Methods to improve the accuracy of recommender systems

Martin Powers
University of Minnesota - Morris
600 East 4th Street
Morris, Minnesota
power182@morris.umn.edu

ABSTRACT

With commercial recommender systems becoming more and more popular, new ways to improve their accuracy and ease of implementation are being evaluated. These methods aim at creating more scalable and accurate systems by examining trends and finding patterns within the data. Two methods that have been developed achieve these goals by using item-based algorithms instead of user-based ones and by adding a temporal dynamic. Item-based systems can produce similar results to user-based systems while the number of users increases by comparing item similarities which are fairly constant. Temporal dynamics are important to examine because user preference changes over time and so does their average rating of an item.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Algorithms

Keywords

collaborative filtering, recommender systems

1. INTRODUCTION

In the age of E-commerce, more products are being released to the public than any one person has time to experience. With this vast number of options to process, people rely on recommendations on which movies they should watch and books they should read [4]. These software systems are called *Recommender Systems* and can now replace the need to know someone who has seen a movie you're interested in; recommender systems often give more accurate predictions when a large dataset is used. This computation goes mostly unnoticed by users while browsing E-commerce websites but has become some company's central way of making money. Because it can be extremely beneficial to both a company and its users, there has been a great deal of research to improve the efficiency and scalability of recommender systems [4]. Out of the many areas where recommender systems are being improved, this paper will focus on two: item-based algorithms and the temporal model. Item-based algorithms focus on the similarities between items rather than users and can show improvements over other methods in certain cases. A recommender system that uses a temporal model is one that takes into account how a user's or item's properties change over time. By analyzing user and item trends while tracking temporal dynamics, it is possible to make improvements to a recommender system.

In this paper, we will look at the fundamental aspects of recommender systems and explore some of the new advances. Starting in 2, we will take a look at what a user-based recommender system looks like. From there we will look at two new methods to see how they differ from common systems. Item-based systems are explored in 3 and a temporal aware model is shown in 4.

2. BACKGROUND

Recommender Systems make suggestions to users based on information the system has available to it such as item properties or user preference patterns. The majority of recommender systems used today are based on information that users provide to the system. These types of recommender systems are called Collaborative Filters and rely on a large number of users rating a set of items. These ratings could be collected from users explicitly, like having them give a movie a star rating, or behind the scenes, like counting how many times a song is played. Users are more inclined to input their ratings into a system where there is a noticable reward such as accurate suggestions [2]. It is easy to see why this type is the most successful and common recommender systems as they help both the user find things they like and the owner of the implementation get more business.

2.1 Collaborative filter method

The basic elements needed in a collaborative filter is a list of m users $U = \{u_1, u_2, \cdots, u_m\}$ and a list of n items $I = \{i_1, i_2, \cdots, i_n\}$; these are commonly very large lists. Each user u_i, has a list of items I_{u_i} that they have rated in some way, and using these ratings and the ratings of other users we can find users with similar taste or items that have similar likeness. The idea behind these suggestions is that if two users rate an item or items similarly, they might like the other's highly rated items. There are two methods through

which collaborative filters can suggest items to an active user: Making a prediction or making a recommendation.

A prediction is described in [4] as "a numerical value, $P_{a,j}$, expressing the predictive likeliness of item $i_j \notin I_{u_a}$ for the active user u_a." The item must not exist within the list of items already rated by the user. This method can be used to predict how a user might like an item based on properties the item shares with other items the user has rated. This is used for items that are newly introduced and have no ratings yet, hoping to find users that will like it based on little data.

The recommendation method is also known as the Top-N recommendation and is a list of items that the user will probably like based on comparing ratings with other users or items with similar qualities to the active user or the items the active user likes.

The collaborative process is diagramed in Figure 1, taken from [4], where the process is shown in the three steps: building the user-item matrix from the data, running a filtering algorithm, and finally the outputted information.

