

Detecting Profile Injection Attacks in Collaborative Recommender Systems *

Robin Burke, Bamshad Mobasher, Chad Williams, Runa Bhaumik
Center for Web Intelligence, DePaul University
School of Computer Science, Telecommunication, and Information Systems
Chicago, Illinois, USA
{rburke, mobasher, cwilli43, rbhaumik}@cs.depaul.edu

Abstract

Collaborative recommender systems are known to be highly vulnerable to profile injection attacks, attacks that involve the insertion of biased profiles into the ratings database for the purpose of altering the system's recommendation behavior. In prior work, we and others have identified a number of models for such attacks and shown their effectiveness. This paper describes a classification approach to the problem of detecting and responding to profile injection attacks. This technique significantly reduces the effectiveness of the most powerful attack models previously studied.

1. Introduction

Recent research has begun to examine the vulnerabilities of different recommendation techniques, such as collaborative filtering, in the face of what has been termed “shilling” attacks [4, 2, 7, 9]. We use the more descriptive phrase “profile injection attacks”, since promoting a particular product is only one way such an attack might be used. In a profile injection attack, an attacker interacts with a collaborative recommender system to build within it a number of profiles associated with fictitious identities with the aim of biasing the system's output.

It is easy to see why collaborative filtering is vulnerable to these attacks. A user-based collaborative filtering algorithm collects user profiles, which are assumed to represent the preferences of many different individuals and makes recommendations by finding peers with like profiles. If the profile database contains biased data (many profiles all of which rate a certain

item highly, for example), these biased profiles may be considered peers for genuine users and result in biased recommendations. This is precisely the effect found in [7] and [9].

Our prior work [2, 3] identified a number of attack models, based on different assumptions about attacker knowledge and intent. The overall conclusion is that an attacker wishing to “push” a particular product (make it more likely to be recommended) or to “nuke” it (make it less likely to be recommended) can do so with a relatively modest number of injected profiles, with a minimum of system-specific knowledge and with only the kind of general knowledge about likely user ratings distribution that one might find by reading the newspaper. We also know that profile injection attacks are not merely of theoretical interest, but have been uncovered at e-commerce sites.

Defense against profile injection can take many forms. Some collaborative algorithms are more robust than others against such attacks. Part of our on-going work is to explore the robustness of the most popular algorithms for collaborative recommendation. One recent finding is that a hybrid recommendation approach incorporating both collaborative and knowledge-based components offers significantly improved robustness as opposed to the collaborative system alone. [8]

However robust an algorithm may be, it is impossible to have complete security against profile injection attacks. A collaborative system is designed to adjust its behavior in response to user inputs, and in theory, an attacker could swamp the system with so many profiles as to control it completely. One common defense is to simply make assembling a profile more difficult. A system may require that users create an account and perhaps respond to a captcha¹ before doing so. This increases the cost of creating bogus accounts (although

*This research is supported, in part, by the National Science Foundation Cyber Trust Grant IIS-0430303.

¹www.captcha.net/

with offshore data entry outsourcing available at low rates, the cost may still not be too high for some attackers.) Such measures come at a high cost for the system owner as well, however – they drive users away from participating in collaborative systems, systems which rely on user input to function. In addition, such measures are totally ineffective for recommender systems based on implicit measures such as usage data mined from web logs.

Our aim is to build systems that can detect and respond to the most effective known attack models. Attackers wishing to evade detection will need to adopt less effective attacks, which by definition require greater numbers of profiles to produce the desired change in recommendation behavior. Larger attacks, however, are also conspicuous and in this way, we hope to render profile injection attacks relatively harmless.

This paper describes an approach to the detection of profile injection attacks based on supervised classification. The system is given training data containing attacks generated by our attack models, and learns a classifier to distinguish attack data from genuine user profiles. The primary contribution of the paper is set of classification attributes, derived automatically based on the expected characteristics of attack profiles, that allow for learning highly effective classification models. These attributes include both generic attributes that capture expected distribution of user data within profiles, as well as attributes based in the characteristics of well-known attack models.

