Naïve Filterbots for Robust Cold-Start Recommendations

Seung-Taek Park[1], David Pennock[2], Omid Madani[1], Nathan Good[3], Dennis DeCoste[1]

[1]Yahoo! Research, 3333 Empire Ave, 2nd Floor, Burbank, CA 91504 USA
[2]Yahoo! Research, 45 West 18th Street, 6th Floor, New York, NY 10011 USA
[3]Yahoo! Research, 1950 University Ave., Suite 200, Berkeley, CA 94704 USA

{parkst,pennockd,madani,ngood,decosted}@yahoo-inc.com

ABSTRACT

The goal of a recommender system is to suggest items of interest to a user based on historical behavior of a community of users. Given detailed enough history, item-based collaborative filtering (CF) often performs as well or better than almost any other recommendation method. However, in cold-start situations—where a user, an item, or the entire system is new—simple non-personalized recommendations often fare better. We improve the scalability and performance of a previous approach to handling cold-start situations that uses filterbots, or surrogate users that rate items based only on user or item attributes. We show that introducing a very small number of simple filterbots helps make CF algorithms more robust. In particular, adding just seven global filterbots improves both user-based and item-based CF in cold-start user, cold-start item, and cold-start system settings. Performance is better when data is scarce, performance is no worse when data is plentiful, and algorithm efficiency is negligibly affected. We systematically compare a non-personalized baseline, user-based CF, item-based CF, and our bot-augmented user- and item-based CF algorithms using three data sets (Yahoo! Movies, MovieLens, and EachMovie) with the normalized MAE metric in three types of cold-start situations. The advantage of our "naïve filterbot" approach is most pronounced for the Yahoo! data, the sparsest of the three data sets.

Categories and Subject Descriptors: I.2.6 [Artificial Intelligence]: Learning

General Terms: Algorithms, Experimentation, Measurement, Performance

Keywords: Performance analysis, cold start, collaborative filtering, recommender systems, robustness, hybrid content and collaborative filtering, naïve filterbots

1. INTRODUCTION

A recommender system uses algorithmic means to churn through the available data of user preferences in order to suggest items of interest. For example, a recommender system for books would use the user's past ratings on books to find similar users, and determine algorithmically how much that user would like a book he/she has not read before. Recommender systems have been used for all types of information, from websites to CD-Roms, books, etc. Recommender systems are widely deployed on the web, appearing on everything from independent community-driven web sites to e-commerce powerhouses.

Once a substantial amount of preference data has been gathered about a particular user and his community, collaborative filtering (CF) algorithms are among the most effective recommendation algorithms. In particular, item-based CF [29], a simple variant of the very first user-based CF algorithms, is hard to beat when enough data is available. For example, many studies including [29, 13, 7, 22] demonstrate that item-based CF is as good or better than other approaches across a variety of settings in the movie domain.

However one situation when CF algorithms are less effective is when data is sparse, either because the target user is new to the system, an item is new, or both. In fact, in extreme cases, when data is very scarce, simple non-personalized recommendations based on global averages can perform better than CF algorithms. We study the behavior of a number of CF algorithms as data availability on users and items grows, examining three types of cold-start situations: cold-start user, cold-start item, and cold-start system. Our algorithm is a simple variation of the filterbot algorithm proposed by Good et al. [9], which we call the *naïve filterbot* algorithm. A filterbot algorithm injects pseudo users or *bots* into the system. These bots rate items algorithmically according to attributes of items or users, for example according to who acted in a movie, or according to the average of some demographic of users. Ratings generated by the bots are injected into the user-item matrix along with actual user ratings. Then standard CF algorithms are applied to generate recommendations.

In this paper, we examine how simple filterbots effect the performance of standard user-based (UB) and item-based (IB) collaborative filtering algorithms. Our approach improves on the scalability of Good et al.'s original filterbot approach. In previous work Good et al. saw improvement in performance from a set of 2N personal filterbots for N users and 20 genrebots. In our paper, we demonstrate improvement in performance with a smaller set of only a total of seven global filterbots. In addition to changing the number of filterbots, we changed the scale on which the filterbots rated movies from binary (high and low in Good et al.) to

KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.

average ratings. We also demonstrate that switching from Good et al.'s UB baseline to IB significantly improves accuracy. We then show that our naïve filterbots improve accuracy further, especially in cold-start settings, when users, items, or both are relatively new to the system, and thus have little data associated with them. The computational cost of employing 5-7 naïve filterbots is almost negligible compared to the baseline.

