A Survey of Collaborative Recommendation and the Robustness of Model-Based Algorithms*

J.J. Sandvig and Bamshad Mobasher and Robin Burke
DePaul University
School of Computer Science, Telecommunications
and Information Systems
{jsandvig,mobasher,rburke}@cs.depaul.edu

Abstract

The open nature of collaborative recommender systems allows attackers who inject biased profile data to have a significant impact on the recommendations produced. Standard memory-based collaborative filtering algorithms, such as k-nearest neighbor, are quite vulnerable to profile injection attacks. Previous work has shown that some model-based techniques are more robust than standard k-nn. Model abstraction can inhibit certain aspects of an attack, providing an algorithmic approach to minimizing attack effectiveness. In this paper, we examine the robustness of several recommendation algorithms that use different model-based techniques: user clustering, feature reduction, and association rules. In particular, we consider techniques based on k-means and probabilistic latent semantic analysis (pLSA) that compare the profile of an active user to aggregate user clusters, rather than the original profiles. We then consider a recommendation algorithm that uses principal component analysis (PCA) to calculate the similarity between user profiles based on reduced dimensions. Finally, we consider a recommendation algorithm based on the data mining technique of association rule mining using the Apriori algorithm. Our results show that all techniques offer large improvements in stability and robustness compared to standard k-nearest neighbor. In particular, the Apriori algorithm performs extremely well against low-knowledge attacks, but at a cost of reduced coverage, and the PCA algorithm performs extremely well against focused attacks. Furthermore, our results show that all techniques can achieve comparable recommendation accuracy to standard k-nn.

1 Introduction

A widely accepted approach to user-based collaborative filtering is the k-nearest neighbor algorithm. However, memory-based algorithms such as k-nn do not scale well to commercial recommender systems. Model-based algorithms are widely accepted as a way to alleviate the scaling problem presented by memory-based algorithms in data-intensive commercial recommender systems. Building a model of the dataset allows off-line processing for the most rigorous similarity calculations. In some cases, this is at the cost of lower recommendation accuracy [1].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work was supported in part by the National Science Foundation Cyber Trust program under Grant IIS-0430303.

A positive side effect of a model-based approach is that it may provide improved robustness against attacks. An adaptive system dependent on anonymous, unauthenticated user profiles is subject to manipulation. The standard collaborative filtering algorithm builds a recommendation for a target user by combining the stored preferences of peers with similar interests. If a malicious user injects the profile database with a number of fictitious identities, they may be considered peers to a genuine user and bias the recommendation. We call such attacks *profile injection attacks* (also known as *shilling* [2]).

Recent research has shown that surprisingly modest attacks are sufficient to manipulate the most common CF algorithms [3, 2, 4, 5]. Profile injection attacks degrade the objectivity and accuracy of a recommender system over time, causing frustration for its users and potentially leading to high user defection. However, a model-based approach is an abstraction of detailed user profiles. We hypothesize that this abstraction minimizes the influence of an attack, because attack profiles are not directly used in recommendation.

In our study, we have focused on the robustness of user clustering, feature reduction, and association rules. We first consider techniques based on k-means clustering and probabilistic latent semantic analysis (pLSA) that compare the profile of an active user to aggregate user clusters, rather than the original profiles. Probabilistic latent semantic analysis is used infer hidden relationships among groups of users, which are then used to form "fuzzy" clusters. Each user has a degree of association with every cluster, allowing particularly authoritative users to exercise greater influence on recommendation.

We then consider a recommendation algorithm that uses principal component analysis (PCA) to calculate the similarity between user profiles based on reduced dimensions. Principal component analysis tries to extract a set of uncorrelated factors from a given set of multicolinear variables. By keeping only those principal components that explain the greatest amount of variance in the data, we effectively reduce the number of features that must be used for a similarity calculation.

Finally, we consider a recommendation algorithm based on the data mining technique of association rule mining using the Apriori algorithm. Association rule mining is a technique common in data mining that attempts to discover patterns of products that are purchased together. These relationships can be used for myriad purposes, including marketing, inventory management, etc. We have adapted the Apriori algorithm [6] to collaborative filtering in an attempt to discover patterns of items that have common ratings.