2. Attack Models

An attack model is an approach to constructing an attack profile, based on knowledge about the recommender system, its rating database, its products, and/or its users. The general form of a push or nuke attack profile is depicted in Figure 1. An attack profile consists of an m -dimensional vector of ratings, where m is the total number of items in the system. The rating given to the pushed item, the target, is r_{target} . Generally, in a push attack, $r_{target} = r_{max}$, while for a nuke attack, $r_{target} = r_{min}$, where r_{max} and r_{min} are the maximum and minimum allowable rating values, respectively. The ratings r_1 through r_{m-1} are assigned to the corresponding items according to the specific attack model. For some attack models, ratings for selected groups of items may be pre-specified, and other item ratings may be missing or determined according to a random variable. Indeed, the specific strategy used to assign ratings to items 1 through $m - 1$ is what determines the type of attack model used.

Two basic attack models, introduced originally

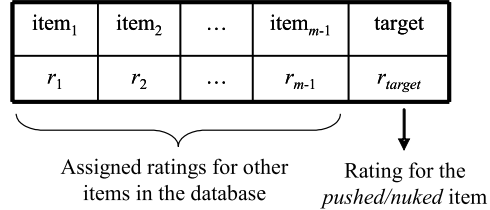


Figure 1. The general form of a push/nuke attack profile.

in [7], and further generalized in [2, 3], are the *Random* and *Average* attacks. Both of these models involve the generation of attack profiles using randomly assigned ratings given to some filler items in the profile. In the Random attack the assigned ratings are based on the overall distribution of user ratings in the database, while in the Average attack the rating for each filler item is computed based on its average rating for all users. Of these, the Average attack is by far the more effective, but it may be impractical to mount, given the degree of system-specific knowledge of the ratings distribution that it requires. Further, as we show in [3], it is not particularly effective against item-based collaborative algorithms. For these reasons, we have developed a number of additional attack models, described in greater detail in [4, 2, 3]. In this paper, however, we will concentrate on the Average and Random attacks.

3. Detecting Profile Injection Attacks

One of the main strengths of collaborative recommender systems is the ability for users with unusual tastes to get meaningful suggestions by the system identifying users with similar peculiarities. This strength is also one of the challenges in securing recommender systems. Specifically the variability of opinion make it difficult to say with certainty whether a particular profile is an attack profile or the preferences of an eccentric user. It is unrealistic to expect all profiles to be classified correctly. The goals for detection and response will therefore be:

- Minimize the impact of an attack, and
- Reduce the likelihood of a successful attack

The attacks that we outlined above work well against collaborative algorithms because they were created by reverse engineering the algorithms to devise inputs with maximum impact. Attacks that deviate from these patterns will be less effective than those

that conform to them. Our approach to attack detection will therefore focus on recognizing attacks that our known attack models. An ideal outcome would be one in which a system could be rendered secure by making attacks against it no longer cost effective, where cost is measured in the attacker’s knowledge, effort, and time.

Our approach to attack detection is one that builds on multiple sources of evidence:

Time series: The entry of ratings that comprise a given user profile represent a time series. Obviously, a profile that is built up at inhuman speed (for example, 100 ratings in one minute) is almost certain to be an attack. Similarly, the set of ratings associated with a given item also can be examined as a time series. An item that shows a sudden burst of identical high or low ratings may be one that is being pushed or nuked.

Vulnerability: Some items in the database are more vulnerable to attack than others. Items with very few ratings will be more volatile in the face of attack profiles, which may overwhelm the genuine ratings. Our experiments have shown that while relatively popular items are still vulnerable to attack, items that are sparsely rated are most vulnerable.

Profile characteristics: Profiles designed as part of an attack will look very different in aggregate than genuine user profiles, simply because they are designed to be efficient – to achieve maximum change in recommendation behavior with the minimum number of profiles injected.

Critical mass: A single profile that happens to match the pattern of, for example, an Average attack may just happen to be a statistical fluke. A thousand such profiles, all of which target the same item, are probably not. Profile injection attacks need a critical mass of attack profiles in order to be successful. Such profiles will not only have suspicious characteristics but they will share a focus on a single attacked item.

In other words, we will expect an attack to consist of a set of profiles having ratings that accrue in an atypical pattern, incorporating ratings for vulnerable items, having statistical properties matching those of a known attack model and focusing on a particular item. The more of these properties that are true of a set of profiles the more likely it is that they constitute an attack.

