

# Shilling Recommender Systems for Fun and Profit

Shyong (Tony) K. Lam

John Riedl

GroupLens Research  
Computer Science and Engineering  
University of Minnesota  
Minneapolis, MN 55455  
{lam,riedl}@cs.umn.edu

## ABSTRACT

Recommender systems have emerged in the past several years as an effective way to help people cope with the problem of information overload. One application in which they have become particularly common is in e-commerce, where recommendation of items can often help a customer find what she is interested in and, therefore can help drive sales. Unscrupulous producers in the never-ending quest for market penetration may find it profitable to *shill* recommender systems by lying to the systems in order to have their products recommended more often than those of their competitors. This paper explores four open questions that may affect the effectiveness of such shilling attacks: which recommender algorithm is being used, whether the application is producing recommendations or predictions, how detectable the attacks are by the operator of the system, and what the properties are of the items being attacked. The questions are explored experimentally on a large data set of movie ratings. Taken together, the results of the paper suggest that new ways must be used to evaluate and detect shilling attacks on recommender systems.

## Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval; H.3.4 [Information Storage and Retrieval]: Systems and Software; H.3.5 [Information Storage and Retrieval]: Online Information Services

## General Terms

Experimentation, Algorithms

## Keywords

collaborative filtering, recommender systems, shilling

## 1. INTRODUCTION

People face a bewildering number of choices when looking for items that they are interested in. A seemingly never-ending flood of content is available, but certainly not enough time exists to evaluate all possible choices. This, in a nutshell, is the problem of information overload. In recent years, recommender systems have emerged as one tool that can help people overcome this problem and quickly locate items to consume. These systems use opinions about items in some information domain in order to make

Copyright is held by the author/owner(s).  
WWW2004, May 17–22, 2004, New York, New York USA.  
ACM 1-58113-844-X/04/0005.

recommendations to a user regarding which items she may find interesting. One instance of a recommender system is *MovieLens* (<http://www.movielens.org>). GroupLens, our research group, operates this recommender system, which makes personalized recommendations suggesting movies that a user might like based on movies that she has seen and has expressed an opinion about.

While recommender systems are clearly beneficial to users, they can also be a valuable asset to retail companies in helping their customers find things that they might want to buy and, in effect, increasing not only sales, but perhaps also cross-sales and customer retention. This is particularly true in the realm of e-commerce. For example, *Amazon.com* has made many recommender systems available to their customers. These range from manually operated recommenders where users can recommend items to other users by writing reviews or creating lists, to automated systems where the site generates a list of recommended items based on what the user has looked at recently or has purchased in the past.

Producers of items (manufacturers, authors, etc.) would like their products to sell well in the marketplace. With recommender systems, there is a natural motivation to want one's own products to be recommended more often than those of a competitor. Of course, one way to accomplish this is to produce quality goods that people like and regard highly. However, unscrupulous producers may opt to take a more deceitful route; they may try to influence recommender systems in such a way that their items are recommended to users more often, whether or not they are of high quality. An instance of a company generating false "recommendations" to consumers arose in June 2001 when Sony Pictures admitted that it had used fake quotes from non-existent movie critics to promote a number of newly released films.<sup>1</sup> The online retailer *Amazon.com* has found that their recommenders are prone to some level of abuse on at least two different occasions.<sup>2,3</sup> Also, *eBay*, which uses a recommender system as a reputation mechanism, has found itself continually dealing with users who subvert the system in various ways, including purchasing good ratings (feedback) from other members in order to bolster their own reputations.<sup>4</sup>

One way to influence a recommender system is to arrange to have a group of users (human or agent) enter the system and vouch for the items in question. These users become *shills*, whose false opinions are intended to mislead other users. Shills pose a serious threat to users and operators of recommender systems. They may cost users time and money by recommending bad items. They may

<sup>1</sup><http://news.bbc.co.uk/1/hi/entertainment/film/1368666.stm>

<sup>2</sup><http://www.wired.com/news/ebiz/0,1272,53634,00.html>

<sup>3</sup><http://news.com.com/2100-1023-976435.html>

<sup>4</sup><http://www.auctionbytes.com/cab/abn/y03/m09/i17/s01>

cost operators by degrading the user’s level of trust in the recommender system and the retailer behind it.

This paper focuses on recommender systems that use automated collaborative filtering (ACF) to generate recommendations. ACF is a class of algorithms commonly used in the implementation of recommender systems. These algorithms operate on the basis that similar users have similar tastes; thus, if people similar to you can be located, then the items they enjoy are likely to be ones you will also enjoy. These algorithms normally have two modes of operation: prediction and recommendation. In the *prediction* mode, the algorithm simply predicts how much a user will like some item or set of items. The items may have been selected by the user through browsing or searching. In the *recommendation* mode, the algorithm produces an ordered list of items that it believes the user is most likely to enjoy. This distinction will become important as we explore the practical effects of attacks on recommender systems.

The rest of the introduction comprises related work and the statement of hypotheses, setting the stage for the experimental work in the rest of the paper.

## 1.1 Related Work

### 1.1.1 Recommender Systems

One of the earliest instances of a collaborative filtering based recommender system is *Tapestry* [6]. In 1994, Resnick et al. [16] automated the collaborative filtering process and introduced an ACF algorithm based on k-Nearest-Neighbor. Since then, a number of improvements to kNN have been proposed [9, 8]. The *user-user* algorithm we study in this paper is a tuned version of the original kNN algorithm. Another related algorithm that we study is a tuned version of the item-based k-Nearest-Neighbor [18]. We call this algorithm the *item-item* algorithm in this paper. Many other recommendation algorithms have been developed as well, including ones based on singular value decomposition [17], Bayesian networks [2], and factor analysis [3].

Besides *MovieLens*, a myriad of other recommender systems exist, particularly on e-commerce sites. Schafer [19] examines and categorizes a large set of these commercialized recommender systems. In addition, numerous recommenders in a variety of domains have been developed for research purposes, including *GroupLens* (Usenet news) [11], *Ringo* (music) [20], and *Jester* (jokes) [7].

### 1.1.2 Attacks on Recommenders

While a large body of work exists on recommender system algorithms, less attention has been devoted to exploring and improving their resistance to attacks. Dellarocas [5] outlines several attacks on reputation systems used in online trading communities such as *eBay* and proposes a predictive algorithm similar to existing collaborative filtering algorithms which helps minimize the effect of the attacks. Canny [3] presents a system in which user preferences (ratings) are kept private both from the recommender system’s administrators and from other users, which he believes can mitigate attacks where the system administrators are involved (e.g. a retailer being paid to place a company’s items highly on recommendation lists).

