proposal is somewhat similar to our approach, although no experiences from real world applications are reported.

One goal of our planned future activities is to integrate implicit ratings that can be observed from user behavior. So far, all recommendations are based on explicit ratings users have made. We are currently investigating methods to measure the exact time a user spends with a video. When a user is watching a video until completion, one can assume that she liked the video, which relates to a good rating. If the user cancels the playback soon after the start, we would assign a low rating. Subsequently, we want to measure whether recommendation based on these implicit ratings derived from user observation performs as well as the explicit user ratings.

References

1. Lin, K.-J.: Building Web 2.0. IEEE Computer, vol. 40, no. 5, pp. 101-102 (2007)
2. Adomavicius, G.; Tuzhilin, A.: Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Transactions on Knowledge, and Data Engineering, vol. 17, no. 6 (2005)
3. Sarwar, B.; Karypis, G.; Konstan, J.; Riedl, J.: Item-based Collaborative Filtering Recommendation Algorithms. Proc. 10th International Conference on World Wide Web (WWW10), Hong Kong, China (2001)
4. Lemire, D., MacLachlan, A.: Slope One Predictors for Online Rating-based Collaborative Filtering. SIAM Data Mining (2005)
5. Lemire, D., McGrath, S.: Implementing a Rating-Based Item-to-Item Recommender System in PHP/SQL. Available at: http://www.daniel-lemire.com/fr/documents/publications/webpaper.pdf (2005)
6. Anderson, M., Ball, M., Boley, H., Greene, S., Howse, N., Lemire, D., McGrath, S.: RACOFI: Rule-Applying Collaborative Filtering Systems. IEEE/WIC COLA'03, Halifax, Canada, (2003)
7. Pu, P., Bridge, D., Mobasher, B., Ricci, F.: Proceedings of the 2008 ACM Conference on Recommender Systems (RecSys '08). ACM (2008)
8. Leimstoll, U., Stormer, H.: Collaborative Recommender Systems for Online Shops. AMCIS 2007, Keystone, CO (2007)