3.1. Profile Classification

Since we have some knowledge of what types of attacks are successful, we can treat attack identification as a traditional pattern classification problem in which we seek to classify profiles as matching known attack models. It may be that some genuine users will be classified as attackers, with consequences that we explore later.

In this paper, we concentrate on identifying suspicious profiles by their aggregate properties. This is a classification approach which extends some of the features introduced originally in [5]. Our approach also differs since rather than constructing an ad-hoc classifier, we use training data based on our attack models to build a classifier to separate attack profiles from genuine users. The classification attributes are created using two different types of analysis. The first type are created looking at the profile as a whole and are thus generic and not specific to any attack model. The second type is attack-model based, and generates attributes related to detecting characteristics of a specific attack model. We investigate using two common and well-understood classifier learning methods: decision-tree learning using C4.5 [10] and simple nearest-neighbor classification using kNN. We plan to investigate other learning methods, such as clustering.

3.2. Attack Response and System Robustness

Once attack profile(s) have been detected, the question then becomes how the system should respond in order to eliminate or reduce the bias introduced by the attack. Ideally all attack profiles would be ignored and the system would function as if no bias had been injected. However a more likely scenario is there are a number of profiles that are suspected of being part of an attack without 100% certainty. If such a suspicion could be quantified reliably, the probability that a profile was part of an attack could be used as a weight to discount the contribution of such questionable profiles towards any recommendation the system makes. In our experiments here, we use the simpler method of ignoring profiles labeled as attacks when making predictions.

Although we have focused primarily on the direct affect of the push and nuke attacks on the target items, it is worth mentioning that bias in the overall system is also an important aspect of robustness. For a system to be considered robust, it should not only be able to withstand a direct attack on an item with minimal prediction shift; it should also be able to provide just as accurate predictions for all other items.

4. Classification Attributes

We use two types of derived features, generic and model-derived. Generic features are basic descriptive statistics for each profile. The model-derived features are those that are implemented to detect characteristics of specific attack models. These attributes are generated based on the ratings profiles and the detection attributes alone are used by the classifiers.

4.1. Generic Attributes

The attack profiles that we have found experimentally to be most effective at biasing system behavior do not look very much like real user profiles. In particular, they will always have a single rating r_{target} aimed at biasing the system in some extreme direction. We can capture the effect of this extreme value (and also attend particularly to the vulnerable items with fewer ratings) through a measure called *Rating Deviation from Mean Agreement* (RDMA) developed by Chirita et al. [5]

The RDMA attribute combines the difference in rating from the item’s average rating and the inverse rating frequency. The attribute is calculated as follows:

$$RDMA_j = \frac{\sum_{i=0}^{N_j} \frac{|r_{i,j} - Avg_i|}{NR_i}}{N_j}$$

where N_j is the number of items user j rated, $r_{i,j}$ is the rating given by user j to item i , NR_i is the overall number of ratings in the system given to item i .

For an attack to have a significant impact for any item with any significant number of previous ratings, it will likely need to consist of more than just a single attack profile. In our experiments, the most effective attacks are those in which a large number of profiles with very similar characteristics are introduced. If we look at the differences between a given profile and its nearest neighbors, an attack profile will differ little because it is similar to other attack profiles, all generated based on the same model. Genuine profiles will be more dispersed. This intuition is captured in the *Degree of Similarity with Top Neighbors* (DegSim) feature, also introduced in [5].

The DegSim attribute is based on the average Pearson correlation of the profile’s k nearest neighbors and is calculated as follows:

$$DegSim_j = \frac{\sum_{i=1}^k W_{ij}}{k}$$

where W_{ij} is the Pearson correlation between users i and j , and k is the number of neighbors.

A third generic attribute that we have introduced is based on the number of total ratings in a given profile. Some attacks require profiles that rate many if not all of the items in the system. If there is a large number of possible items, it is unlikely that such profiles could come from a real user, who would have to enter them all manually, as opposed to a soft-bot implementing a profile injection attack. This feature, called *Length Variance*, is computed as follows:

$$LengthVar_j = \frac{|\#ratings_j - \overline{\#ratings}|}{\sum_{i=0}^N (\#ratings_i - \overline{\#ratings})^2}$$

where $\#ratings_j$ is the total number of ratings in the system for user j , and N is the total number of users in the system.