More recently, O’Mahony et al. [14] have performed empirical studies of the resistance of the kNN user-user algorithm to attacks based on injecting a number of shill users into the system. The attacks were shown to be successful both at *push*-ing items by raising predicted ratings, and at *nuke*-ing items by lowering predicted ratings. Furthermore, they present a theoretical analysis of the effect of noise – perhaps injected by shills – on the performance of ACF algorithms and perform several experiments with real-world data

sets to evaluate the generated models. Our paper builds on their work by including more recommender algorithms and by evaluating the attacks on recommendations as well as on predictions.

### 1.1.3 Impact of Successful Attacks

Psychologists have shown experimentally that people tend to conform with the opinions of others, even when those opinions may be incorrect. This was the case in the classic conformity study by Asch [1] where test subjects were asked to perform a simple task in a small group. The other members of the group were working for the experimenter and deliberately gave incorrect responses. Even though the correct choice was easy to see, the test subjects agreed with the group and made the wrong choice in one-third of all trials.

A similar effect is likely to occur in interactions between computers and people. Nass and Moon [13] find that people tend to treat computers as they would treat people – in one of their experiments, they show that people are less likely to criticize a computer’s performance if that computer asks for the evaluation than if another computer asks.

Cosley et al. [4] find that users are indeed affected by manipulated predictions provided by a recommender. As suggested by the conformity studies mentioned above, users tend to rate items toward the predicted rating provided by the recommender regardless of prediction accuracy. Whether this represents a genuine change in opinion is unknown – it might just be that users conform in what they say, not in what they believe – but the result indicates that it may very well be possible that some users can be fooled into accepting a bad recommendation. Further, the manipulated predictions experiment suggests that shilling may be doubly dangerous, since affected users may rate towards the manipulated predictions, influencing other users who trust them.

## 1.2 Hypotheses

The previous research on shilling has demonstrated that shilling should be of concern in recommender systems, but leaves several important questions unanswered. The first of these questions is about how effective shilling is on finely tuned versions of the user-user [9] and item-item [18] algorithms. Though the algorithms are based on fundamentally similar ideas about relationships between users and items, their implementations are sufficiently different that we suspect they may exhibit different behavior under shilling attack. Formally:

**HYPOTHESIS 1.** *Different ACF algorithms respond differently to shilling attacks.*

This hypothesis has importance for designers of ACF algorithms, who may be able to design algorithms that are more resistant to attack, for operators of recommender systems, who may be able to select algorithms that are resistant to likely attacks, and for evaluators of shilling attacks, who may need to be aware their results are algorithm dependent. We suspect there will be differences in shilling resistance among other less similar recommender algorithms as well; if we find differences between these two algorithms we will recommend future work to understand in detail the shilling resistance of the entire suite of known recommender algorithms.

Recommender systems in e-commerce, where shilling is most often feared [5, 14], are used more often to produce recommendations [19] than predictions. Attacks should be judged based on how they affect the recommender in its most common mode of operation – after all, if a user only looks at the top 10 items on a list, does it matter that the shilling attack has changed the prediction she would have seen for the 500th item on her list?

**HYPOTHESIS 2.** *Shilling attacks affect recommender algorithms differently from prediction algorithms.*

If this hypothesis is supported, it will mean that researchers who evaluate attacks must base their measure of the effectiveness of the attack on the specific ways the targeted recommender system is being used. Metrics used in the past for evaluating recommender algorithms have all been focused on predictions; perhaps for recommendation tasks recommendation metrics should be used instead.

The fact that past research has shown that shilling is effective in some cases, raises the question of whether the operators of recommender systems can detect that their systems are under effective shilling attack. Our next hypothesis is that they cannot do so with existing tools. Note that this hypothesis is distinct from hypothesis 2 even though both are about metrics. Hypothesis 2 is about measures that evaluate shilling attacks based on knowing exactly which items are being attacked, which the operator of the recommender system will not know.

**HYPOTHESIS 3.** *Shilling attacks are not detectable using traditional measures of algorithm performance.*

ACF algorithm designers often utilize metrics such as Mean Absolute Error (MAE) to evaluate the overall predictive accuracy of their algorithms and to compare it with other algorithms. Thus, it may be tempting to use such metrics to detect attacks by looking for changes in algorithm quality caused by attacks. We believe this will generally not be possible; that is, attacks can be subtle and focused enough that their overall effect on the system is minimal.

This hypothesis is unnerving to those who – like us – run a recommender system, since it means that our systems may already be under successful attack, and that despite all of the measurement tools at our disposal we may be unaware of all but the crudest attacks. We seek to spur understanding of which evaluation techniques are best for detecting shilling attacks.

One possible place to look for detecting shilling attacks is to understand which target items are most vulnerable to them. We hypothesize that the number of ratings of an item, and the spread of those ratings over possible ratings values influence the extent to which an attack can succeed.

**HYPOTHESIS 4.** *Ratings distribution of the target item influences attack effectiveness.*

We will study popularity (number of ratings), likability (average rating), and entropy as the variables that describe the ratings distribution. We believe that the following statements will be true:

- Likability – the more well-liked an item is (that is, the higher its average rating is), the easier it is to cause that item to be recommended more often
- Popularity – the less popular an item is (that is, the fewer the number of ratings it has), the easier it is to manipulate the predictions and recommendations for that item
- Entropy – the higher the entropy of an item’s ratings, the easier it is to manipulate the predictions and recommendations for that item

### 1.3 Contributions

This paper builds on the work presented in [14] and further explores the feasibility and effectiveness of influencing recommender systems based on ACF algorithms through shill attacks.

First, we propose a set of dimensions that describe and categorize a wide variety of shill attacks. These dimensions set the stage for

the types of evaluation we carry out, though our evaluation so far is only of the dimensions we expect to be most important in practice.

Second, we build on past work to develop two basic attack types, and perform a series of experiments to study their effectiveness in influencing both the predictions and the recommendations made by two different ACF algorithms. In our experimental work we simultaneously evaluate all three aspects of the shilling attack: the attack itself, the ACF algorithm under attack, and the different metrics used to evaluate both the algorithms and the attacks.

Finally, we present the results of the experiments and examine how they support or refute our hypotheses, and conclude with a look at some open questions and possible future work.

## 2. DIMENSIONS OF ATTACKS

In this paper we only consider *shill attacks* where the attacker’s available action is to introduce a new set of users and a set of ratings made by those new users to the recommender system. In the case of online recommender systems, more traditional attacks such as denial-of-service, password cracking, system hacking, and bribery are possible, but are beyond the scope of this paper.

Each shill attack has a number of intrinsic properties that can be useful in describing and comparing different attacks. A list of these properties, or dimensions, follows.