The primary contribution of this paper is to demonstrate that model-based algorithms provide an algorithmic approach to robust recommendation. Our results show that all techniques offer large improvements in stability and robustness compared to standard k-nearest neighbor. In particular, the Apriori and PCA algorithms performs extremely well against low-knowledge attacks, but in the case of Apriori at a cost of reduced coverage, and the k-means and pLSA algorithms perform extremely well against focused attacks. Furthermore, our results show that all techniques can achieve comparable recommendation accuracy to standard k-nn.

2 Recommendation Algorithms

In general, user-based collaborative filtering algorithms attempt to discover a neighborhood of user profiles that are similar to a target user. A rating value is then predicted for all missing items in the target user's profile, based on ratings given to the item within the neighborhood. We begin with background information on the standard memory-based k-nn. We then present several recommendation algorithms based on model-based techniques of user clustering (k-means and pLSA), feature reduction (PCA), and association rules (Apriori).

2.1 k-Nearest Neighbor

The standard k-nearest neighbor algorithm is widely used and reasonably accurate [7]. Similarity between the target user, u, and a neighbor, v, is computed using Pearson's correlation coefficient:

$$sim_{u,v} = \frac{\sum_{i \in I} (r_{u,i} - \bar{r}_u) * (r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_u)^2} * \sqrt{\sum_{i \in I} (r_{v,i} - \bar{r}_v)^2}} \tag{1}$$

where $r_{u,i}$ and $r_{v,i}$ are the ratings of some item i for u and v, respectively; and $\bar{r}_u$ and $\bar{r}_v$ are the average of the ratings of u and v over I, respectively.

After similarities are calculated, the k most similar users that have rated the target item are selected as the neighborhood. This implies a target user may have a different neighborhood for each target item. It is also common to filter neighbors with similarity below a specified threshold. This prevents predictions being based on very distant or negative correlations. After identifying a neighborhood, we compute the prediction for a target item i and target user u as follows:

$$pred_{u,i} = \bar{r}_u + \frac{\sum_{v \in V} sim_{u,v}(r_{v,i} - \bar{r}_v)}{\sum_{v \in V} |sim_{u,v}|} \tag{2}$$

where V is the set of k similar neighbors that have rated i; $r_{v,i}$ is the rating of i for neighbor v; $\bar{r}_u$ and $\bar{r}_v$ are the average ratings over all rated items for u and v, respectively; and $sim_{u,v}$ is the Pearson correlation between u and v. The formula computes the degree of preference for all neighbors, weighted by their similarity, and then adds this to the target user's average rating.

2.2 k-Means Clustering

A standard model-based collaborative filtering algorithm uses k-means to cluster similar users. Given a set of user profiles, the space can be partitioned into k groups of users that are close to each other based on a measure of similarity. The discovered user clusters are then applied to the user-based neighborhood formation task, rather than individual profiles.

To make a recommendation for a target user u and target item i, we select a neighborhood of user clusters that have a rating for i and whose aggregate profile v_k is most similar to u. This neighborhood represents the set of user segments that the target user is most likely to be a member, based on a measure of similarity. For this task, we use Pearson's correlation coefficient. We can now make a prediction for item i as described in the previous section, where the neighborhood V is the set of user cluster aggregate profiles most similar to the target user.

2.3 Probabilistic Latent Semantic Analysis

Probabilistic latent semantic analysis (pLSA) models [8] provide a probabilistic approach for characterizing latent or hidden semantic associations among co-occurring objects. We have applied pLSA to the creation of user clusters in the context of collaborative filtering [9].

Given a set of n users, $U = \{u_1, u_2, \cdots, u_n\}$, and a set of m items, $I = \{i_1, i_2, \cdots, i_m\}$ the pLSA model associates an unobserved factor variable $Z = \{z_1, z_2, \cdots, z_l\}$ with observations in the rating data. For a target user u and a target item i, the following joint probability can be defined:

$$P(u, i) = \sum_{k=1}^{l} Pr(z_k) \cdot Pr(u|z_k) \cdot Pr(i|z_k) \tag{3}$$

In order to explain a set of ratings (U, I), we need to estimate the parameters $Pr(z_k)$, $Pr(u|z_k)$, and $Pr(i|z_k)$, while maximizing the likelihood of the rating data $L(U, I) = \sum_{u \in U} \sum_{i \in I} r_{u,i} \cdot \log Pr(u, i)$ where $r_{u,i}$ is the rating of user u for item i. The Expectation-Maximization (EM) algorithm is used to perform maximum likelihood parameter estimation, based on initial values of $Pr(z_k)$, $Pr(u|z_k)$, and $Pr(i|z_k)$. Iterating the expectation and maximization steps monotonically increases the total likelihood of the observed data $L(U, I)$, until a local optimum is reached.