4.2. Model-Derived Attributes

Model-derived attributes are those that aim to recognize the distinctive signature of a particular attack model. The Average and Random attacks each incorporate a form of partitioning. One item is chosen as the target, additional items are chosen as filler, and other items are ignored. Our detection model attempts to identify a partitioning of each profile that maximizes its similarity to the attack model. One useful property of a partition-based features is that their derivation can be sensitive to additional information (such as time-series or critical mass data) that suggests likely attack targets.

4.2.1 Average Attack Detection Model

The Average attack model divides the profile into three partitions: the target item given an extreme rating, the filler items given other ratings (determined based on the attack model), and unrated items. The model essentially just needs to select an item to be the target and all other rated items become fillers. By the definition of the Average attack, the filler ratings will be populated such that they closely match the rating average for each filler item. We would expect that a profile generated by an average attack would exhibit a high degree of similarity (low variance) between its ratings and the average ratings for each item except for the single item chosen as the target.

The formalization of this intuition is to iterate through all the rated items, selecting each in turn as the possible target, and then computing the mean variance between the non-target (filler) items and the overall average. Where this metric is minimum, the target item is the one most compatible with the hypothesis of the

profile as being generated by an average attack, and the magnitude of the variance is an indicator of how confident we might be with this hypothesis. More formally, then, we compute $MeanVar$ for each possible r_{target} in the profile P_j of user j .

$$MeanVar(r_{target}, j) = \frac{\sum_{i \in (P_j - r_{target})} (r_{i,j} - \bar{r}_i)^2}{|K|}$$

where P_j is the profile of user j , r_{target} is hypothesized target item, $r_{i,j}$ is the rating user j has given item i , and \bar{r}_i is the mean rating of item i across all users.

We then select the target t with rating r_{target} such that $MeanVar(r_{target}, j)$ is minimized. From this optimal partitioning of P_j , we use $MeanVar(t, j)$ as the *Filler Mean Variance* feature for classification purposes. We also compute *Filler Mean Difference*, which is the average of the absolute value of the difference between the user’s rating and the mean rating (rather than the squared value as in the variance.)

Finally, in an average attack, we would expect that attack profiles would have very similar within-profile variance: they would have more or less similar ratings for the filler items and an extreme value for the target item. So, our third model-derived feature is *Profile Variance*, simply the variance associated with the profile itself,

4.2.2 Random Attack Detection Model

The random attack is also a partitioning attack, but in this case, the ratings for the filler items are chosen randomly such that their mean is the overall system average rating across all items.

As in the average attack, we can test various partitions by selecting possible targets and computing a metric for each possible choice. Since we are here aiming to detect a random attack, we use the correlation between the profile and the average rating for each item. Since the ratings are generated randomly, we would expect low correlation between the filler items and the individual ratings. (Note that this is the opposite of what we would expect for an average attack that would be very highly correlated with the item average.) We again select the most likely target t , for which the correlation between the filler and the item averages is minimized. We call this minimum the *Filler Average Correlation*.

Another feature that would be expected of random attack profiles is that the filler item will have a mean rating very close to the overall system average. This expectation is captured in a another feature, the *Filler Mean Difference*, the difference between the mean rating across filler items and the overall system mean.

5. Experiments

In our experiments we have used the publicly-available Movie-Lens 100K dataset². This dataset consists of 100,000 ratings on 1682 movies by 943 users. All ratings are integer values between one and five where one is the lowest (disliked) and five is the highest (most liked). Each user in the dataset has rated at least 20 movies.

5.1. Recommendation Algorithm

We used the standard user-based collaborative recommendation algorithm using k -nearest-neighbor prediction [6, 11]. The algorithm assumes there is a single user / item pair for which a prediction is sought. In our experiments this is generally the pushed item, since we are primarily interested in the impact that attacks have on this item. The k NN-based algorithm operates by selecting the k most similar users to the target user, and formulates a prediction by combining the preferences of these users. Similarity is measured using Pearson’s r -correlation coefficient: similar users are those whose profiles are highly correlated with each other. In our implementation, we use a value of 20 for the neighborhood size, and we filter out all neighbors with a similarity of less than 0.1.