2.2 User-based algorithms

The most common collaborative filtering algorithms find predictions for a user based on other users with similar taste. Similar users are called neighbors and are determined by comparing the items both of them have rated. The Pearson correlation algorithm is one of the methods to find how similar two users are to each other by comparing the set of items both users have rated:

$$\text{userSim}(u,n) = \frac{\sum_{i \in I_{u,n}} (r_{ui} - \overline{r}_u)(r_{ni} - \overline{r}_n)}{\sqrt{\sum_{i \in I_{u,n}} (r_{ui} - \overline{r}_u)^2} \sqrt{\sum_{i \in I_{u,n}} (r_{ni} - \overline{r}_n)^2}}$$

This algorithm will find how similar user u is to its neighbor, user n based on all the items that both have rated, shown as $I_{u,n}$. The rating user u has given item i is shown as r_{ui}. The result will be within the range from -1, showing perfect disagreement, and 1, being perfect agreement [5].

This similarity is used when determining a predicted rated, $P_{u,i}$ for an item i that user u hasn't rated yet. This is calculated by finding the sum of all ratings for i given by u's neighbors, each weighted by how similar u is to each neighbor. This is divided by the sum of u's similarity with their neighbors so that the result will be within the same scale as the ratings. This prediction is given by:

$$P_{u,i} = \frac{\sum_{n \in N_u} \text{userSim}(u,n) \cdot r_{ni}}{\sum_{n \in N_u} \text{userSim}(u,n)}$$

where N_u is the set of u's neighbors.

This algorithm works best when all users have the same rating distribution which can be seen by looking at the average rating of of a user, $\overline{r}_u$. In a system that uses a 5 star rating scale, one user might rate items conservatively and never have a 5 star rating, using 4 stars as a sign of a great item. Compared to another user who has many items rated as 5 stars, they both have a skewed scale in relation to each other. Because most users use a unique rating scale, compensating for this bias greatly improves prediction accuracy. To do this, we subtract each neighbor's average rating, $\overline{r}_n$, from the rating they gave item i and add user u's average rating, $\overline{r}_u$ to the total prediction:

$$P_{u,i} = \overline{r}_u + \frac{\sum_{n \in N_u} \text{userSim}(u,n) \cdot (r_{ni} - \overline{r}_n)}{\sum_{n \in N_u} \text{userSim}(u,n)}$$

3. ITEM-BASED METHOD

Item-based collaborative filtering algorithms use properties of the items instead of using user similarity as the means to create recommendations and predictions. The properties of items used by a recommender system change less often than the number of users and their preferences. This consistency helps improve how scalable the system is, which is important as the number of users increases and each user provides more ratings.

3.1 Challenges of user-based collaborative filtering algorithms

There are two major challenges that the user-based method faces: scalability and sparsity. When a user-based recommender system needs to make a suggestion for a user, it must first find the user's neighborhood of users with similar taste which requires making a fair number of comparisons between how users have rated a set of items. With each user having the ability to rate as many items as they want, this number of comparisons grows quickly with each new user.

Also, some users also might not give any ratings or a very small number of them and the system needs to be able to work with these users as well. This sparsity is a problem for new users who haven't rated any items and expect results from the system immediately, but get poor recommendations. Both these challenges are addressed by item-based systems and are a major reason that large scale systems might use them as an alternative or in addition to other methods.

3.2 Item similarity computation

Item-based algorithms start by taking two items and finding their similarities through a similarity algorithm. These algorithms use the list of all items and find users who have rated both items. With this list they can figure out how similar the two items are. If a lot of users have given both items similar ratings, it can be said that the items are alike, and the more similar the ratings, the more similiar the items. This can be done many different ways and three are explained here: cosine-based, correlation-based, and adjusted cosine.

3.2.1 Cosine-based similarity

The two items i and j are represented as the vectors $\vec{i}$ and $\vec{j}$ in the m dimensional user-space to be able to calculate their similarity. The similari]ty is measured by computing the cosine of the angle between the two vectors. This is formally shown as

$$\text{itemSim}(i,j) = \cos(\vec{i},\vec{j}) = \frac{\vec{i} \cdot \vec{j}}{||\vec{i}||^2 * ||\vec{j}||^2}$$

where "·" equals the dot product of two vectors.