We analyze the performance of a non-personalized baseline recommender (AVG), traditional user-based (UB) and item-based (IB) CF algorithms, and our naïve filterbot algorithms in three cold start situations with three large data sets in the movie domain: Yahoo! Movies,[1] MovieLens,[2] and EachMovie.[3] We use normalized mean absolute error (NMAE) as our performance metric, as employed in [15]. Among the non-filterbot approaches, IB appears best across a wide range of data availability settings, confirming previous results. We find that AVG does outperform IB when the user-item matrix is extremely sparse. In general, our naïve filterbot algorithms either equal or beat their non-filterbot counterparts over all datasets and cold-start settings. The benefit of naïve filterbots grows as the training matrix becomes more sparse. The benefit of filterbots is most clear with the Yahoo! data set, which also happens to be the sparsest data set.

2. RELATED WORK

Collaborative filtering (CF) systems use community ratings to determine recommendations for users. CF systems typically work by associating a user with a group of like-minded users, called neighborhoods, then recommending items enjoyed by others in the group. The movie domain is probably the most widely studied application area for CF systems [9, 28, 29, 25, 17, 32], though the same methodology is applicable across a number of domains, including books, music [30], web pages [6], jokes [8], news, newsgroups [27, 14, 19], advertisements, and more.

CF algorithms range from simple nearest-neighbor methods [6, 27, 29] (Breese et al. [6] call these "memory-based" methods) to more elaborate model-based methods which first train or "compile" a model—for example, a Bayesian network [6], a classifier [5, 20], a co-clustered matrix [31], or a truncated singular value decomposition matrix [5, 8, 28]—based on historical data, then use the trained model to generate recommendations. Hundreds of CF algorithms have been proposed and studied, including machine learning methods [5, 4, 15, 26], graph-based methods [1, 12], linear algebra based methods [5, 8, 28], and probabilistic methods [11, 23, 24]. A number of hybrid methods combining information filtering (IF) and CF techniques have also been proposed [3, 2, 4, 24, 18, 9], which are especially useful when data is sparse, for example in cold-start situations. In fact, in the extreme cold-start setting, CF methods cannot provide recommendations at all, and IF methods or hybrid IF/CF methods are needed.

In general, given enough data, a straightforward item-based (IB) nearest-neighbor method [29, 13, 7] is hard to beat decisively. Park et al. [22] showed that the Fast MMMF algorithm, which has recently been cited as one of the best CF algorithms [26], does not beat IB. We base our naïve filterbot approach on Good et al.'s filterbot algorithm [9], a hybrid CF/IF approach. A filterbot is an automated agent that rates all or most items using IF techniques. These filterbots are then treated as additional users in a CF system.

[1]Yahoo! Movies, http://movies.yahoo.com/

[2]MovieLens, http://movielens.org/

[3]The data was available at http://www.research.compaq.com/src/eachmovie/ until October 2004 when it was finally retired.

3. DATA SETS

We use three movie ratings datasets: Yahoo! Movies, MovieLens and EachMovie. We use movie content information (e.g., cast, genre, synopsis, etc.) from Yahoo! Movies. All movie-ids in MovieLens and EachMovie were converted to the Yahoo! movie-ids to index the same movie content information. User ratings in the Yahoo!, MovieLens and EachMovie data sets are integers ranging from 1 (F) to 13 (A+), from 1 to 5, and from 0 to 5, respectively, where higher ratings mean "more preferred".

Due to the large number of experiments we run in this study, we choose to use smaller datasets. We use the 100K user rating set, which is sampled and provided by MovieLens. Then, we randomly select 10% of ratings for testing and the rest for training. For the EachMovie dataset, we first remove all of the "sounds awful" ratings since these ratings (which have a weight less than one) described users' impressions of the item but not their actual experience. Next, we sample 10% of users randomly, and randomly assign 10% of their ratings into test and the rest into training set. In our dataset, we choose to keep users who have rated less than 20 movies in the EachMovie data set. We believe that many users in real systems provide only a few ratings to test the system, and then never come back. In order to keep these users, increasing the initial accuracy seems very important. TheYahoo! dataset consists of two files in chronological order, where the test data was gathered after the training data. Because of this, the Yahoo! training and testing data sets have slightly different rating-frequency distributions.

Table 1 displays summary statistics of the three datasets. For example, the Yahoo! training data contains 211,327 ratings from 7,642 users for 11,916 movies. The average user rating ($\overline{R_u} = \frac{\sum_u \overline{r_u}}{|U|}$) is 9.639 and the average item rating ($\overline{R_i} = \frac{\sum_i \overline{r_i}}{|I|}$) is 9.325 where $\overline{r_u}$ and $\overline{r_i}$ are the average ratings of user u and item i. The average number of ratings ("votes") per user, denoted $\overline{V_u}$, is 27.65. The matrix density ratio ($\delta = \frac{100*|R|}{|U|*|I|}$) of the Yahoo! training data is 0.23, meaning that only 0.23% entries in the user-item matrix are filled. The Yahoo! dataset is more biased toward positive user preferences than the other data sets, with a greater frequency of top ratings and a higher average rating. Yahoo! users are also more likely express extreme preferences at either ends of the scale (13=A+ or 1=F), showing a bimodal ratings distribution. The Yahoo! dataset has the lowest density ($\delta = 0.23$) while the MovieLens has the highest density ($\delta = 6.3$).