### 2.1 Attack Intent

Different shill attacks may have very different intents. While the direct result of a shill attack is generally that the predictions made to users are manipulated in some way, the eventual goal of a shiller can be one of several alternatives. Two straightforward intents are to “push” one or more items in the system in order to have them recommended to more users and, conversely, to “nuke” a set of items to cause them to be recommended to fewer users.

Another possible intent is to simply damage the recommender system as a whole; that is, to reduce prediction and recommendation quality across the board with the goal of causing users to stop trusting the system and eventually to stop using it. A successful attack of this nature might benefit competing recommender systems.

### 2.2 Targets

Shill attacks can be directed at a particular subset of users and a subset of items in a recommender system. It is in an attacker’s best interest to restrict the effect of an attack to a small target set of items in order to be more subtle and try to avoid detection by the system operators. Additionally, it might be beneficial to also restrict the effect to some desirable set of target users. For instance, it might induce suspicion to cause a heavy metal album to be recommended to a connoisseur of classical music who would have absolutely no interest in such an album.

### 2.3 Required Knowledge

Attacks may require some level of knowledge about the items, users, ratings, and algorithms in the recommender system being attacked. An informed attack will generally be more effective than an uninformed attack. Further knowledge about the system such as ratings sparsity, ratings distribution, and algorithm parameters can help in choosing which attack to employ and in tuning attack parameters to maximize effectiveness and minimize detectability.

### 2.4 Cost

A shill attack has an associated cost value that depends on the level of effort and information needed to successfully execute the attack. With a cost dimension and a suitable means of evaluating attack effectiveness, one might be able to evaluate attacks on a

cost/benefit basis to determine if it is economically worthwhile to execute an attack. The following factors contribute to the cost of a given attack:

- Size of attack: the number of new users and ratings.
- Difficulty of interacting with the recommender system. For instance, an attack on a system that employs anti-automation techniques such as CAPTCHAs may have a far higher cost than an attack on a system that does not [21].
- Obtaining required knowledge about the algorithm, users, items, and ratings in the recommender system.
- Any other resources required for attack planning or execution, such as additional logistical, computational, or technical requirements.

## 2.5 Algorithm Dependence

Some shill attacks may be specifically designed to exploit a particular weakness in a specific algorithm or class of algorithms, while others might be more general and can be effective against a variety of algorithms. More specific attacks will intuitively require fewer resources for the same effectiveness, but require detailed knowledge of the algorithm being used and its parameter settings.

## 2.6 Detectability

Inherent properties of attacks may make them more or less easily detectable, both to users and operators of the recommender system. In general, an attacker would like to be less detectable to operators in order to be able to sustain the attack for as long as possible before being discovered and stopped. The importance of detectability to users depends on the attack intent. This dimension is one that can evolve very rapidly as shill detection methods are developed or improved, similar to the current arms race seen in spam (unsolicited commercial email) detection and detection evasion.

# 3. EXPERIMENTAL DESIGN

## 3.1 Our Shill Attack Design

This paper focuses on algorithmic attacks that attempt to push or nuke an item (that is, raise or lower the recommender's predictions for the item) by introducing shills into the system. It is assumed that the attacker does not have access to the ratings matrix but can obtain broad statistical measures of the ratings data and other publicly available data. We do not concern ourselves with reducing the cost of the attacks or with designing attacks that are difficult to detect. If these brute force attacks prove difficult for operators to detect, recommender systems operators should be concerned indeed!

We begin with the type of attacks used in [5]. These attacks inject a collection of new users into the system, each of which has rated a set of items to try to be similar to existing users, and has rated the particular item being attacked very high in order to push it. Practical implementations of both user-user and item-item algorithms scale correlations according to the number of ratings in common [9]. It has been observed that the attack from [5] does not work well for this variant of the kNN algorithms if too few items are rated by the shill users because the similarities are scaled down too far to be considered by the algorithm [14]. We therefore modify the attack to rate *all* movies in the system to maximize the number of items in common between shill users and real users. Attacks like these are related to *filterbots*, which also rate all items [8]. However, filterbots are used to improve recommendation coverage and

quality, while these shills ("shillerbots," perhaps) will be used to decrease quality for a small subset of items.

One way in which the ACF algorithms we test are different from those used in the past [5, 14] is that we do not use negative correlations between items. We made this decision because our past experience has been that negative correlations often lead to recommendations that are inconsistent with user preferences. Shills might exploit negative correlations to produce very strong attacks – but system operators will likely disable the use of negative correlations to improve quality first.

Our attacks target the entire population of users and a small target set of items while requiring relatively limited knowledge about the ratings matrix. The number of new shill users introduced to the recommender system is varied between 25 and 100. The intent of these attacks is to either push or nuke the target set of movies. The two attack methods developed are:

### 3.1.1 RandomBot

RandomBots are a naive attack in which each introduced user rates items not in the target set randomly on a normal distribution with mean 3.6 and standard deviation 1.1. These values are chosen because they represent the ratings distribution in the data set. Even if these values are not known to the attacker, they can be estimated relatively easily, perhaps by observing people using the recommender system and obtaining a sample of their ratings. A normal distribution is used to approximate the observed user rating behavior in *MovieLens*. To accomplish its objective, the filterbot rates items in the target set with value equal to either the minimum or maximum allowed rating, depending on its intentions (nuke or push, respectively).

### 3.1.2 AverageBot

AverageBots are a somewhat more sophisticated attack than RandomBots and require knowledge of the average rating of each item in the system. A number of recommender systems, including *MovieLens*, will readily provide this information. Furthermore, such aggregate information about users' preferences may be found from other sources. In the case of movies, the *Internet Movie Database* (<http://www.imdb.com>) publicly displays the average user ratings of listed movies.

Each introduced user rates items not in the target set randomly on a normal distribution with mean equal to the average rating of the item being rated and standard deviation 1.1. The rationale behind this attack is that providing ratings centered around the average rating will help the filterbot be more similar to existing users, and thus, have a larger effect on the recommendations. As with the RandomBot, items in the target set are rated with a minimal or maximal value depending on the intent.

## 3.2 Data Set

A data set derived from *MovieLens* consisting of 999,799 ratings on 3,404 movies by 7,463 users is used in all experiments in this paper. All ratings are integral values between 1 and 5, inclusive, where 1 represents a poor movie and 5 represents an excellent one.

## 3.3 ACF Algorithms

We experimented with two commonly-used ACF algorithms – kNN user-user and kNN item-item.

### 3.3.1 kNN User-User

The classic kNN user-user algorithm introduced by Resnick in [16] is chosen because it is considered to be a good baseline algorithm and is widely used in both academia and industry.