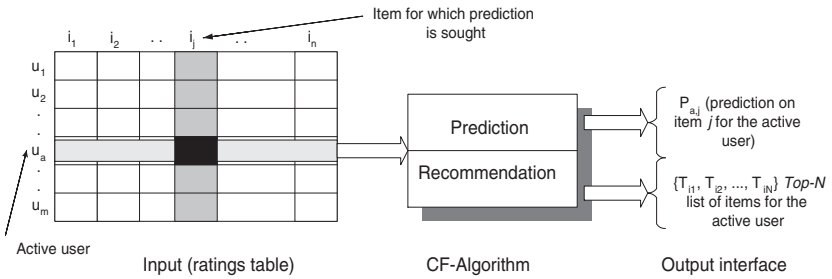

Figure 1: Collaborative Filtering Process

3.2.2 Correlation-based similarity

This method computes the Pearson-r correlation, as seen in Section 2.2, to measure the likeness between two items. First, the users who have rated both i and j are found and represented by U. Then the similarity is

$$\text{itemSim}(i,j) = \frac{\sum_{u \in U}(r_{u,i} - \overline{r}_i)(r_{u,j} - \overline{r}_j)}{\sqrt{\sum_{u \in U}(r_{u,i} - \overline{r}_i)^2}\sqrt{\sum_{u \in U}(r_{u,j} - \overline{r}_j)^2}}$$

where $r_{u,i}$ is the rating user u gave item i and $\overline{r}_i$ is the average rating of item i over all users in U.

3.2.3 Adjusted cosine similarity

Unlike user-based algorithms, item-based comparisons are made using many users, each of which have a different rating style. Some users, for example, might reserve the highest rating for only a few items and give all other items a lower-than-average rating. To compensate for this, the cosine similarity is adjusted by subtracting a user's average rating from their rating of the two items being compared. This is another form of the Pearson-r correlation agorithm, adjusted to compensate for a user's average rating instead of an item's. This is shown as

$$\text{itemSim}(i,j) = \frac{\sum_{u \in U}(r_{u,i} - \overline{r}_u)(r_{u,j} - \overline{r}_u)}{\sqrt{\sum_{u \in U}(r_{u,i} - \overline{r}_u)^2}\sqrt{\sum_{u \in U}(r_{u,j} - \overline{r}_u)^2}}$$

.

3.3 Prediction computation

Once a set of items similar to each other is generated, the next step in an item-based system is to consider a user's recorded ratings and create a predicted rating for each unrated item. This can be done a number of ways, two of which are thes weighted sum algorithm and a regression model.

3.3.1 Weighted sum

The weighted sum method computes the predicted score of an item for a user by analyzing the other items that the user has rated that are similar to the item being considered. Each other rating is weighted according to its similarity to the new item and then is scaled by the sum of all similarities so it is within the same rating scale as other items. This is shown as

$$P_{u,i} = \frac{\sum_{n \in N_i} \text{itemSim}(i,n) \cdot r_{un}}{\sum_{n \in N_i} \text{itemSim}(i,n)}$$

where N_i is the set of items similar to i.

3.3.2 Regression

This method can work better than a weighted sum because the two item's vectors might be distant computationally but have a high similarity to users. To avoid this error, a linear regression model is created and used in place of the similar items' ratings in the weighted sum formula. This algorithm requires a model to be derived from the items available and can work better than a weighted sum prediction.

3.4 Experimental evaluation

3.4.1 Dataset

The MovieLens dataset was used for evaluation of the item-based algorithm by randomly selecting enough users to have 100,000 ratings from the dataset only considering users that have rated 20 or more movies [2]. The dataset was then separated into two sets, a training and a test set. The percentage of data to be used as the training set is represented by the variable x so that when $x = 0.8$, 80% of the data was used for training and 20% was used for testing. The dataset was converted into a user-item matrix A that had 943 rows and 1682 columns. Each row relates to one user and each column relates to one movie rated by at least one user. The sparsity level of the data sets was also considered and defined as $1 - \frac{\text{nonzero entries}}{\text{total entries}}$. The sparsity level of the MovieLens dataset is then $1 - \frac{100{,}000}{943 \times 1682}$, which is 0.9369.