4. SIMULATION PROTOCOLS

4.1 CF algorithms

We test three CF algorithms: user-based (UB), item-based (IB), and our naïve filterbot modifications of UB and IB. In this section, we briefly explain the first two algo-

Table 1: Data statistics: the number of ratings ($|R|$), the number of users ($|U|$), the number of items ($|I|$), the average number of votes per user ($\overline{V_u}$), the average number of votes per item ($\overline{V_i}$), the average user rating ($\overline{R_u}$), the average item rating ($\overline{R_i}$) and matrix density (δ).

	Yahoo! training	Yahoo! test	MovieLens, 1M	MovieLens, 100K	EachMovie	EachMovie, Sample
$\|R\|$	211,327	10,136	1,000,209	10,.000	2,811,983	254,321
$\|U\|$	7,642	2,309	6,040	943	61,265	6,174
$\|I\|$	11,916	2,380	3,652	1,682	1,607	1,461
$\overline{V_u}$	27.65	4.39	165.6	106.04	45.9	41.19
$\overline{V_i}$	17.73	4.26	273.88	59.45	1,749.83	174.07
$\overline{R_u}$	9.64	9.66	3.7	3.59	3.15	3.42
$\overline{R_i}$	9.32	9.54	3.24	3.08	2.3	3.23
δ	.23	.18	4.53	6.3	2.86	2.82

rithms. Our modifications will be explained with more detail in the Section 5.

4.1.1 The UB algorithm

UB first calculates the similarities between user ratings using the *pearson correlation coefficient* [6, 14, 27].

$$sim(u,v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \overline{r}_u) \cdot (r_{v,i} - \overline{r}_v)}{\sqrt{\sum_i (r_{u,i} - \overline{r}_u)^2} \cdot \sqrt{\sum_i (r_{v,i} - \overline{r}_v)^2}} \quad (1)$$

where $r_{u,i}$ is the rating of user u for item i and $\overline{r}_u$ is user u's average rating for items rated by both u and v. I_u is the set of items that user u has rated. It helps to penalize similarity scores that are based on a small number of overlapping items in order to reflect a lack a confidence, yielding a modified similarity score $sim'(u,v)$ as follows [10]:

$$sim'(u,v) = \frac{min(|I_u \cap I_v|, \gamma)}{\gamma} * sim(u,v) \quad (2)$$

We set $\gamma = 50$. Then, the predicted rating of item j for user u is calculated as:

$$p_{u,j} = \overline{r}_u + \frac{\sum_{v \in U} sim'(u,v) * (r_{v,j} - \overline{r}_v)}{\sum_{v \in U} |sim'(u,v)|} \quad (3)$$

where $\overline{r}_u$ is user u's average rating.

4.1.2 The IB algorithm

IB is very similar to UB. Instead of calculating similarities between users, it calculates similarities between items using *adjusted cosine similarity* [29, 16].

$$sim(i,j) = \frac{\sum_{u \in U} (r_{u,i} - \overline{r}_u) \cdot (r_{u,j} - \overline{r}_u)}{\sqrt{\sum_{u \in U} (r_{u,i} - \overline{r}_u)^2} \cdot \sqrt{\sum_{u \in U} (r_{u,j} - \overline{r}_u)^2}} \quad (4)$$

Note that when $\overline{r}_u$ is calculated, all items which user u has rated are considered. Similarities based on a small number of common items are penalized similarly to (2). The predicted rating of the item i for user u is:

$$p_{u,i} = \overline{r}_i + \frac{\sum_{j \in I_u} sim'(i,j) * (r_{u,j} - \overline{r}_j)}{\sum_{j \in I_u} |sim'(i,j)|} \quad (5)$$

where $\overline{r}_i$ denotes the average rating of item i.

Note that IB algorithm can be considered as a model based approach and it consists of two parts; item similarity computations (or model learning stage) and neighbor selection (or prediction calculation stage) using the the model. For example, IB first calculates item similarities and it can be done at offline. Then when a new user-item pair comes to the system, IB selects the top k nearest neighbors of the target item in the candidate set — items the target user has rated so far — by using the item similarities matrix. Then the prediction of the target item for the target user is given by the sum of the average rating of the target item and the weighted average of its neighbors.