We now identify clusters of users that have similar underlying interests. For each latent variable z_k, we create a user cluster C_k and select all users having probability $Pr(u|z_k)$ exceeding a certain threshold μ. If a user does not exceed the threshold for any latent variable, it is associated with the user cluster of highest probability. Thus, every user profile will be associated with at least one user cluster, but may be associated with multiple clusters. This allows authoritative users to have broader influence over predictions, without adversely affecting coverage in sparse rating data. A recommendation is made for a target user u and target item i in a similar manner to k-means clustering.

2.4 Principal Component Analysis

Principal component analysis is a dimensionality reduction technique that tries to extract a set of uncorrelated factors from a given set of multicolinear variables. Each factor represents a latent pattern that is explained by the degree of correlation to the explicit variables. Factor analysis in general assumes there is an underlying structure to the explicit variables. In a recommendation context, the factors may represent fine-grained groupings of items. For example, movies may have implicit groupings such as style and genre.

PCA identifies the orthogonal axes of variance within a dataset, where the first axis represents the largest variance in the data, the second axis represents the second largest variance, and so on. It is based on a theorem of linear algebra stating that for any real symmetric matrix A, there exists a unitary matrix Λ such that $\Sigma = \Lambda^T A \Lambda$ and Σ is diagonal.

A solution is found using the eigenvectors of A, where the columns of Λ are the eigenvectors ordered in decreasing eigenvalues. Then, $\Sigma_{ii} = \lambda_i$ is the i^{th} largest eigenvalue of A. The principal components are calculated using the covariance matrix of the user data U with respect to items, such that $A = \frac{1}{n-1} U^T U$. Prior to calculating the covariance matrix, it is important to adjust the matrix U such that each item vector is zero-mean. For each u_i in U, modify the user vector such that $u'_i = u_i - m$ where $m = \frac{1}{n} \sum_{i=0}^{n} u_i$ is the vector of item means.

A caveat to this approach is the potential effect of missing data. Collaborative filtering datasets are notoriously sparse, but PCA requires a dense covariance matrix to calculate eigenvectors. We have resolved this issue with an elegant solution. Before adjusting for item means and calculating the covariance matrix, we subtract each user's mean rating from the user vector, where the mean is calculated by ignoring the missing ratings. The idea is that different users may have different "baselines" around which their ratings are distributed. We then set all missing values to 0 under the assumption that a user has no preference for an item that has not been rated.

In order to reduce the number of dimensions in the feature vector Λ, we simply keep the eigenvectors with the largest eigenvalues and discard the rest. There are several ways to choose the number of eigenvectors to keep for PCA, but in our experiments we have found the percentage of variance criteria to yield the most accurate results. We keep the number of eigenvectors such that the total cumulative percentage of variance surpasses some threshold, μ.

To calculate a prediction for a target item i and target user u, we modify the standard k-nn algorithm, such that Equation 1 is calculated with respect to the reduced dimension vectors of user u and neighbor v. Each reduced dimension vector is calculated as $u' = \Lambda'^T (u - m)$, where Λ' is the reduced dimension feature vector; m is the vector of item means; and u is the target user or neighbor vector, mean adjusted according to that user's mean.

2.5 Association Rule Mining

Association rule mining is a common technique for performing market basket analysis. The intent is to capture relationships among items based on patterns of co-occurrence across transactions. We have applied association rules to the context of collaborative filtering [10]. Considering each user profile as a transaction, it is possible to use the Apriori algorithm [6] and generate association rules for groups of commonly liked items.

Given a set of user profiles U and a set of item sets $I = \{I_1, I_2, \ldots, I_k\}$, the *support* of an item set $I_i \in I$ is defined as $\sigma(I_i) = |\{u \in U : I_i \subseteq u\}| \,/\, |U|$. Item sets that satisfy a minimum support threshold are used to generate association rules. These groups of items are referred to as *frequent item sets*. An association rule r is an expression of the form $X \implies Y(\sigma_r, \alpha_r)$, where X and Y are item sets, σ_r is the support of $X \cup Y$, and α_r is the *confidence* for the rule r given by $\sigma(X \cup Y)/\sigma(X)$. In addition, association rules that do not satisfy a minimum *lift* threshold are pruned, where lift is defined as $\alpha_r/\sigma(Y)$.