The user-user algorithm uses the following formula to compute a predicted rating  $p$  for a user  $u$  on an item  $i$ .

$$p_{u,i} = \bar{r}_u + \frac{\sum_{v \in U_{u,i}} [w_{u,v}(r_{v,i} - \bar{r}_v)]}{\sum_{v \in U_{u,i}} |w_{u,v}|}$$

Here,  $\bar{r}_u$  is user  $u$ 's average rating over all rated items,  $w_{u,v}$  is the mean-adjusted Pearson correlation ("similarity") between users  $u$  and  $v$ , and  $U_{u,i}$  is user  $u$ 's neighborhood with respect to item  $i$  and consists of the  $k$  users who have rated  $i$  and have the greatest Pearson correlation with  $u$ .  $k$  is a tunable parameter and represents the number of neighbors.

Several optimizations and suggested parameters from [9] are used. In particular, we set  $k$  to 20 and use the  $n/50$  significance weighting and deviation from mean optimizations. A similarity threshold of 0.1 is also used. The only difference between the user-user variant used in this paper and the published algorithm is that only positive similarities are considered here; negative similarities and ones under the threshold are ignored. This variant was also used for several years in the *MovieLens* recommender system and does not appear to be detrimental to performance.

### 3.3.2 kNN Item-Item

The kNN item-item algorithm was introduced by Sarwar et al. [18] and is similar to user-user in both definition and predictive accuracy. However, it computes and uses similarities between items rather than users. We choose to experiment with this algorithm in order to explore how well an attack that is effective against user-user operates with a somewhat-different algorithm. The formula used to compute a prediction in item-item is:

$$p_{u,i} = \frac{\sum_{j \in \text{allsimilaritems}} [s_{i,j} * r_{u,j}]}{\sum_{j \in \text{allsimilaritems}} |s_{i,j}|}$$

Here,  $s_{i,j}$  is the similarity between items  $i$  and  $j$ . The algorithm implementation used in this paper is the *MultiLens* recommender engine developed by Brad Miller [12], which uses the adjusted cosine method of computing similarity, and considers the 20 rated items with highest similarity to be the set of "all similar items" (again, ignoring negative similarities). A tuned version of this algorithm is currently utilized in *MovieLens*.

## 3.4 Methods

A total of twenty-four experiments were performed in a 2x2x2x3 design. The algorithm (user-user or item-item), attack type (AverageBot or RandomBot), attack intent (nuke or push), and number of new users/bots (25, 50, or 100) were varied in each experiment.

The target set for the experiments consists of 22 items. This set was selected to include a variety of different movie types including future releases, new releases, obscure films, popular films, controversial films, and long-standing favorites. In terms of ratings properties, this selection of items represents a wide range of popularity (number of ratings), entropy (a measure of the variance of ratings), and likability (mean rating). Table 1 displays the properties of items in the target set.

## 3.5 Metrics

There are two things that we would like to be able to measure:

1. How much the overall accuracy of the ACF algorithm is affected by an attack.
2. How effective an attack is in accomplishing its goal.

**Table 1: Properties of movies in chosen target set. Ratings is total number of ratings (popularity), mean is average rating (likability), and entropy is the standard information-theoretic entropy of the ratings distribution. Recall that ratings are provided on a 5-point scale.**

Item	Ratings	Mean	Entropy
1	0	N/A	N/A
2	7	2.57	0.99
3	8	2.88	1.56
4	17	3.00	2.04
5	17	3.24	1.61
6	18	2.11	1.99
7	27	3.15	2.21
8	34	4.00	1.53
9	46	1.78	1.67
10	48	2.60	2.18
11	92	3.17	2.15
12	105	1.72	1.65
13	106	2.71	2.12
14	116	4.12	1.59
15	234	2.44	2.17
16	263	1.89	1.82
17	354	4.28	1.60
18	422	3.20	2.26
19	1298	3.78	1.68
20	1828	4.49	1.39
21	2316	3.35	2.18
22	2654	3.74	2.08

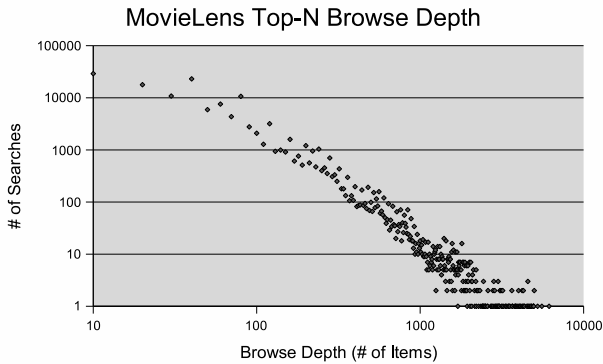
We consider many metrics here. The first time we introduce a metric, the name will be in **bold face**.

To address the first requirement, we turn to a metric commonly used to evaluate ACF algorithms, **Mean Absolute Error**, or MAE. This is simply the average absolute difference between the predicted rating and actual rating over all users and items in a test set. The algorithm performance is evaluated before and after each attempted attack to determine each attack's effect on the system as a whole.

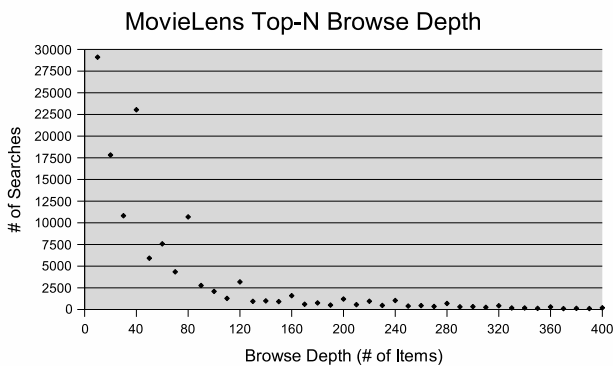
While MAE is generally considered to be a standard in evaluation of ACF algorithms, we find that there is a mismatch between what it measures and what really matters to users in a recommender system. Recommender systems are a decision support tool, and in general, the exact predicted value is of far less importance to a user than the fact that an item is recommended.

In practice many recommender systems are used by e-commerce sites to help their customers find products to purchase. One might expect that many of these systems produce lists of recommendations for their users, rather than interfaces that require the user to explicitly seek out the products that she desires. In fact, Schafer et al. found that by far the most common output of e-commerce recommender engines was suggestions of items for the customers to purchase [19]. Therefore, the quality of the *recommendations* should be emphasized rather than the quality of the predictions.