4.2 Evaluation metrics

We use "normalized MAE" (NMAE), proposed in [15], as a performance metric. MAE [19, 6] can be calculated in two ways: *macro-averaged* and *micro-averaged*. Macro-average first calculates the mean absolute error of each user and averaging them over all users. Micro-averaged MAE averages errors over all ratings. From a system-wide perspective, a recommender service may want to reward users who provide more ratings. If so, micro-averaged metrics which put more weight on high-volume raters are more appropriate. For this reason, we use micro-averaged MAE for our cold-start system experiments. For our cold-start user and item experiments, we use macro-averaged MAE in order to better measure the average incremental effect of each new user/item. Then MAE is normalized by MAE_{random}, which denotes the expected MAE where predictions are randomly selected. If NMAE is smaller than 1, the algorithm works better than random. We used 4.824, 1.6, 1.944 as the MAE_{random} for Yahoo!, MovieLens and EachMovie.[4] Note that NMAE provides the similar performance scale over three different datasets and make comparison easier.

4.3 Cold-start setting

We measure prediction accuracy of AVG, UB, IB and our naïve filterbot variations of UB and IB in three cold-start situations: *cold-start system*, where a new service starts and the user-item matrix is extremely sparse; *cold-start user*, where a new user comes to the system and the system has little knowledge about the user; and *cold-start item*, where a new item comes to the system and the system has little knowledge about the item.

4.3.1 Cold-start system

To analyze performance variations of CF algorithms in cold-start settings, we first generate training sets of varying degrees of sparsity. We randomly select ratings from the original training data with probability of γ and use them for training algorithms. We increase γ from 0.1 to 1. We

[4]We empirically calculate MAE_{random} for Yahoo!, but use the same MAE_{random} in [15] for MovieLens and EachMovie.

Table 2: Average density ratio ($\bar{\delta}$) of sparse training data in cold-start system setting.

γ	Yahoo!	MovieLens	EeahMovie
0.1	0.082	0.81	0.431
0.2	0.1	1.397	0.67
0.3	0.12	1.986	0.917
0.4	0.139	2.558	1.115
0.5	0.156	3.122	1.396
0.6	0.173	3.709	1.619
0.7	0.189	4.259	1.856
0.8	0.204	4.819	2.088
0.9	0.228	5.366	2.325
1	0.232	5.946	2.557

generate 10 different sparse training data for each γ with 10 different seed numbers. Table 2 shows the average density ratio and the average number of users/items in the sparse training data while γ increases. For each test, bots use only user ratings in the sparse training data. The same user demographics, content information of items and test data are used for all test. All results in this environment are averaged over 10 runs, each with a different collection of randomly selected ratings used for training.

4.3.2 Cold-start user

To study the effect of a new user on CF algorithms, we select users who have rated more than 40 items in the training data and at least 1 item in the test data. We select 432 users from Yahoo!, 612 users from MovieLens, and 1,845 users from EachMovie. Then, we generate 5 different test-user sets, each of which includes 20% of those users. We remove all ratings of test users in the training data, then filterbot ratings and item similarities are calculated based on the given matrix. In UB, we keep ratings of non-test users and one test user in the memory and user similarities between the target and only non-test users arer calculated. Then, in each successive round of testing, we randomly add two more of the target user's ratings back into the training matrix. Each algorithm computes predictions for all of the target user's test ratings. The process repeats, restoring two more training ratings to the training set at each step. The results shown in this setting are the average of five different test-user data.

4.3.3 Cold-start item

Similar to cold-start user, we select items that have been rated by more than 40 users in the training data and by at least 1 user in the test data as test items. We select 803 test items from Yahoo!, 651 items from MovieLens, and 702 items for EachMovie. Then, we divide items into 5 test sets. We remove all ratings of test items in the training data and ratings of content-based bots are calculated based on the given matrix. In IB, when 2 more ratings are added, item similarities between the target and other items are recalculated. The system keep ratings of only non-test items for each user, thus the size of the candidate sets will not change. In UB, user similarities are calculated with only training data and will not be recalculated during the test. Similar to cold-start user, in each successive round of testing, we randomly add two more of the target item's ratings back into the training matrix. Each algorithm computes predictions for all of the target item's test ratings. We do not use award and average critic rating information in this experiment since this information is generally not available for a new movie. The results shown in this setting are the average of five different test-item data.

5. NAÏVE FILTERBOTS

Our algorithm is a variation of Good et al.'s [9] filterbot algorithm discussed briefly in Section 2. We were interested in applying filter-bots to the item-based algorithm, which seems to be the best state of the art CF algorithm. In this section, we discuss the problems and limitations of the previous approaches and describe our solution to address these problems. Please refer [22] for more details.