Once the most similar users are identified, we use the following formula to compute the prediction for an item i for target user u .

$$p_{u,i} = \bar{r}_a + \frac{\sum_{v \in V} sim_{u,v}(r_{v,i} - \bar{r}_v)}{\sum_{v \in V} |sim_{u,v}|}$$

where V is the set of k similar users and $r_{v,i}$ is the rating of those users who have rated item i , \bar{r}_v is the average rating for the target user over all rated items, and $sim_{u,v}$ is the mean-adjusted Pearson correlation described above.

5.2. Experimental Setup

The attack detection and response experiments were conducted using a separate training and test set by partitioning the ratings data in half. The first half was used to create training data for the attack detection classifiers used in later experiments. For each test the 2nd half of the data was injected with attack profiles and then run through the classifier that had been trained on the augmented first half of the data. This

²<http://www.cs.umn.edu/research/GroupLens/data/>

approach was used since a typical cross-validation approach would be overly biased as the same movie being attacked would also be the movie being trained for. Thus requiring the assumption that the system had a priori knowledge of which item(s) would be attacked.

The training set for each classifier consisted of a set of authentic profiles together with attack profiles generated using the Average and Random attack models. Several training sets were created by inserting a mix of Average and Random attacks at different filler sizes (items rated in addition to the target item) and attack sizes against 1 or 2 movies. The attacked movies in the training sets were chosen at random from movies that had between 80 and 100 ratings; about 1/4 of the movies in the database have more ratings. This range was selected so that there are enough ratings to balance the somewhat large training attack, while still making the training sensitive to smaller attacks on less frequently rated items. Because the extracted features are aggregated over a single profile and not across profiles, the selection of these items in the training data makes very little difference in the trained classifier.

The detection attributes were then automatically generated based on the augmented dataset and a class attribute (authentic/attack) was added. The detection attributes generated consisted of the generic attributes mentioned above using $k = 3$ for DegSim in addition to the Average and Random attack model-specific attributes for both push ($r_{target} = 5$) and nuke ($r_{target} = 1$) attacks. Thus a total of 14 attributes were used for training including the binary attack class attribute.

To compare the performance of different classifiers, we selected two algorithms: C4.5 [10] and kNN [1]. All classification tests were performed using Weka’s implementations of C4.5 and kNN classifier [12]. For the kNN classifier a neighborhood size of 9 was used ($k = 9$) and 1/distance weighting was used. In the classification accuracy experiments, 50 movies were selected taking care that the distribution of ratings for these movies matched the overall ratings distribution. These movies were then attacked individually and results reported are based on the average across those 50 movies, none of which were attacked in the training set.

5.3. Results

When analyzing the classifier accuracy, both type I (false positive) and type II (false negative) errors are important. Type I errors mean that real users are labeled as attackers; type II errors result in attackers slipping past our detection algorithm. However, as we show below, false positives are not particularly harm-

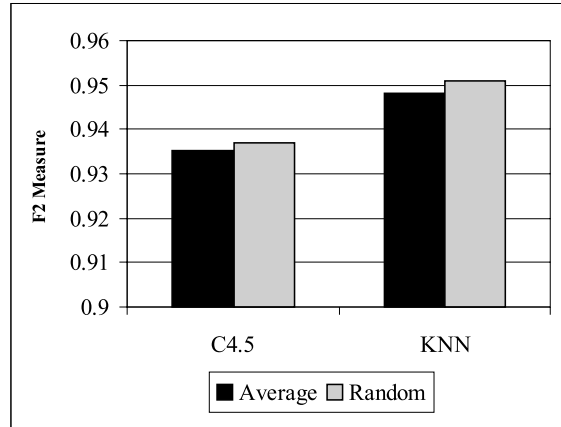


Figure 2. F-measure for C4.5 and kNN classifiers using a 1% attack.

ful if the system has a sufficiently large user base. This means that recall (finding all of the attackers) should be valued more than precision (detecting only real attackers.) We therefore calculate the F2 measure, which is a weighted combination of precision and recall, in which recall has twice the weight of precision.