Furthermore, since users do not tend dig very "deeply" when shown a list of things to examine, only the items near the top of a recommendation list should be considered when evaluating algorithm quality. We analyzed *MovieLens* user behavior from the search logs, and discovered that the median recommendation search ends within the first 40 items displayed. Figures 1 and 2 show the browse depth of 137,991 *MovieLens* recommendation (top- $N$ )



**Figure 1: Log-log graph of top- $N$  search browse depths in *MovieLens*. The median search depth is 40, and 79% of recommendation searches end at or before 80 items.**



**Figure 2: Partial graph of top- $N$  search browse depths in *MovieLens*. The long tail to the right of 400 items is not shown.**

searches. Figure 2 is truncated at 400 items so that the low median is easier to visualize. Figure 1 shows the entire set of data on a log-log scale so that the long tail is visible. The phenomenon that users do not dig deeply through a list of results may be even more acute in other domains. For instance, one study found that 54 percent of users view only a single page of search results in each session [10]. Developing a top- $N$  recommendation accuracy metric for entire recommender systems is beyond the scope of this paper; however, we will develop an attack effectiveness metric that is centered around recommendations rather than predictions.

Before we do so, we first examine the attack effectiveness metrics used in previous work. One metric that is introduced in [14] is the **Stability of Prediction** metric, which measures the relative number of predictions for target items that are *not* manipulated beyond some given threshold. This metric does not directly address the notion of manipulating top- $N$  recommendation lists. It treats all prediction manipulations of equal amount as being of equal importance. However, this is intuitively not the case – for instance, on a 5-point rating scale, causing a prediction to change from 2 to 3 is far less meaningful than moving it from 4 to 5. An item with a 3-point prediction is usually far less likely to appear on a recommendation list than an item with a 5-point prediction.

Another metric also introduced in [14] is the **Power of Attack** metric, which is defined as the average change in prediction toward some target value (usually the minimum allowed rating  $r_{min}$  or the maximum allowed rating  $r_{max}$ ) over all target users and items. We believe that this has the same drawbacks as the Stability of Predic-

tion metric. Since Power of Attack is defined *differently* in earlier work [15] by the same authors, we will refer to this metric as **Prediction Shift** when presenting our results.

We will use the version of the metric defined in [15] when referring to **Power of Attack**. This is the percentage of predictions for target items *not* manipulated to some target value (again,  $r_{min}$  or  $r_{max}$ ). For attacks with a push intent, we find that this metric is somewhat useful for gauging impact on recommendation lists for attacks with a push intent, even though it is a prediction-based metric. An item that has its predicted rating manipulated to  $r_{max}$  is indeed likely to appear in a top- $N$  list.

However, this metric falls short in a number of ways. First, the metric is less suitable for measuring the effectiveness of attacks with a nuke intent. In the vast majority of cases, it is unnecessary to manipulate a prediction to  $r_{min}$  to cause it to not appear in a top- $N$ . Secondly, some algorithms are extremely conservative in making predictions of  $r_{max}$ . This metric is less able to accurately gauge attack performance in this case, as the values the metric takes will be universally high (remember: high values mean most items *not* successfully manipulated). Conversely, some algorithms may be fairly liberal in producing predictions of  $r_{max}$ . This metric does not take into account the possibility that more than  $N$  items can have a predicted rating of  $r_{max}$  – in such cases, the top- $N$  recommendation list is not well-defined and the method used to break such “ties” to determine which  $N$  items to display is implementation-dependent.

So, we are left with a need to define a metric that directly measures the effect of an attack on top- $N$  recommendation lists. Furthermore, the metric should take into account the possibility of items tied for inclusion in these lists. Based on these requirements, we propose a metric called **Expected Top- $N$  Occupancy** (ExpTop $N$ ). The metric is defined as the expected number of occurrences of *target items* in a top- $N$  recommendation list, measured over all users, assuming that the displayed ordering of items tied at any particular rank is random. We believe that in the absence of any information about the algorithm implementation, the metric should treat all possible tiebreaker functions equally, which has the beneficial side effect of leading to a metric that gives a lower value for attacks that result in many ties involving target items.

For example, let the target items be items  $E$  and  $F$ , and suppose that the candidates for inclusion in a top-5 recommendation list for some user are as shown below.

Rank	Item	Pred
1	$B$	5.0
2	$E$	4.9
3	$D$	4.7
4	$A$	4.5
4	$C$	4.5
4	$F$	4.5

In this case, there are three items tied for the fourth and fifth entry in the top-5 list. One of these three items is in the target set. Assuming all orderings of these three items are equally likely, the Expected Top-5 Occupancy for this user is  $1.\overline{666}$ . The target item  $E$  in the top-5 contributes 1, and item  $F$  being in two out of the three permutations for the fourth and fifth items displayed on the list contributes 0.666.

By examining the change in this metric caused by some attack, one can determine the attack’s effect as perceived by users of the recommender system. To reflect actual *MovieLens* usage, we will use  $N = 40$  in accordance with our system usage analysis. To compare this metric with the prediction-centric ones, we will report MAE as well as the value of both Power of Attack metrics. The

**Table 2: Effect of attacks as measured by the Prediction Shift metric (PredShift) and the change in MAE ( $\Delta$ MAE). An increase in MAE indicates lower overall predictive accuracy.**

Algorithm	Intent	Attack	Bots	PredShift	$\Delta$ MAE
User-user	Push	Random	25	0.499	0.002
			50	0.671	0.004
			100	0.830	0.009
		Average	25	1.032	0.006
			50	1.189	0.011
			100	1.300	0.019
	Nuke	Random	25	0.422	0.002
			50	0.589	0.004
			100	0.759	0.010
		Average	25	0.656	0.007
			50	0.815	0.014
			100	0.956	0.023
Item-item	Push	Random	25	0.030	0.002
			50	0.053	0.002
			100	0.069	0.004
		Average	25	0.363	0.002
			50	0.426	0.004
			100	0.471	0.010
	Nuke	Random	25	-0.046	0.002
			50	-0.069	0.002
			100	-0.092	0.004
		Average	25	0.332	0.003
			50	0.354	0.006
			100	0.361	0.014

earlier definition from [15] will be referred to as Power of Attack (POA) and the later definition from [14] will be referred to as Prediction Shift (PredShift).

#### 4. RESULTS AND DISCUSSION

The discussion of results is organized according to the hypotheses they relate to.

**HYPOTHESIS 1.** *Different ACF algorithms respond differently to shilling attacks.*

Table 2 shows how each of the attacks affected the predictions made by the ACF algorithms. This table reports the two prediction-centric metrics, PredShift and change in MAE. These metrics are most appropriate for applications that form predictions for items selected by the user. First, we examine the attack effect as measured by the prediction shift metric. We delay discussion of MAE until the presentation of the hypothesis about detecting shilling, because we found MAE not very useful for the other hypotheses. In general the MAE changes were very small, and seem unlikely to represent noticeable change to users.