5.1 The effect of bots on IB and UB

A bot can be generated as either an artificial user or item. One example of user-bots are RipperBots in [9]. RipperBots generated personalized ratings of items based on the item features and user profiles. An ActionBot, on the other hand (which rates 'action' movies) can be considered as an item which generates ratings of all users based on their ratings of action movies. Once the filterbots are defined, we inject their ratings into the system by treating them just like any other users or items, applying either UB or IB to calculate predictions. When any CF algorithm fails to generate a prediction for item i, we use the item's average rating as a default prediction. Note that user and item-bots affect IB and UB differently. In IB, user-bots only have an effect on the learning model — the item similarity matrix — but do not increase the size of the candidate set. Since more item similarities can be defined due to addition of "pseudo" users, more neighbors for the target item can be chosen from the candidate set. On the other hand, item-bots affect on the size of the candidate set rather than the learning model itself. In UB, user-bots increase the size of the candidate set — users who have rated the target item — and the item-bots effect on user similarities.

5.2 Injecting critic ratings and feature bots

The first bots we consider are the critic ratings. We select 42 critics[5] who have rated more than 10 movies and insert their ratings directly to the user-item matrix. We find that even though 42 critics seems to be useful for Yahoo! they cause significant performance degradation on MovieLens and EachMovie. Since we consider a media as a critic, ratings of a critic may not be consistent.

We also consider feature-bots, which generate item ratings based on the features of items such as genre and casting information. We inject various feature-bots into the user-item matrix and test their performance. Unfortunately, the performance of this approach is often worse. For example, when we inject many user feature-bots into the matrix, they are often useful for IB when the items have been rated by very few users. In this case, item similarities are mainly calculated based on pseudo ratings generated by bots. However, when items have been rated by many users, it can cause a problem. Note that bots generate ratings of most items and those pseudo ratings becomes major factor for the item similarities even though the item has been rated by many users.

[5]Note that we consider a media, e.g. *New York Times*, as a critic rather than an individual reviewer, e.g. *Stephen Holden*, to increase the coverage of each critic.

Thus, item similarities often overfit contents too much. We observe that item feature-bots are useful for the IB and user-feature bots are useful for the UB in the cold-start setting where the number of bots is large. In both cases, bots increase the size of the candidate sets rather than changing the similarity model. More detailed results are shown in the [22].

5.3 Bot criteria

Even though user-bots are useful for UB and item-bots are useful for IB, their limited scalability can be a critical barrier to adoption by many real world systems, as users and item-bots significantly increase the size of the user-item matrix. Thus, we define three criteria, which our bot-augmented approach should meet.

- *Coverage:* If a bot represents a pseudo user, it should be able to rate most of the items. If a bot represents a pseudo item, ratings of most users should be calculated based on some sort of user information. If a bot does not meet this requirement, it may be useless in the cold start situation.
- *Scalability:* The injection of the bots should not decrease the system's overall scalability significantly. Since most recommender systems are used online, the scalability of the system should be considered.
- *Performance:* The injection of the bots should improve recommendation quality in cold-start situations while it performs at least as good as IB and UB when enough user ratings are provided. In the other words, predictions should not overfit pseudo ratings when the system already have enough information. This criteria comes from our experience that users' own ratings seems to be the best resource to predict their preferences on other items.

5.4 Injecting GBots

Based on three criteria, we reject personal-bots because of the increased computational complexity. Similarly we reject user and item feature bots. Instead of them, we propose 7 global user-bots (GBots) based on aggregate rating information and item content information. AVGBot generates ratings based on average item ratings over all users. VTBot generates the rating of an item i according to: $r_i = \log_\beta V_i$, where V_i is the number of users who have rated item i. β is a normalization factor that caps ratings at the maximum available rating (13 for Yahoo!; 5 for MovieLens and EachMovie). We set $\beta = 2$ for Yahoo! and $\beta = 4$ for MovieLens and EachMovie. Critic-bot (CRBot) generates item ratings based on their average critic ratings. Award-bot (AWBot) first partitions items based on how many awards they have won (won_i) or nominated ($nomi_i$) such as $Cl_i = int((won_i + 0.5 * nomi_i)/3)$. Then, the rating of each item is generated based on the average rating of the items in the partition that the item i belongs to. Actor-bot (ATBot) first calculates ratings of actors over all users. The rating of an actor is the average rating of movies in which the actor has starred. Then, it generates the rating of movie i based on the average of actor ratings who starred in the movie i. Here, we only consider the first five featured actors among the movie's cast. Director-bot (DRBot) and Genre-bot (GRBot) generate ratings of items similarly to ATBot.