Figure 2 shows these classification results. Both classifiers perform extremely well with F-measures above 0.9. The kNN classifier is slightly more effective with an F-measure of greater than 0.94 for both types of attacks.

An alternate way to visualize the trade off of coverage and accuracy can be seen in the Receiver Operator Characteristics (ROC) curves. The results for the Average and Random attack models, comparing the performance of the two classifiers, are depicted in Figures 3 and 4, respectively. For the Average attack, kNN has the greatest area under the curve at 0.94 followed by C45 (0.86). For the Random attack the difference in areas are more pronounced with kNN (0.94) and C45(0.59).

Two questions follow from these results:

Accuracy: As the figure shows, both detection algorithms incorrectly classify a portion of the authentic users as attack users. Does the system still make good predictions even when some genuine users are labeled as attackers and therefore ignored?

Robustness: Both algorithms also allow some attackers to slip past undetected, but the vast majority of attack profiles are correctly identified. To what extent does this detection ability succeed in de-

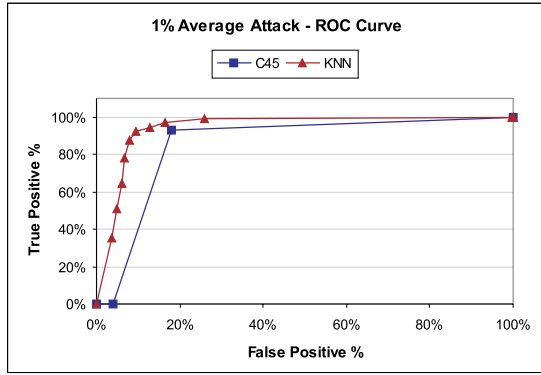


Figure 3. Comparison of 1% attack classification ROC curves for Average Attack.

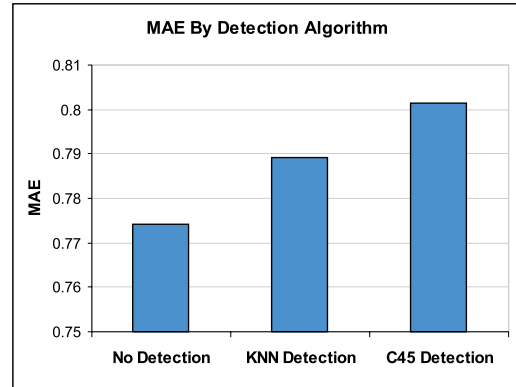


Figure 5. Mean Absolute Error by detection algorithm.

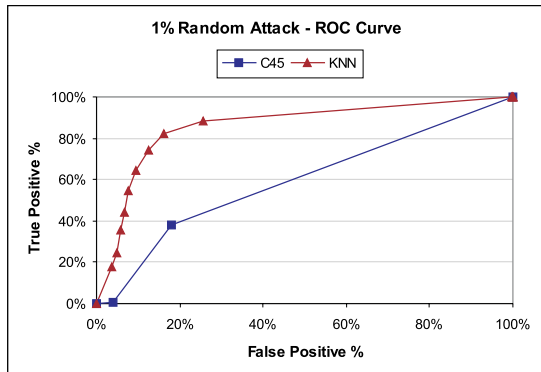


Figure 4. Comparison of 1% attack classification ROC curves for Random Attack.

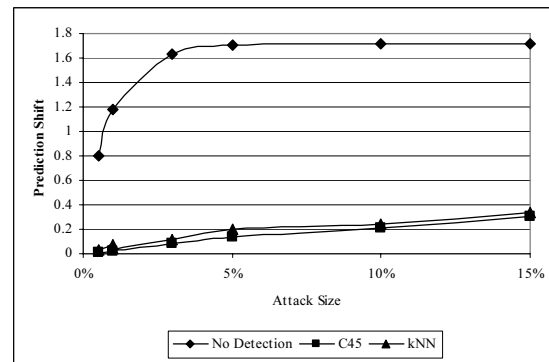


Figure 6. Comparison of Prediction Shift as a function of attack size.

fending the system against the influence of an attack?

To answer these questions, we experimented with a version of the user-based recommendation algorithm in which users identified as attackers were ignored in the generation of recommendations.