The user-user algorithm responds very strongly to all attacks; that is, the attacks are successful to some extent in manipulating the predictions for items in the target set. For the push attacks, the PredShift metric indicates that AverageBot-based attacks are able to raise the predictions by over one point, and RandomBot-based attacks are somewhat less effective. A similar pattern is seen for the nuke attack.

On the other hand, the item-item algorithm responds far less strongly. According to PredShift, the *most* effective push attack on item-item (100 AverageBots) has an effect on predictions that is

**Table 3: Effect of attacks as measured by the Power of Attack metric (POA) and percent change in Expected Top-40 Occupancy (ExpTop40). For POA, a lower value means the attack was more effective. The value of Expected Top-40 Occupancy pre-attack is 0.57 for the user-user algorithm and 0.24 for item-item.**

Algorithm	Intent	Attack	Bots	POA	ExpTop40
User-user	Push	Random	25	0.900	711%
			50	0.865	1190%
			100	0.816	1649%
		Average	25	0.715	1286%
			50	0.609	1674%
			100	0.519	1918%
	Nuke	Random	25	0.943	-39%
			50	0.928	-33%
			100	0.908	-32%
		Average	25	0.963	-67%
			50	0.952	-70%
			100	0.943	-75%
Item-item	Push	Random	25	1.000	150%
			50	1.000	171%
			100	1.000	229%
		Average	25	0.999	158%
			50	0.999	154%
			100	0.999	117%
	Nuke	Random	25	0.954	146%
			50	0.954	204%
			100	0.954	333%
		Average	25	0.955	-33%
			50	0.955	-54%
			100	0.954	-71%

comparable to the *least* effective push attack on user-user (25 RandomBots). RandomBots have an even weaker effect on item-item predictions – no average prediction shift is greater than one-tenth of a point.<sup>5</sup>

Next, we turn to Table 3, which shows the results of the attacks in terms of metrics that measure their ability to affect recommendations. These metrics are more appropriate for applications that generate lists of suggested items for their users.

According to the POA metric, push attacks on the user-user algorithm are more successful than nuke attacks, with AverageBot being superior to RandomBot. In the MovieLens domain push attacks may be easier because the average rating is higher than the midpoint of the scale. With the item-item algorithm, push attacks seem ineffective – the values of 1 and 0.999 indicate that essentially no predictions for items in the target set were successfully manipulated to the target predicted value of  $r_{max}$ , or 5. Nuke attacks on item-item are roughly as effective as they are on user-user, according to POA.

The ExpTop40 metric should be sensitive to prediction changes that influence the top 40 items, whether or not those changes move the predictions to the top of the scale. With this metric, we see a striking difference in response between the two algorithms. The

<sup>5</sup>Note that using RandomBots in a nuke attack actually *increases* predictions for the item slightly. We are unsure why this happens, though we have been able to show in small-scale examples that attacks on one user in item-item sometimes result in a reverse effect for other users. That is, if an item is pushed for some users, that item is nuked for other users. Perhaps the increase is due to a reversal on a subset of users.

user-user algorithm is profoundly affected by push attacks, with AverageBot again being more successful than RandomBot. Note that a 100-AverageBot attack causes a nineteen-fold increase in how often the target items are recommended in the top-40!

Clearly, the push attacks have a far more subdued effect on the item-item algorithm according to ExpTop40. In the strongest case, ExpTop40 increases by 229% with a 100-RandomBot push attack. Unexpectedly, we see that a 100-RandomBot *nuke* attack has an even greater effect, causing target items to be recommended over three times as often! So, while there is a large effect on item-item recommendations in a relative sense, it may not be considered substantive enough to be worthwhile for an attacker. Even after the strongest attack, less than one occurrence of a target item appears in each user’s top-40 on average.

Looking at the nuke attacks, we see that the AverageBot-based attacks are successful on both ACF algorithms in manipulating the top-40. With 100 AverageBots, the number of items from the target set appearing in a user’s top-40 is reduced by about 75% on the user-user algorithm and 71% on the item-item algorithm.

So, we find that the user-user and item-item algorithms do respond differently, particularly in how their recommendations are affected under push attacks. We judge that the evidence supports the hypothesis that different algorithms respond differently to shilling attacks. Algorithms that are less similar than user-user and item-item may react even more differently.

*HYPOTHESIS 2. Shilling attacks affect recommender algorithms differently from prediction algorithms.*

To address this hypothesis, we compare the values of the PredShift, POA, and ExpTop40 metrics. PredShift and POA are prediction metrics, while ExpTop40 is a recommendation metric. For many of the tests both classes of metrics move in the same direction. For instance, under a AverageBot-based push attack on user-user, PredShift is at least a half point, POA shows that at least thirty percent of predictions were successfully manipulated to  $r_{max}$ , and the change in ExpTop40 shows that many target items are pushed into the top 40.

However, there are notable differences, too. For instance, The PredShift metric indicates that RandomBot-based push attacks have a nearly negligible effect on item-item’s predictions for target items. The ExpTop40 metric, on the other hand, says that up to twice as many target items are in the top 40 after this attack.

Conversely, with Average-bot based push attacks on item-item, PredShift shows that there is a measurable effect on the predictions, but the effect on recommendations as measured by ExpTop40 are minimal. Thus, this attack does appear to have different effects on the prediction and recommendation modes of item-item.

Note that in a few cases, the POA metric shows diametrically opposed results in comparison with the other metrics. In the RandomBot nuke attacks on item-item, PredShift and ExpTop40 show that the predictions and recommendation frequencies for target items *increased*, while POA indicates some success in reducing predicted values for target items to  $r_{min}$ , or 1. In fact, POA indicates just as much success for this attack as it does for the AverageBot nuke attack on item-item, which PredShift and ExpTop40 agree actually does nuke the target items. POA is unable to distinguish between AverageBot and RandomBot attacks on item-item even though each attack has different effects on the predictions and recommendations according to the other metrics.

The high values POA takes for attacks on the item-item algorithm is a result of one of the weaknesses we mentioned earlier. We have observed in previous experiments that item-item tends to be more conservative than user-user in giving very high or very low

predictions. Thus, it is difficult to manipulate a prediction to either extreme of the scale, and as a result, the POA metric gives deceptively high values for item-item. Because of this deficiency, we recommend against using POA as the sole metric for evaluating shilling attack performance.