To meet scalability criterium, we try to keep the number of bots as small as possible. If we only inject a few bots into the user-item matrix, additional computation complexity is almost negligible. Also, it will minimal effect on the item similarities when enough ratings are available. For user-based CF algorithm, a user can have only a few "pseudo" neighbors among the top 50. Thus, it provides at least competitive performance with the original IB and UB when the user-item matrix is dense.

6. EMPIRICAL RESULTS IN COLD-START

6.1 Cold-start system

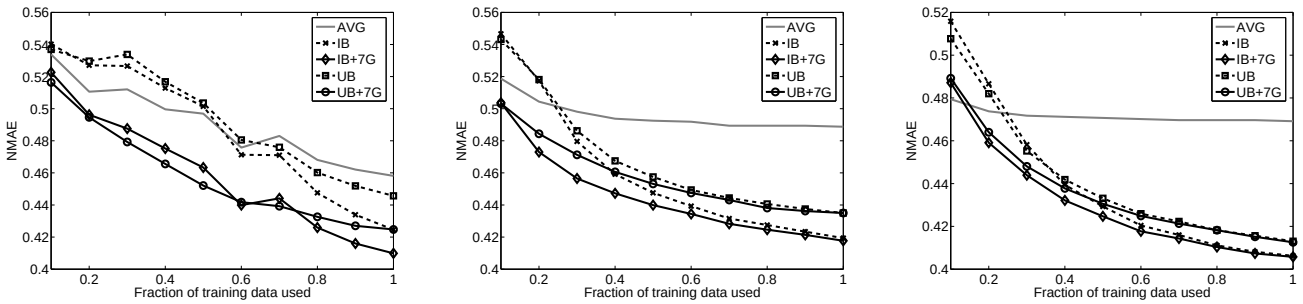
Figure 1 shows NMAE changes of five different recommendation algorithms while the density of the training matrix increases. We do not compare the performance of the original filter-bot algorithm[9] with others due to its limited scalability. Note that we need to run each algorithm 273 times (10 for each $.1 \leq \gamma \leq .9$ and 1 for $\gamma = 1$ with three different data) and it is almost impossible to conduct this experiment with the original algorithm.

Where the matrix is extremely sparse, AVG is more robust than IB and UB. For example, AVG provides better average prediction than IB and UB when $\gamma \leq 0.5$ for Yahoo! and $\gamma \leq 0.2$ for MovieLens and EachMovie. Also, where the training matrix is very sparse (i.e., $\gamma = 0.1$ for Yahoo! and MovieLens and $\gamma \leq 0.3$ for EachMovie), UB provides better predictions than IB. However, the advantage of UB disappears rapidly in all three data sets as the density of the training data increases. Our bot-augmented algorithms with seven GBots (IB+7G and UB+7G) always improve the performance of both IB and UB in cold-start. In general, performance improvements of our naïve filter-bot algorithms become larger as γ decreases and IB+7G outperforms UB+7G. The only exception is when Yahoo! is used, which is the most sparse data among the three. When $\gamma \leq 0.7$, UB+7G is slightly better than IB+7G on Yahoo! data.

6.2 Cold-start user

When a new user comes to the system, the user affects IB and UB differently. The ratings of a single user do not effect IB's item similarities, since the model is already built offline. If a new user rates more items, it increases the size of candidate set for both IB and IB+7G. Note that additional bots only effect the model but do not increase the size of the candidate set. Thus, when the matrix is sparse such as with the Yahoo! dataset, IB+7G shows better performance than IB due to better similarity computation. The improvements will continue while the user rates more items. However, where the matrix is dense, the performance improvements of IB+7G may not be clear because item similarities have not changed signifigantly. The results shown in the Figure 2 confirm our reasoning. Note that among the three datasets, items in EachMovie have been rated by 174 users while items in Yahoo! have been rated by 18 users on average.

On the other hand, ratings of a new user significantly effect user similarities in UB and UB+7G. If the user rates more items, the system can calculate more accurately user similarities because of the additional information. Addition of bots in UB slightly increase the size of the candidate set. We expect that UB+7G shows increased performance improvement when the user has rated very few items. When



(a) Yahoo! (b) MovieLens (c) EachMovie

Figure 1: NMAE changes while the size of training data increases

the user have rated more items, the user similarities in UB and UB+7G will improve, and as the candidate set increases and the effect of bots will be negligible. The result shown in the Figure 2 confirms our hypothesis. With Yahoo!, UB+7G shows performance improvements over all periods due to the matrix sparsity. In this case, the additional candidates, who rate most of items, are chosen as neighbors in most cases and those bots actually help to generate more accurate predictions for the given user-item pairs. With MovieLens and EachMovie, UB+7G shows performance improvements only when a user has rated very few items. One interesting observation is that when a user has voted only 2 items, all algorithm show similar performance due to the size of the candidate set being very small.