Before performing association rule mining on a collaborative filtering dataset, it is necessary to discretize the rating values of each user profile. We first subtract each user's average rating from the ratings in their profile to obtain a zero-mean profile. Next, we give a discrete category of "like" or "dislike" to each rated item in the profile if it's rating value is $>$ or $\leq$ zero, respectively. In classic market basket analysis, it is assumed that a customer will not purchase an item they do not like. Hence, a transaction always contains implicit positive ratings. However, when dealing with explicit rating data, certain items may be disliked. A collaborative recommender must take such preference into account or risk recommending an item that is rated often, but disliked by consensus.

To make a recommendation for a target user profile u, we create a set of candidate items C such that an association rule r exists of the form $X \subseteq u \implies i \in C$ where i is an unrated item in the profile u. In practice, it is not necessary to search every possible association rule given u. It is sufficient to find all frequent item sets $X \subseteq u$ and base recommendations on the next larger frequent itemsets $Y \supset X$ where Y contains some item i that is unrated in u. The candidate set C is then sorted according to confidence scores and the top N items are returned as a recommendation.

A caveat to this approach is the possibility of conflicting recommendations in the candidate set C. For example, one association rule may add item i to the candidate set with a "like" label, whereas another rule may add the same item with a "dislike" label. There is no ideal solution, but we have chosen to assume that there are opposing forces for the recommendation of the item. In our implementation, we subtract the confidence value of the "dislike" label from the confidence value of the "like" label.

3 Profile Injection Attacks

A collaborative recommender database consists of many user profiles, each with assigned ratings to a number of products that represent the user's preferences. A malicious user may insert multiple profiles under false identities designed to bias the recommendation of a particular item for some economic advantage. This may be in the form of an increased number of recommendations for the attacker's product, or fewer recommendations for a competitor's product.

3.1 An Example

Consider an example recommender system that identifies interesting books for a user. The representation of a user profile is a set of product / rating pairs. A rating for a particular book can be in the range 1-5, where 5 is the highest possible rating. Alice, having built a profile from previous visits, returns to the system for new recommendations. Figure 1 shows Alice's profile along with that of seven genuine users.

An attacker, Eve, has inserted three profiles (Attack1-3) into the system to mount an attack promoting the target item, Item6. Each attack profile gives high ratings to Eve's book, labeled Item6. If the attack profiles are constructed such that they are similar to Alice's profile, then Alice will be recommended Eve's book. Even

	Item1	Item2	Item3	Item4	Item5	Item6	Correlation with Alice
Alice	5	2	3	3		?	
User1	2		4		4	1	-1.00
User2	3	1	3		1	2	0.76
User3	4	2	3	1		1	0.72
User4	3	3	2	1	3	1	0.21
User5		3		1	2		-1.00
User6	4	3		3	3	2	0.94
User7		5		1	5	1	-1.00
Attack1	**5**		**3**		**2**	**5**	1.00
Attack2	**5**	**1**	**4**		**2**	**5**	0.89
Attack3	**5**	**2**	**2**	**2**		**5**	0.93
Correlation with Item6	0.85	-0.55	0.00	0.48	-0.59		

Figure 1: an example attack on Item6

without direct knowledge of Alice's profile, similar attack profiles may be constructed from average or expected ratings across all users.

Disregarding Eve's attack profiles for a moment, we can compute Alice's predicted preference for Item6. Assuming 1-nearest neighbor, Alice will not be recommended Item6. The most highly correlated user to Alice is User6, who clearly does not like Item6. Therefore, Alice is expected to dislike Item6.

After Eve's attack, however, Alice receives a very different recommendation. As a result of including the attack profiles, Alice is most highly correlated to Attack1. In this case, the system predicts Alice will like Item6 because it is rated highly by Attack1. She is given a recommendation for Item6, although it is not the ideal suggestion. Clearly, this can have a profound effect on the effectiveness of a recommender system. Alice may find the suggestion inappropriate, or worse; she may take the system's advice, buy the book, and then be disappointed by the delivered product.