Overall, the evidence for this hypothesis is mixed. Usually, all of the metrics we studied move in the same direction, and it is hard to directly compare them. However, in a few cases there are clear differences between an attack’s impact on predictions and on recommendations. We believe these cases are sufficient evidence to argue that selection of an appropriate metric is important. If recommendation is more important than prediction for the recommender system, then a recommendation-based metric such as Expected Top- $N$  Occupancy should be used. On the other hand, if prediction is more important, a prediction-based metric such as Prediction Shift should be used.

*HYPOTHESIS 3. Shilling attacks are not detectable using traditional measures of algorithm performance.*

We return to Table 2 where the last column shows the change in MAE caused by each attack. The MAEs are obtained by performing five-fold cross-validation on a 80%/20% test/train split of the data set before and after each attack. Note that no attack increases the mean absolute error by more than 0.023, and that RandomBot attacks on item-item induce extremely small reductions in accuracy. These changes seem small, but some of them are comparable to the improvements in MAE that are often considered “significant” differences between algorithms [9].

As an aside, it is unclear whether *people* are able to perceive such differences in system accuracy. Cosley et al. [4] show that users are able to detect intentionally-produced shifts of one point on many items in a small set of predictions, but the average change in error here is over 40 times smaller and is distributed across a large set of items. Furthermore, the system’s user interface may not make small changes in prediction evident – for instance, *MovieLens* only displays predictions in half-point increments. Of course, whether users can detect changes in MAE is not directly related to whether MAE can be used by system operators to detect shills.

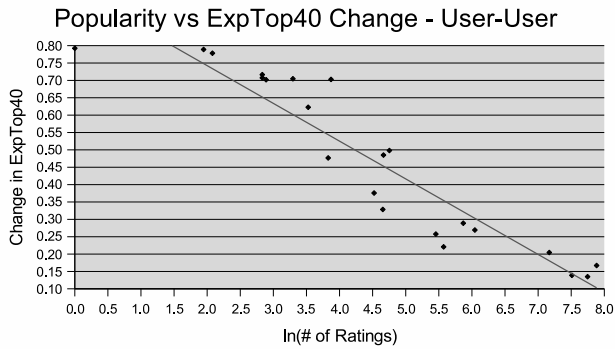
Overall, we find that insufficient evidence exists to support or reject this hypothesis. Some less naive attacks may fall “under the radar,” but others may be detectable by watching standard metrics such as MAE. Still, we believe that this ambiguity suggests that other ACF algorithm performance metrics are necessary to reliably detect attacks.

*HYPOTHESIS 4. Ratings distribution of the target item influences attack effectiveness.*

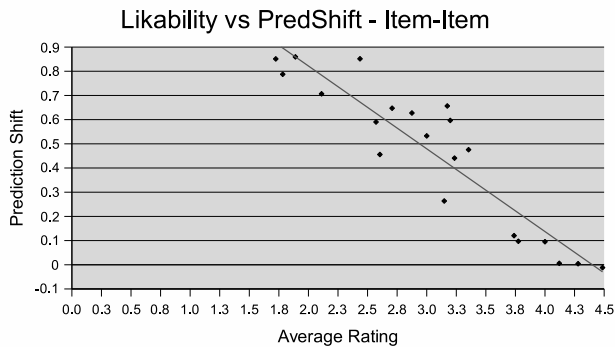
We hypothesized that the properties of an item’s ratings distribution has an effect on how much impact an attack has on that item. Recall that the properties mentioned earlier were popularity, likability, and entropy. Intuitively, if an item has few ratings (low popularity) and/or has a high spread of ratings (entropy), it should be easier to manipulate the predictions and recommendations for that item because it is more “volatile” in some sense. Likewise, items that are already well-liked should be easy to push, while items that are disliked by many should be easy to nuke.

We examined the effects of these variables on PredShift and ExpTop40. We only considered push attacks for ExpTop40, since so few of our target items were in top 40 lists prior to the attacks, so nuke attacks would have few items to “nuke.”

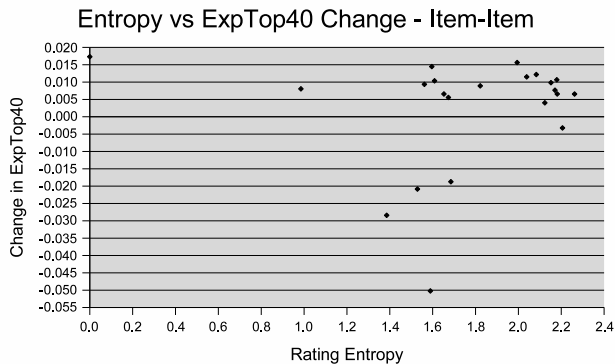
Figures 3, 4, and 5 show several examples of these relationships (or lack thereof). The likability of an item correlates with PredShift



**Figure 3: Relationship between popularity of an item and the effect of a 100-AverageBot push attack on user-user recommendations ( $r^2 = 0.874$ )**



**Figure 4: Relationship between likability of an item and the effect of a 100-AverageBot push attack on item-item predictions ( $r^2 = 0.853$ )**



**Figure 5: (Lack of) Relationship between entropy of an item and the effect of a 100-AverageBot push attack on item-item recommendations ( $r^2 = 0.004$ )**

on both the user-user and item-item algorithms, but not with the change in ExpTop40 in either algorithm. The popularity of an item correlates highly with the change in ExpTop40 on user-user, but not on item-item. Furthermore, popularity does not correlate well with PredShift. Finally, the entropy of an item’s ratings distribution correlates with neither metric and neither algorithm.

So, while a portion of this hypothesis is rejected, the supported part of the hypothesis does carry an important implication for sys-

tems that use the user-user algorithm for *recommendations*. Items that have low popularity can be an easy push target for attackers. This result is significant because new items initially simultaneously have low popularity, and are highly desirable to push, since new items often sell at a premium.<sup>6</sup>

Note that these results also provide further support for our first hypothesis that different ACF algorithms respond differently to attacks.

## 5. CONCLUSIONS AND FUTURE WORK

Overall, our results are mixed. However, there are some conclusions that can be drawn for operators of recommender systems who would like to reduce the threat of shilling to their systems. We first identify those lessons, then discuss what all of the shilling studies to date together say about the dimensions of shilling, and close by suggesting some rich areas for future work.

**Prefer Item-Item.** The item-item algorithm was much less affected by the attacks in our study than was the user-user algorithm. In many domains, item-item provides recommendations that are the same or better quality as user-user, and the same or better performance. Our study suggests one more reason for operators to prefer item-item: it appears to be more resistant to shilling.

One internal reviewer of our paper suggested that the reason user-user is easier to attack is that the version we used has a neighborhood size of only 20. The argument is that the small neighborhood makes it easy for shills to displace “good” neighbors. To test this hypothesis we repeated some of our experiments with a neighborhood size of 500. We found only very small differences – and the larger neighborhood proved overall easier to shill!