6.3 Cold-start item

When a new item has received more ratings, item similarities in IB will improve. When the item has been rated by few users and the matrix is dense as in EachMovie, the addition of bots may cause inaccurate item similarities because the similarities largely depend on "pseudo" ratings rather than users' "real" ratings . However, when the new item has received enough ratings, the effect of bots will disappear or even improve the quality of predictions. With Yahoo! where the matrix is very sparse, the system often ends up failing to find neighbors in the candidate set due to lack of information. In cases where there datasets are very sparse and there are no neighbors to correlate with, item similarities will be based solely off of the bot ratings.

In UB algorithm, more ratings of new items will increase the size of the candidate set. Since additional bots only provide a small number of additional candidates, the effect of bots will be clear only when the item has been rated by a few users. When more users rate the item, the size of the candidate set will be larger and the effect of the bots will disappear. Similar to the cold-user setting, the effect of the bots is lessoned or worse when an item has been rated by very few users. It seems that "pseudo" ratings are more useful when they are used as supplements of user ratings rather than by themselves.

7. CONCLUSIONS AND FUTURE WORK

We conduct performance analyses of five CF algorithms: AVG, IB, UB and our "naïve filterbot" variations of IB and UB in three different cold-start environments. We use three data sets from Yahoo!, MovieLens and EachMovie and the performances of algorithms are measured by NMAE. Our filterbot algorithms clearly demonstrate better robustness than UB and IB in all three cold-start situations. The advantage of our algorithm is more clear if we use Yahoo! data, which is the most sparse among the three data sets. We see our main contribution as a detailed study of a number of different filter-bot generation methods and demonstration that a very few number of simple ("naïve") filterbots help collaborative filtering algorithms work better in cold-start situation, with negligible impact on non-cold-start recommendation accuracy and system efficiency. In the future, we plan to develop a new algorithm which exploits user implicit data such as user pageview history and search logs. Also, we plan to study how our filterbot algorithm reacts when attacks are introduced to the system. We plan to deploy our algorithm within MAD6 [21], a personalized movie search engine developed at Yahoo! Research. We plan to make our Yahoo! data available to academic researchers in the near future.

8. ACKNOWLEDGMENTS

We thank Yahoo! Movies, GroupLens (MovieLens), and Digital Equipment Corporation (EachMovie) for providing valuable movie ratings and content data.

9. REFERENCES

[1] C. C. Aggarwal, J. L. Wolf, K.-L. Wu, and P. S. Yu. Horting hatches an egg: a new graph-theoretic approach to collaborative filtering. In *ACM KDD*, pages 201–212, 1999.

[2] M. Balabanovic and Y. Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[3] J. Basilico and T. Hofmann. A joint framework for collaborative and content filtering. In *ACM SIGIR*, 2004.

[4] C. Basu, H. Hirsh, and W. W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *AAAI/IAAI*, pages 714–720, 1998.

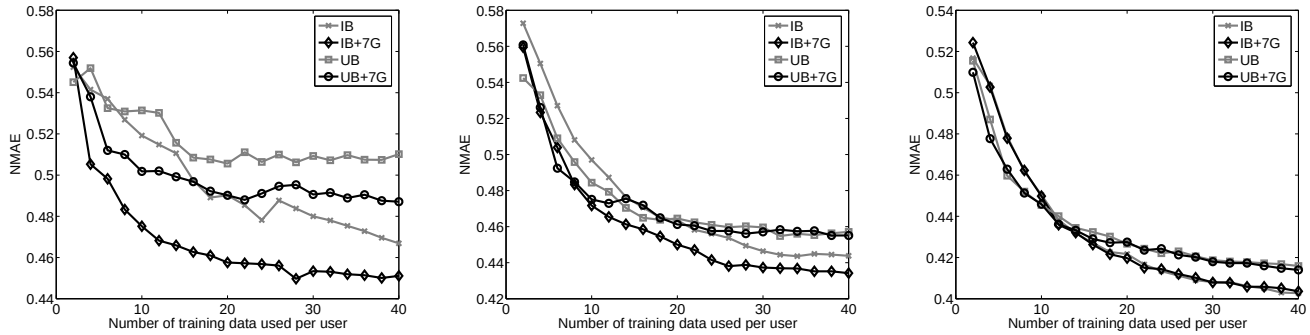
[5] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *ICML*, pages 46–54, 1998.

[6] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI*, pages 43–52, 1998.

[7] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM TOIS*, 22(1):143–177, Jan 2004.