3.4.2 Evaluation metrics

The accuracy of the item-based algorithms was computed using the *Mean Absolute Error* (MAE) between the predicted rating and the actual user rating. MAE is the measure of the deviation of recommendations, and the lower the number is, the more accurate the recommendations are. Each movie has a pair of ratings, (p_i, q_i) associated with it that represent the predicted rating and the acutal rating. MAE is calculated by taking each of these pairs and computing the absolute error between them, $|p_i - q_i|$. Each of these absolute errors are summed and then averaged:

$$\text{MAE} = \frac{\sum_{i=1}^{N} |p_i - q_i|}{N}$$

3.4.3 Experiment procedure

For the experiments, the data was first divided into training and test sets. Using the training set, each of the three similarity algorithm's parameters were tuned by testing different values for each. These tests were done by splitting the training set into its own training and test sets, so as to not derive

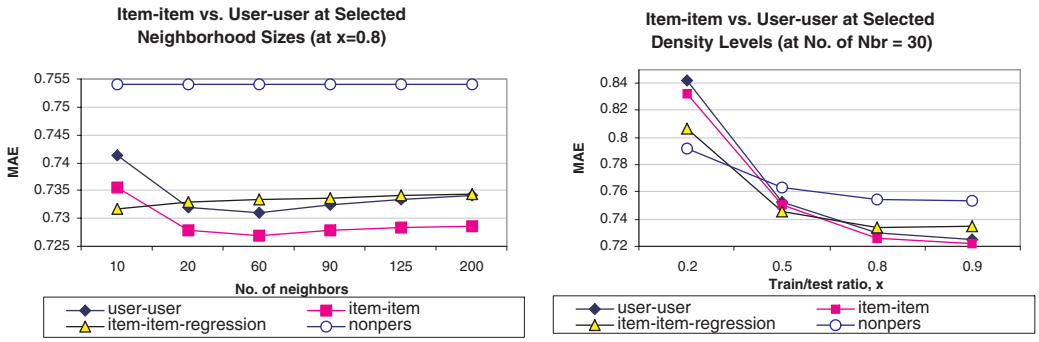

Figure 2: Comparison between different prediciton algorithms

any extra accuracy improvement from the data. The algorithms were then tested and the MAE was recorded. Each algorithm was tested ten times on randomly chosen training and test sets and the MAEs were averaged. To compare the performance of the item-based algorithms, a user-based algorithm was tested as well. This algorithm was a Pearson nearest neighbor algorithm, based on the best published one available.

3.4.4 Experimental results

The first thing tested was the accuracy of each of the three similarity algorithms mentioned earlier. Each algorithm was implemented to predict the similar items and then the weighted sum algorithm was used to generate predictions. The results showed that the adjusted cosine algorithm was the best performer and so it was used for the rest of the experiments. The next test was to determine what amount of the data set should be used for training and a value of 80% was found to be the optimal choice for both the weighted sum and regression methods. The neighborhood size, or the number of similar items used in the prediction computation showed that the weighted sum algorithm performance peaks at size 30 and the regression model diminishes in accuracy as the size is increased. The neighborhood size was set at 30 for best performance from both algorithms.

Comparing both prediction methods with the user-based algorithm was done by letting one of the parameters vary while setting the other at its optimal value. When varying x between 0.2 and 0.9 by 0.1 intervals, neighborhood size was set as 30, then x was set at 0.8 when varying neighborhood size between 10 and 200 at intervals of 10. Using these settings, the weighted sum algorithm out performed the user-based algorithm at all values of x (neighborhood size = 30) and all values of neighborhood size ($x = 0.8$), as shown in Figure 2, taken from [4]. The regression based algorithm showed better performance with low values of both x and neighborhood size than both other algorithms but lost its lead as parameters were increased. These results show that item-based algorithms can produce more accurate results than the standard user-based methods.

4. TEMPORAL DYNAMICS

When considering the methods that people use to rate things, it is not clear that anyone uses a set method for their entire interaction with a collaborative filter. A user could have a skewed rating system that tends to be above the median and then switch to rating closer to the minimum. This doesn't necessary mean that the items the user is rating have gotten worse, but could show a trend in that user's ratings. When looking at the NetFlix dataset in Figure 3 from [3], there is a clear trend that a movie's ratings tend to increase with the age of the movie. The more time between when the a user rates a movie and when it was released, the higher the average rating. This trend can help improve the accuracy of recommender systems if they could take temporal change into consideration.