We can get at the question of accuracy by looking at the Mean Absolute Error (MAE) for the system's predictions. To compute this value, we compare predicted and actual ratings over all users and movies in the original test set, applying the classifier such that all authentic users labeled as attackers are not included in predictions. If the detection system discarded too many real profiles (false positives), we would expect that prediction accuracy would go down and the error would go up. Figure 5 shows that, although both C4.5 and kNN detection incorrectly classified some authentic users, the algorithms still are quite accurate with

less than 0.02 on a rating scale of 1-5 or less than 1% difference from the system without detection.

It is also interesting to note that although the kNN algorithm misclassified fewer authentic users than C4.5, the authentic users it did misclassify had a larger impact on the prediction accuracy, although this difference is not statistically significant.

The question of robustness can be addressed several ways, but one metric that we have found useful is *Prediction Shift*, the extent to which the system's predicted rating for the target item changes as a result of the attack. Figure 6 shows the resulting prediction shift caused by an average attack with a filler size of 3%. Either detection algorithm is quite successful in reducing the impact caused by even large average attacks. For the prediction shift experiments, attack classification was incorporated by eliminating any user from similarity consideration if it was classified as an attack user. User-based kNN collaborative recommen-

dition was then applied with a neighborhood size of $k = 20$.

6. Conclusion

Profile injection attacks have been shown to be effective threats to the robustness of collaborative recommender systems. Our work and others have pointed out the vulnerabilities shared by the most commonly implemented collaborative algorithms. In this paper, we demonstrate a classifier learning approach that can quite accurately distinguish real from attack data and limit the damage attacks can cause.

Several outstanding questions remain, however. We have incorporated attack-specific feature extraction into the classifiers. It remains to be seen how well classifiers trained in this way can identify attack profiles generated by some of the other attack models that have arisen in our research. Another area we plan to explore are the methods an attacker may employ to make their attacks harder to detect and the robustness of a classification approach to attacks that deviate from known attack models.

Some preliminary results, not included here for reasons of space, indicate that the classifiers trained on average and random attacks do work well on some but not all of the other attack models that we have identified. Further research is necessary to determine if different training data or additional feature extraction is necessary to cover these attacks. We also need to study the robustness of our detection method for nuke attacks.

Our model of detection described above incorporates multiple dimensions, such as time series and critical mass information. The results reported here, however, do not incorporate temporal properties and use the profiles in isolation without attempting to identify common items under attack. We expect that taking these features into account will provide further enhancements to our detection accuracy.

References

- [1] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, January 1991.
- [2] R. Burke, B. Mobasher, and R. Bhaumik. Limited knowledge shilling attacks in collaborative filtering systems. In *Proceedings of the 3rd IJCAI Workshop in Intelligent Techniques for Personalization*, Edinburgh, Scotland, August 2005.
- [3] R. Burke, B. Mobasher, C. Williams, and R. Bhaumik. Segment-based injection attacks against collaborative filtering recommender systems. In *Proceedings of the International Conference on Data Mining (ICDM 2005)*, Houston, December 2005.
- [4] R. Burke, B. Mobasher, R. Zabicki, and R. Bhaumik. Identifying attack models for secure recommendation. In *Beyond Personalization: A Workshop on the Next Generation of Recommender Systems*, San Diego, California, January 2005.
- [5] P.-A. Chirita, W. Nejdl, and C. Zamfir. Preventing shilling attacks in online recommender systems. In *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 67–74, New York, NY, USA, 2005. ACM Press.
- [6] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd ACM Conference on Research and Development in Information Retrieval (SIGIR'99)*, Berkeley, CA, August 1999.
- [7] S. Lam and J. Reidl. Shilling recommender systems for fun and profit. In *Proceedings of the 13th International WWW Conference*, New York, May 2004.
- [8] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams. Effective attack models for shilling item-based collaborative filtering systems. In *Proceedings of the 2005 WebKDD Workshop, held in conjunction with ACM SIGKDD'2005*, Chicago, Illinois, August 2005.
- [9] M. O'Mahony, N. Hurley, N. Kushmerick, and G. Silvestre. Collaborative recommendation: A robustness analysis. *ACM Transactions on Internet Technology*, 4(4):344–377, 2004.
- [10] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [11] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collab