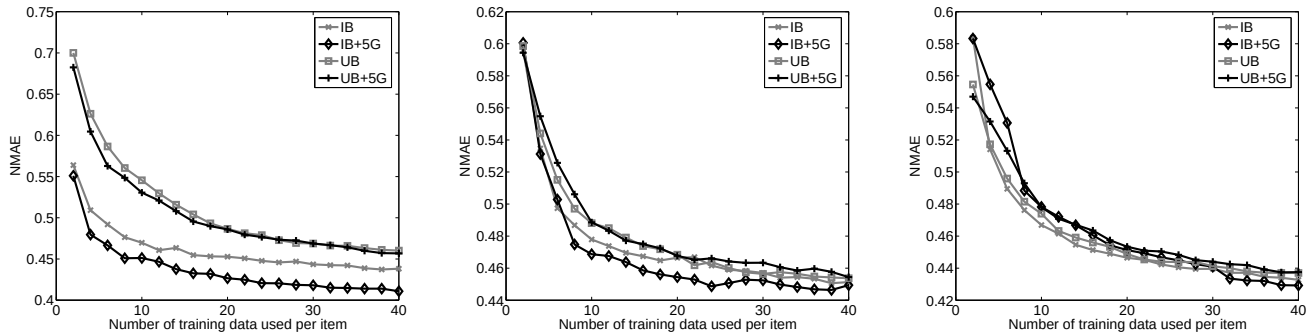
[8] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[9] N. Good, J. B. Schafer, J. A. Konstan, A. Borchers, B. M. Sarwar, J. L. Herlocker, and J. Riedl. Combining collaborative



(a) Yahoo! (b) MovieLens (c) EachMovie

Figure 2: NMAE changes while the number of ratings from a new user increases



(a) Yahoo! (b) MovieLens (c) EachMovie

Figure 3: NMAE changes while the number of ratings for a new item increases

filtering with personal agents for better recommendations. In *AAAI/IAAI*, pages 439–446, 1999.

[10] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *ACM SIGIR*, pages 230–237, 1999.

[11] T. Hofmann and J. Puzicha. Latent class models for collaborative filtering. In *IJCAI*, pages 688–693, 1999.

[12] Z. Huang, H. Chen, and D. Zeng. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM TOIS*, 22(1):116–142, Jan 2004.

[13] G. Karypis. Evaluation of item-based top-n recommendation algorithms. In *CIKM*, pages 247–254, 2001.

[14] J. A. Konstan, B. N. Miller, D. Maltz, J. L. H. L. R. Gordon, and J. Riedl. GroupLens: applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[15] B. Marlin. *Collaborative filtering: A machine learning perspective*. Master's thesis, University of Toronto, Computer Science Department.

[16] M. R. McLaughlin and J. l. Herlocker. A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In *ACM SIGIR*, pages 329–336, 2004.

[17] S. McNee, S. Lam, J. Konstan, and J. Riedl. Interfaces for eliciting new user preferences in recommender systems. In *UM*, pages 178–188, 2003.

[18] P. Melville, R. Mooney, and R. Nagarajan. Content-boosted collaborative filtering. In *AAAI*, 2002.

[19] B. N. Miller, J. T. Riedl, and J. A. Konstan. Experience with grouplens: Making usenet useful again. In *USENIX annual technical conference*, pages 219–231, 1997.

[20] K. Miyahara and M. J. Pazzani. Collaborative filtering with the simple bayesian classifier. In *PRICAI*, pages 679–689, 2000.

[21] S.-T. Park, D. M. Pennock, and D. DeCoste. Applying collaborative filtering techniques to movie search for better ranking and browsing. In *AAAI Workshop on Intelligent Techniques for Web Personalization (ITWP 2006)*, 2006.

[22] S.-T. Park, D. M. Pennock, O. Madani, N. Good, and D. DeCoste. Naive filterbots for robust cold-start recommendations. Technical report, YRL-2005-058, Nov 2005.

[23] D. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles. Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach. In *UAI*, pages 473–480, 2000.

[24] A. Popescul, L. Ungar, D. Pennock, and S. Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *UAI*, pages 437–444, 2001.

[25] A. Rashid, I. Albert, D. Cosley, S. Lam, S. Mcnee, J. Konstan, and J. Riedl. Getting to know you: Learning new user preferences in recommender systems. In *IUI*, pages 127–134, 2002.

[26] J. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *ICML*, 2005.

[27] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *ACM CSCW*, pages 175–186, 1994.

[28] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender systems–a case study. In *ACM WebKDD Workshop*, 2000.

[29] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001.

[30] U. Shardanand and P. Maes. Social information filtering: Algorithms for automating "word of mouth". In *CHI*, 1995.

[31] L. Ungar and D. Foster. Clustering methods for collaborative filtering. In *Workshop on Recommendation Systems at AAAI*, 1998.

[32] M. R. W. Hill, L. Stead and G. Furnas. Recommending and evaluating choices in a virtual community of use. In *ACM CHI*, pages 194–201, 1995.