3.2 Attack Types

A variety of attack types have been studied for their effectiveness against different recommendation algorithms [4, 5]. An *attack type* is an approach to constructing attack profiles, based on knowledge about the recommender system, its rating database, its products, and/or its users. In a push attack, the target item is generally given the maximum allowed rating. The set of *filler items* represents a group of selected items in the database that are assigned ratings within the attack profile. Attack types can be characterized according to the manner in which they choose filler items, and the way that specific ratings are assigned. In this paper, we focus on three attack types that have been shown to be very effective against standard user-based collaborative filtering recommenders.

The random attack is a basic attack type that assigns random ratings to filler items, distributed around the global rating mean [2, 4]. The attack is very simple to implement, but has limited effectiveness.

The average attack attempts to mimic general user preferences in the system by drawing its ratings from the rating distribution associated with each filler item [2, 4]. An average attack is much more effective than a random attack; however, it requires greater knowledge about the system's rating distribution. In practice, the additional knowledge cost is minimal. An average attack can be quite successful with a small filler item set, whereas a random attack usually must have a rating for every item in the database in order to be effective.

An attacker may be interested primarily in a particular set of users – likely buyers of a product. A segment attack attempts to target a specific group of users who may already be predisposed toward the target item [4]. For example, an attacker that wishes to push a fantasy book might want the product recommended to users expressing interest in *Harry Potter* and *Lord of the Rings*. A typical segment attack profile consists of a number of selected items that are likely to be favored by the targeted user segment, in addition to the random filler items. Selected items are expected to be highly rated within the targeted user segment and are assigned the maximum

rating value along with the target item.

4 Experimental Evaluation

To evaluate the robustness of model-based techniques, we compare the results of push attacks using different parameters. In each case, we report the relative improvement over the k-nearest neighbor approaches.

4.1 Dataset

In our experiments, we have used the publicly-available Movie-Lens 100K dataset[1]. 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 (liked). Our data includes all users who have rated at least 20 movies.

To conduct attack experiments, the full dataset is split into training and test sets. Generally, the test set contains a sample of 50 user profiles that mirror the overall distribution of users in terms of number of movies seen and ratings provided. The remaining user profiles are designated as the training set. All attack profiles are built from the training set, in isolation from the test set.

The set of attacked items consists of 50 movies whose ratings distribution matches the overall ratings distribution of all movies. Each movie is attacked as a separate test, and the results are aggregated. In each case, a number of attack profiles are generated and inserted into the training set, and any existing rating for the attacked movie in the test set is temporarily removed.

For every profile injection attack, we track *attack size* and *filler size*. Attack size is the number of injected attack profiles, and is measured as a percentage of the pre-attack training set. There are approximately 1000 users in the database, so an attack size of 1% corresponds to about 10 attack profiles added to the system. Filler size is the number of filler ratings given to a specific attack profile, and is measured as a percentage of the total number of movies. There are approximately 1700 movies in the database, so a filler size of 10% corresponds to about 170 filler ratings in each attack profile. The results reported below represent averages over all combinations of test users and attacked movies.

4.2 Evaluation Metrics

There has been considerable research on the accuracy and performance of recommender systems [11]. We use the mean absolute error (MAE) accuracy metric, a statistical measure for comparing predicted values to actual user ratings.We define coverage as the percentage of items in the database for which an algorithm is able to make a prediction.

However, our overall goal is to measure the effectiveness of an attack; the "win" for the attacker. In the experiments reported below, we measure hit ratio - the average likelihood that a top n recommender will recommend a pushed item, compared to all other items.

Hit ratio measures the effectiveness of an attack on a pushed item compared to other items. Let R_u be the set of top n recommendations for user u. For each push attack on item i, the value of a recommendation hit for user u denoted by H_{ui}, can be evaluated as 1 if $i \in R_u$; otherwise H_{ui} is evaluated to 0. We define hit ratio as the number of hits across all users in the test set divided by the number of users in the test set. The hit ratio for a pushed item i over all users in a set can be computed as $\sum H_{ui} / |U|$. Average hit ratio is calculated as the sum of the hit ratio for each push attack on item i across all pushed items divided by the number of pushed items.

Hit ratio is useful for evaluating the pragmatic effect of a push attack on recommendation. Typically, a user is only interested in the top 20 to 50 recommendations. An attack on an item that significantly raises the hit

[1] http://www.cs.umn.edu/research/GroupLens/data/

ratio, regardless of prediction shift, can be considered effective. Indeed, an attack that causes a pushed item to be recommended 80% of the time has achieved the desired outcome for the attacker.