4.1 Netflix prize and dataset

The Netflix Prize is a noteworthy competition that brought a lot of attention and research to the collaborative filtering field, both from academic and private interests. Netflix is a movie rental service that lends movies to customers via the mail and also has streaming services that allow customers to watch movies in places with an internet connection. Netflix's recommender system, Cinematch, was very successful at suggesting new movies to users and this accuracy was a main reason for customers to keep their subscriptions. In October 2006, Netflix announced the Netflix Prize to improve Cinematch's accuracy, they releasied a large data set to the public. With a $1,000,000 prize for the team that could reduce the root mean squared error (RSME) that Cinematch was capable of by 10 percent. Teams were given over 100 million ratings from 480,000 users on 17,770 movies. [1]

4.2 Parts of a temporal model

To incorporate into predictions, the time, t, at which a rating is made is recorded along with the user, u, and the item, i. A rating $r_{ui}(t)$ indicates the preference by user u of item i at day t and a predicted rating $\hat{r}_{ui}(t)$ is the predicted rating user u will give item i at time t. Each (u, i, t) triplet where $r_{ui}(t)$ is known is stored in the set $K = \{(u,i,t)|r_{ui}(t)\text{is known}\}$ and is used to give predicted ratings. To find $\hat{r}_{ui}$, each user u is associated with a vector $p_u \in R^f$ and each item i is associated with a vector $q_i \in R^f$ space. A rating is predicted by the rule:

$$\hat{r}_{ui} = q_i^T p_u$$

This rating does not do a good job at detecting effects that does not involve user-item interaction and so do not allow room for time to be included as a factor. To solve this a baseline rating is made by adding the average rating of every

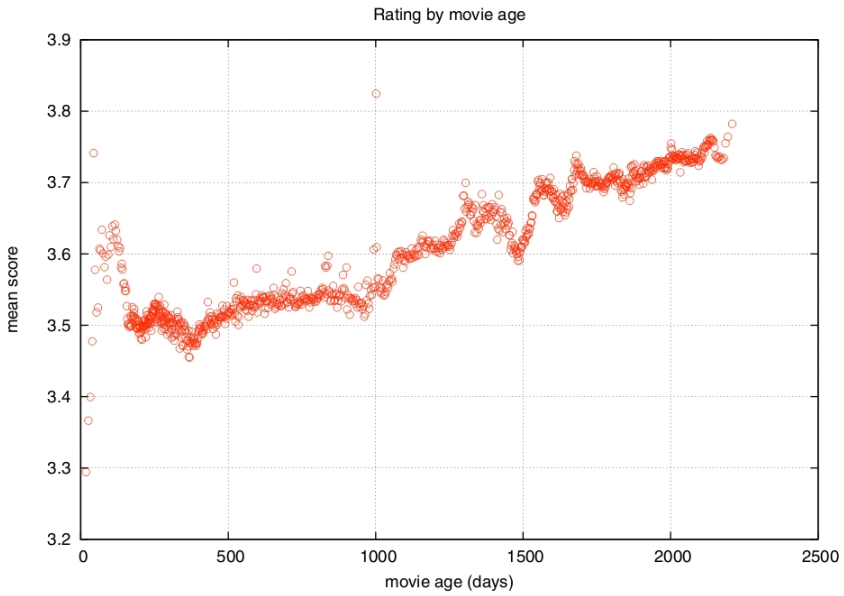

Figure 3: Movie rating by movie age

item and the deviations both the item and the user have from the average. We denote μ to be the overall average ratings of all items by all users and b_u and b_i to be the deviation for the user and item respectively. This creates the baseline for an unknown rating, b_{ui} to be:

$$b_{ui} = \mu + b_u + b_i$$

[3] explains this formula with an example:

> Suppose that we want a baseline estimate for the rating of the movie Titanic by user Joe. Now, say that the average rating over all movies, μ, is 3.7 stars. Furthermore, Titanic is better than an average movie, so it tends to be rated 0.5 stars above the average. On the other hand, Joe is a critical user, who tends to rate 0.3 stars lower than the average. Thus, the baseline estimate for Titanic's rating by Joe would be 3.9 stars by calculating 3.7 - 0.3 + 0.5.