**Use Recommendation Metrics.** Most recommender systems in practice are used as a source of recommendations, rather than predictions [19]. Our results show that metrics that are sensitive to changes in prediction accuracy may be less sensitive to changes in recommendation accuracy. We recommend that operators who are producing mostly recommendations use a recommendations-centric metric, such as Expected Top- $N$  Occupancy, to evaluate the effectiveness of shilling attacks on their systems. We also recommend that researchers consider focusing on recommendation accuracy, rather than prediction accuracy, in future shilling studies.

**Watch Metrics, but Worry Anyway.** Operators wish to know whether their recommendations systems are under attack. Watching for sharp changes in the value of traditional algorithm performance metrics such as MAE may be useful for detecting some attacks. However, our results suggest that many effective attacks will not be visible through simple aggregate metrics like MAE, so work needs to be done to develop more reliable tools to detect attackers. Note that the attacks we used in this paper could be easily detected by watching for individual users with unreasonably large numbers of ratings. Real-world attacks are likely to be more subtle.

**Protect New Items.** Our experiments show that new or obscure items, particularly in the user-user algorithm, are especially susceptible to attack. Recommender system operators should consider obtaining ratings for such items from a trusted source in order to make them less vulnerable. For instance, the ratings for a new item could be seeded with ratings from professional critics, with ratings from trusted volunteers, or with filterbots [8].

The research that has been done on shilling in recommender systems, including the present paper, considers attacks that are in a

<sup>6</sup>In fact, part of the motivation for this study is a long-term MovieLens user who is convinced that new movies are frequently shilled. Inspired by his suspicions, we looked hard for ratings behavior that looks like shilling. We have been unable to find evidence of shilling of new items in MovieLens to date.

fairly narrow region of the space of attacks spanned by our dimensions from Section 2. Here we consider which parts of that space has been covered, and suggest how future studies might cover other areas. We consider each of the dimensions for Section 2 in turn.

Along the **intent** dimension, only basic push and nuke attacks have been examined. One can imagine a world in the not-too-distant future where competing retailers, all operating recommender systems, may deliberately try to sabotage each other's systems in an attempt to frustrate and lure away customers. Attacks that accomplish this goal will likely be quite different from the attacks that have been proposed and tested thus far.

Along the **targets** dimension, the known attacks tend to target broad groups of users (in our case, all users). Our work suggests that this may be difficult under some algorithms such as item-item. We hypothesize that targeted attacks will be more effective, easier to formulate, and harder to detect.

Most attacks described to date assume a limited amount of **knowledge** about the recommender system. They are primarily targeted against **algorithms** like kNN user-user or its kin. They have little **cost/benefit analysis** beyond basic attack size, and have not been analyzed for **detectability**.

Besides formulation of additional classes of attacks, many other open questions remain in this area. Are there highly-robust ACF algorithms that are resistant to most attacks? Why are some attacks better against certain algorithms? Are there reliable ways of detecting attacks? What can system operators do if their system is attacked? Can users of recommender systems do anything to prevent or detect shilling attacks? These questions are becoming increasingly important as recommender systems become more and more commonplace in commercial applications.

## 6. ACKNOWLEDGMENTS

We gratefully thank the members of the GroupLens Research Project for many interesting and inspiring discussions while we carried out this work. We especially acknowledge Brad Miller for the MultiLens system, which we used for many of the experiments. We also appreciate the feedback from Dan Frankowski, Sean McNee, and Al Mamunur Rashid on early drafts of this document. This work was supported by grants from the NSF (DGE 95-54517, IIS 96-13960, IIS 97-34442, IIS 99-78717, and IIS 01-02229)

## 7. REFERENCES

- [1] S. E. Asch. Effects of group pressure upon the modification and distortion of judgements. *Groups, Leadership, and Men*, pages 177–190, 1951.
- [2] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, July 1998.
- [3] J. Canny. Collaborative filtering with privacy via factor analysis. In *IEEE Conference on Security and Privacy*, May 2002.
- [4] D. Cosley, S. K. Lam, I. Albert, J. Konstan, and J. Riedl. Is seeing believing? How recommender system interfaces affect users' opinions. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, 2003. CHI Letters 5(1).
- [5] C. Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *ACM Conference on Electronic Commerce*, pages 150–157, 2000.
- [6] D. Goldberg, D. Nichols, B. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.
- [7] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.
- [8] N. Good, B. Schafer, J. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the 1999 Conference of the American Association of Artificial Intelligence (AAAI-99)*, July 1999.
- [9] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 1999 Conference on Research and Development in Information Retrieval (SIGIR-99)*, Aug. 1999.
- [10] B. J. Jansen and A. Spink. An analysis of web documents retrieved and viewed. In *Internet Computing Conference*, Las Vegas, 2003.
- [11] B. Miller, J. Riedl, and J. Konstan. GroupLens for Usenet: Experiences in applying collaborative filtering to a social information system. In C. Leug and D. Fisher, editors, *From Usenet to CoWebs: Interacting with Social Information Spaces*. Springer-Verlag, 2002.
- [12] B. N. Miller. *Toward a Personal Recommender System*. PhD thesis, University of Minnesota, 2002.
- [13] C. Nass and Y. Moon. Machines and mindlessness: Social responses to computers. *Journal of Social Issues*, pages 81–103, 2000.
- [14] M. P. O'Mahony, N. Hurley, N. Kushmerick, and G. Silvestre. Collaborative recommendation: A robustness analysis. *ACM Transactions on Internet Technology*, 2003. Special Issue on Machine Learning for the Internet.
- [15] M. P. O'Mahony, N. J. Hurley, and G. C. Silvestre. Promoting recommendations: An attack on collaborative filtering. In *Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 494–503. Springer Verlag, 2002.
- [16] P. Resnick, N. Iacovou, M. Sushak, P. Bergstrom, and J. Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of CSCW 1994. ACM SIG Computer Supported Cooperative Work*, 1994.
- [17] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system – a case study. In *ACM WebKDD 2000 Web Mining for E-Commerce Workshop*, 2000.
- [18] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International World Wide Web Conference (WWW10)*, Hong Kong, May 2001.
- [19] J. Schafer, J. Konstan, and J. Riedl. Electronic commerce recommender applications. *Data Mining and Knowledge Discovery*, Jan. 2001.
- [20] U. Shardanand and P. Maes. Social information filtering: Algorithms for automating 'word of mouth'. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 210–217, 1995.
- [21] L. von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *Proceedings of Eurocrypt, 2003*, 2003.