4.3 Accuracy Analysis

We first compare the accuracy of k-nn versus the model-based algorithms. To monitor accuracy, and to assist in tuning the recommendation algorithms, we use MAE. In all cases, 10-fold cross-validation is performed on the entire dataset and no attack profiles are injected.

In neighborhood formation, we achieved optimal results using k = 20 users for the neighborhood size of the k-nn algorithm. For the model-based algorithms, we obtained the most favorable results using k = 10 user segments for the neighborhood size. In all cases, we filter out neighbors with a similarity score less than 0.1. For pLSA, we observed an optimum threshold of μ = 0.035. We obtained the best results for PCA by extracting the factors that explain at least 60% of total variance. The average number of extracted principal components was approximately 100.

Table 1 displays the results from one of five test runs performed. The difference in accuracy between the standard and PCA approaches is not statistically significant. This is a very promising result, as the scalability of model-based algorithms often come at the cost of lower recommendation accuracy [1]. For example, k-means and pLSA show small decreases in accuracy compared to standard k-nn.

Determining a suitable evaluation metric for the Apriori recommender was challenging because it is based on a fundamentally different approach. The k-nn algorithm predicts a rating value for each target item and ranks all items based on this score. The association rule algorithm produces a ranked list, such that the recommendation score is the confidence that a target user will like the recommended item. It is not possible to make a prediction of the rating value from the association rule recommendation list. However, the association rule recommender does make a more general prediction; it predicts a binary "like" or "dislike" classification for a recommended item if the confidence value is positive or negative, respectively.

For brevity, we do not include the derived metric, but a detailed description can be found in [10]. Our results showed the difference in accuracy between the association rule recommender and k-nn to be statistically insignificant. But because Apriori selects recommendations from only among those item sets that have met the support threshold, it will by necessity have lower coverage than the other model-based algorithms. There will be some items that do not appear and about which the algorithm cannot make any predictions. This problem may occur in a k-nn algorithm as well, since there may be no peer users who have rated a given item. However, this is a relatively rare occurrence. The coverage of the k-nn algorithm is near 100%, while Apriori is consistently around 47%.

The Apriori algorithm would therefore lend itself best to scenarios in which a list of top recommended items is sought, rather than a rating prediction scenario in which the recommender must be able to estimate a rating for any given item. The selectivity of the algorithm may be one reason to expect it will be relatively robust - it will not make recommendations without evidence that meets the minimum support threshold.

4.4 Robustness Analysis

To evaluate the robustness of model-based algorithms, we compare the results of push attacks on collaborative recommendation algorithms using k-nearest neighbor, k-means clustering, pLSA, PCA, and Apriori techniques.

Table 1: Accuracy

	k-nn	k-means	plsa	pca
mae	0.7367	0.7586	0.7467	0.7327

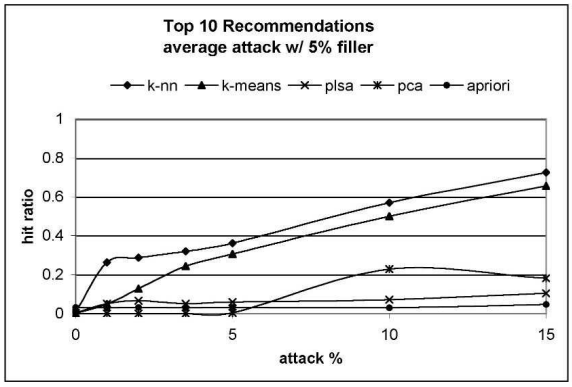

Figure 2: Average attack hit ratio at 5% filler size

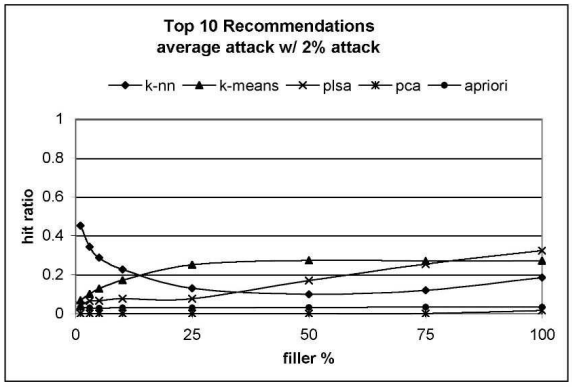

Figure 3: Average attack hit ratio at 2% attack size

We report the results for average and segment attacks, and exclude results for random attack, because average attack is more effective and exhibits similar robustness trends.

