Robustness of Collaborative Recommendation Based On Association Rule Mining *

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

ABSTRACT

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 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 a recommendation algorithm based on the data mining technique of association rule mining. Our results show that the Apriori algorithm offers large improvement in stability and robustness compared to k-nearest neighbor and other model-based techniques we have studied. Furthermore, our results show that Apriori can achieve comparable recommendation accuracy to k-nn.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*data mining*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Algorithms, Human Factors, Security

Keywords

Recommender systems, collaborative filtering, association rule mining, data mining, security

1. INTRODUCTION

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. However, in some cases, this is at the cost of lower recommendation accuracy [13].

A positive side effect of a model-based approach is that it may provide improved robustness against profile injection attacks. 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.

Previous work has shown the vulnerability of the basic k-nearest neighbor algorithm to attack [10]. Model-based algorithms that cluster similar users have shown different degrees of improvement with respect to robustness. Two successful approaches include k-means clustering and, particularly, probabilistic latent semantic analysis (PLSA) [11].

In this paper we explore the robustness of a recommendation algorithm based on association rule mining. 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 [1] 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 an association rule based recommender provides an algorithmic approach to robust recommendation. Our implementation shows significant improvement in stability compared to the standard memory-based k-nearest neighbor, and achieves comparable accuracy.

Furthermore, experimental results suggest that the association rule based recommender is more robust than other model-based techniques we have studied, particularly against non-focused attacks and attack profiles with large numbers of ratings. This provides further evidence that model-based algorithms are more robust than k-nn.

2. PROFILE INJECTION ATTACKS

We assume that an attacker intends to bias a recommender system 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.

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 promote or demote the recommendation of a particular

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

RecSys'07, October 19–20, 2007, Minneapolis, Minnesota, USA.

	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

item. We call such attacks *profile injection attacks* (also known as *shilling* [9]).

2.1 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 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.

2.2 Attack Types

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.

A variety of attack types have been studied for their effectiveness against different recommendation algorithms [10, 11]. 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 [9, 10]. 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 [9, 10]. 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 [10]. 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.

3. RECOMMENDATION ALGORITHMS

We first present a collaborative recommendation algorithm based on association rule mining. We next provide background information on several user-based recommenders included as baseline algorithms. This includes the standard memory-based k-nn and two model-based algorithms that cluster user profiles.

3.1 Association Rule Mining

Association rule mining is a common technique for performing market basket analysis. The intent is to gain insight into customers' buying habits and discover groups of products that are commonly purchased together. As an example, an association rule may show that 98% of all customers that purchase frozen pizza also purchase soda.

Association rules capture relationships among items based on patterns of co-occurrence across transactions. In [12], association rules were applied to personalization based on web usage data. We have adapted this approach to the context of collaborative filtering. Considering each user profile as a transaction, it is possible to use the Apriori algorithm [1] 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 $\alpha_r/\sigma(Y)$.

If there is not enough support for a particular item, that item will never appear in any frequent item set. The implication is that such an item will never be recommended. The issue of coverage is a tradeoff. Lowering the support threshold will ensure that more items can be recommended, but at the risk of recommending an item without sufficient evidence of a pattern.

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.

Discretizing the dataset effectively doubles the total number of features used in analysis, but is necessary for inferring recommendable items. 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. It is clear that 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.

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 confidence of 90% whereas another rule may add the same item with a confidence of 5%. In this case, we simply use the rule with the highest confidence.

Another possibility is that 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 not an ideal solution in this case, 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.

To facilitate the search for item sets, we store the frequent item sets in a directed acyclic graph, called a *Frequent Itemset Graph* [12]. The graph is organized into levels from 0 to k, where k is the maximum size among all frequent item sets. Each node at depth d in the graph corresponds to an item set I of size d and is linked to item sets of size $d+1$ that contain I. The root node at level 0 corresponds to the empty item set. Each node also stores the support value of the corresponding frequent item set.