Adding the predicted rating, $\hat{r}_{ui}$ into the baseline prediction, we get an equation that has the main effects isolated, allowing them to be modified later to take time as a factor. The combined equation is:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

Separating each of these factors allows each to be treated with a different temporal effect. User bias (b_u) changes over time, item bias (b_i) changes over time, and user preference (p_u) also changes over time. Meanwhile, the item characteristics, represented by the vector (q_i), would not be expected to change over time.

4.3 Baseline with temporal dynamics

The two parts of a baseline prediction that are affected by time are the item and user bias, b_i and b_u. An item's average rating might fluctuate depending on outside factors such as a sequel being made or an actor's change in popularity. A user might also change rating bias by setting a lower or higher standard for how they determine a movie's quality. Making both these parameters as functions of time, the baseline prediction becomes:

$$b_{ui}(t) = \mu + b_u(t) + b_i(t)$$

The function $b_{ui}(t)$ represents the baseline estimate of user u's rating of item i at day t. When finding $b_i(t)$, it is important to note that within the context of a movie recommender, movies do not change popularity on a daily basis and so a less specific view can be used. The dataset was divided into sections called bins that represent a time period. Each time period should be large enough for a movie's rating to be noticeably changed. Each time period will relate to a single bin and have a single bias amount $b_i(t)$. For the Netflix dataset, it was found that there is a wide range of bin sizes that resulted in the same accuracy and so 30 bins were used. These 30 bins span the entire dataset and each relates to ten weeks of data. A day t is associated with an integer $\text{Bin}(t)$ depending on which time period it is in. Then the temporal baseline is added to the baseline to form:

$$b_i(t) = b_i + b_{i,\text{Bin}(t)}$$

Applying this technique to users is not as simple due to the frequency that users rate items being low. A variety of functions can be created to model a user's rating with temporal dynamics, ranging from simple to complex. These different predictors are:

- *static*: no temporal effects: $b_{ui}(t) = \mu + b_u + b_i$
- *mov*: accounting only to movie-related temporal effects: $b_{ui}(t) = \mu + b_u + b_i + b_{i,\text{Bin}(t)}$
- *linear*: linear modeling of user biases: $b_{ui}(t) = \mu + b_u + \alpha_u \cdot \text{dev}_u(t) + b_i + b_{i,\text{Bin}(t)}$

- *spline*: spline modeling of user biases: $b_{ui}(t) = \mu + b_u + \frac{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|}} + b_i + b_{i,\mathrm{Bin}(t)}$
- *linear+*: linear modeling of user biases and single day effect: $b_{ui}(t) = \mu + b_u + \alpha_u \cdot \mathrm{dev}_u(t) + b_{u,t} + b_i + b_{i,\mathrm{Bin}(t)}$
- *spline+*: spline modeling of user biases and single day effect: $b_{ui}(t) = \mu + b_u + \frac{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|}} + b_{u,t} + b_i + b_{i,\mathrm{Bin}(t)}$

When compared with the static predictor, each algorithm shows improvement in accuracy with the *linear+* and *spline+* showing the greatest decrease in error.

5. CONCLUSION

As recommender systems become used more, improvements have been made on the standard approach to the point where the accuracy of a system depends on the traits of its unique dataset. A user-based system can be improved by implementing new features dependant on a particular dataset, but most recommenders have a peak performance and then don't improve too much. This calls for new ways to create collaborative filtering techniques that improve performance on a systematic level and not an implementation one. The two techniques discussed here, item-based algorithms and temporal aware models, show fairly large improvements on the standard user-based methods and are important to the future of recommender systems. Both of these systems are developed by recognizing traits about data and then creating a way to use the traits to improve accuracy. When looking for ways to improve recommender systems it becomes important to recognize attributes of a dataset such as the static nature of items and trends in user and item ratings. Using these observations, new techniques such as item-based algorithms and temporal dynamics may be developed.

6. REFERENCES

[1] R. M. Bell and Y. Koren. Lessons from the netflix prize challenge. *SIGKDD Explor. Newsl.*, 9:75–79, December 2007.

[2] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22:5–53, January 2004.

[3] Y. Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 447–456, New York, NY, USA, 2009. ACM.

[4] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[5] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. The adaptive web. chapter Collaborative filtering recommender systems, pages 291–324. Springer-Verlag, Berlin, Heidelberg, 2007.