4.4.1 Average Attack

Figure 2 presents hit ratio results for top 10 recommendations at different attack sizes, using a 5% filler. With the exception of k-means, the model-based techniques show notable improvement in stability over k-nn. Apriori and pLSA, in particular, have superior performance at all attack sizes, and PCA performs extremely well at small attack sizes of 5% or less. Under a 15% attack, an attacked movie is in a user's top 10 recommended list nearly 80% of the time for k-nn and k-means. However, the attacked movie only shows up in a user's top 10 recommendations slightly greater than 5% of the time for Apriori or pLSA and less that 20% of the time for PCA.

Robustness of the Apriori algorithm may be partially due to lower coverage. However, this does not account for the flat trend of hit ratio with respect to attack size. At a 5% attack, we observed only 26% coverage of the attacked item. But at a 10% attack, we observed 50% coverage, and at 15% attack, we observed a full 100% coverage of the attacked item.

It is precisely the manner in which an average attack chooses filler item ratings that causes the combination of multiple attack profiles to short-circuit the attack. Recall that filler item ratings in an average attack are distributed around their mean rating. When an average attack profile is discretized, there is equal probability that a filler item will be categorized as "like" or "dislike". Therefore, multiple attack profiles will show little more than chance probability of having common itemsets. The lack of mutual reinforcement between filler items prevents the support of itemsets containing the attacked item from surpassing the threshold.

To evaluate the sensitivity of filler size, we have tested a full range of filler items. The 100% filler is included as a benchmark for the potential influence of an attack. However, it is not likely to be practical from an attacker's point of view. Collaborative filtering rating databases are often extremely sparse, so attack profiles that have rated every product are quite conspicuous.Of particular interest are smaller filler sizes. An attack that performs well with few filler items is less likely to be detected. Thus, an attacker will have a better chance of actually impacting a system's recommendation, even if the performance of the attack is not optimal.

Figure 3 depicts hit ratio for top 10 recommendations at the full range of filler sizes with a 2% attack size. Surprisingly, as filler size is increased, hit ratio for standard k-nn goes down. This is because an attack profile with many filler items has greater probability of being dissimilar to the active user. On the contrary, hit ratio for k-means and pLSA tend to rise with larger filler sizes. Eventually, both algorithms are surpassed by k-nn and actually perform worse with respect to robustness.

However, the PCA and Apriori algorithms hold steady at large filler sizes and are essentially unaffected. As

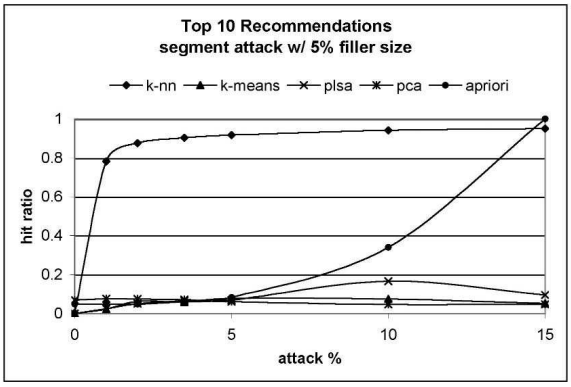

Figure 4: Segment attack hit ratio at 5% filler size

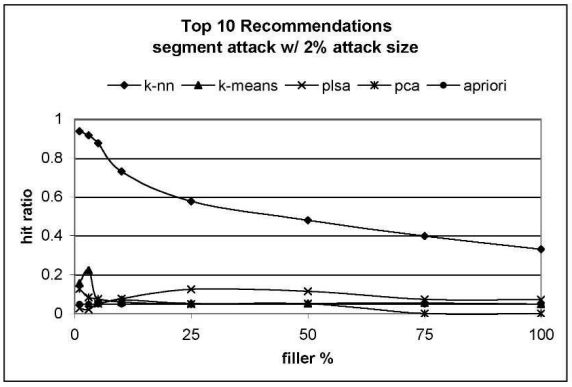

Figure 5: Segment attack hit ratio at 2% attack size

with attack size, the reason that filler size does not affect the robustness of Apriori is because adding more filler items does not change the probability that multiple attack profiles will have common itemsets. The fact that a profile's ratings are discretized to categories of "like" and "dislike" means that an attack profile with 100% filler size will cover exactly half of the total features used in generating frequent itemsets. Therefore, it is very unlikely that multiple attack profiles will result in mutual reinforcement.

4.4.2 Segment Attack

The segment attack is designed to have particular impact on likely buyers, or "in-segment" users. These users have shown a disposition towards items with particular characteristics, such as movies within a particular genre. For our experiments, we selected popular horror movies (Alien, Psycho, The Shining, Jaws, and The Birds) and identified users who had rated all of them as 4 or 5. This is an ideal target market to promote other horror movies, and so we measure the impact of the attack on recommendations made to the in-segment users.

Figure 4 depicts hit ratio for top 10 recommendations at different attack sizes, using a 5% filler. Clearly, the attack is extremely effective against the k-nn algorithm. A meager 1% attack shows a hit ratio of nearly 80%. By contrast, a segment attack has little effect on k-means, pLSA, and PCA.

The Apriori algorithm appears to have the same robustness as the other model-based algorithms at small attack sizes. However, beyond a 5% attack, Apriori performs quite poorly with respect to robustness. Hit ratio reaches 100% at a 15% attack. The cause of such dramatic effect is precise targeting of selected items by the attacker. This is the opposing force to the phenomena witnessed against an average attack. A segment attack profile consists of multiple selected items, in addition to the target item, where the maximum rating is assigned. Clearly, all such items will always be categorized as "like". Therefore, the mutual reinforcement of common item sets is a given, and a user that likes any permutation of the selected items receives the attacked item as a recommendation with high confidence.

Although the performance of Apriori is not ideal against a segment attack, certain scenarios may minimize the performance degradation in practice. In particular, a recommender system with a very large number of users is somewhat buffered from attack. The algorithm is quite robust through a 5% attack, and is comparable to both k-means, pLSA, and PCA. The robustness of Apriori is not drastically reduced until attack size is 10% or greater. Certainly it is feasible for an attacker to inject the necessary number of profiles into a recommender with a small number of users, but it may not be economical for a commercial recommender such as Amazon.com, with millions of users.

5 Conclusion

The standard user-based collaborative filtering algorithm has been shown quite vulnerable to profile injection attacks. An attacker is able to bias recommendation by building a number of profiles associated with fictitious identities. In this paper, we have demonstrated the relative robustness and stability of model-based algorithms over the memory-based approach.

References

[1] M. O'Conner and J. Herlocker, "Clustering items for collaborative filtering," in *Proceedings of the ACM SIGIR Workshop on Recommender Systems*, Berkeley, CA, August 1999.

[2] S. Lam and J. Riedl, "Shilling recommender systems for fun and profit," in *Proceedings of the 13th International WWW Conference*, New York, May 2004.

[3] M. O'Mahony, N. Hurley, N. Kushmerick, and G. Silvestre, "Collaborative recommendation: A robustness analysis," *ACM Transactions on Internet Technology*, vol. 4, no. 4, pp. 344–377, 2004.

[4] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams, "Towards trustworthy recommender systems: An analysis of attack models and algorithm robustness," *ACM Transactions on Internet Technology*, vol. 7, no. 4, 2007.

[5] B. Mobasher, R. Burke, R. Bhaumik, and J. Sandvig, "Attacks and remedies in collaborative recommendation," *IEEE Intelligent Systems*, vol. 22, no. 3, pp. 56–63, May/June 2007.

[6] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*, Santiago, Chile, September 1994.

[7] 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.

[8] T. Hofmann, "Probabilistic latent semantic analysis," in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, Stockholm, Sweden, July 1999.

[9] B. Mobasher, R. Burke, and J. Sandvig, "Model-based collaborative filtering as a defense against profile injection attacks," in *Proceedings of the 21st National Conference on Artificial Intelligence*. AAAI, July 2006, pp. 1388–1393.

[10] J. J. Sandvig, B. Mobasher, and R. Burke, "Robustness of collaborative recommendation based on association rule mining," in *Proceedings of the 2007 ACM Conference on Recommender Systems (RecSys'07)*, October 2007, pp. 105–111.

[11] J.Herlocker, J. Konstan, L. G. Tervin, and J. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems*, vol. 22, no. 1, pp. 5–53, 2004.