Given a target user profile u, we perform a depth-first search of the Frequent Itemset Graph. When we reach a node whose frequent item set I_n is not contained in u, the item $i \in I_n$ not found in u is added to the candidate set C and search at the current branch is ceased.

Note that the item set of the parent node I_p to I_n must be contained in u by definition, and because I_n is of size $d+1$ where I_p is size d, there can be only one item $i \in I_n$ that is not contained in u. It follows that $I_n = I_p \cup \{i\}$ and the two nodes correspond to the rule $I_p \implies \{i\}$. We calculate the confidence of the rule as $\sigma(I_n)/\sigma(I_p)$. The candidate $i \in C$ is stored in a hashtable along with it's confidence value. If it already exists in the hashtable, the highest confidence value takes precedent.

After completion of the depth-first search, all possible candidates for the target user u are contained in C, including items labeled "dislike". In order to properly represent an estimated negative connotation, items labeled "dislike" are given a recommendation score that is the negation of the confidence value. If there is a corresponding "like" label for the item in C, it's recommendation score is decreased by the confidence of the "dislike" label. As a final step, the candidate set C is sorted according to the recommendation scores and the top N items are returned as a recommendation.

3.2 Baseline Algorithms

User-based collaborative filtering algorithms attempt to discover a neighborhood of user profiles that are similar to a target user. A rating value is predicted for all missing items in the target user's profile, based on ratings given to the item within the neighborhood. A ranked list is produced, and typically the top 20 or 50 predictions are returned as a recommendation.

3.2.1 k-Nearest Neighbor

The standard k-nearest neighbor algorithm is widely used and reasonably accurate [3]. Similarity is computed using Pearson's correlation coefficient, and 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 use Resnick's algorithm to compute the prediction for a target item i and target user u:

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

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.

3.2.2 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.

3.2.3 Probabilistic Latent Semantic Analysis

Probabilistic latent semantic analysis (PLSA) models [4] provide a probabilistic approach for characterizing latent or hidden semantic associations among co-occurring objects. In [8, 7] PLSA was applied to the creation of user clusters based on web usage data. We have adapted this approach to the context of collaborative filtering [11].

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:

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

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 following likelihood $L(U, I)$ of the rating data:

$$L(U,I) = \sum_{u \in U} \sum_{i \in I} r_{u,i} \bullet \log Pr(u,i)$$

where $r_{u,i}$ is the rating of user u for item i.

The Expectation-Maximization (EM) algorithm [2] is used to perform maximum likelihood parameter estimation. Based on initial values of $Pr(z_k)$, $Pr(u|z_k)$, and $Pr(i|z_k)$, the algorithm alternates between an expectation step and maximization step. In the expectation step, posterior probabilities are computed for latent variables based on current estimates:

$$Pr(z_k|u,i) = \frac{Pr(z_k) \bullet Pr(u|z_k) \bullet Pr(i|z_k)}{\sum_{k'=1}^{l} Pr(z'_k) \bullet Pr(u|z'_k) \bullet Pr(i|z'_k)}$$

In the maximization step, Lagrange multipliers [5] are used to obtain the following equations for re-estimated parameters:

$$Pr(z_k) = \frac{\sum_{u \in U} \sum_{i \in I} r_{u,i} \bullet Pr(z_k|u,i)}{\sum_{u \in U} \sum_{i \in I} r_{u,i}}$$

$$Pr(u|z_k) = \frac{\sum_{i \in I} r_{u,i} \bullet Pr(z_k|u,i)}{\sum_{u' \in U} \sum_{i \in I} r_{u',i} \bullet Pr(z_k|u',i)}$$

$$Pr(i|z_k) = \frac{\sum_{u \in U} r_{u,i} \bullet Pr(z_k|u,i)}{\sum_{u \in U} \sum_{i' \in I} r_{u,i'} \bullet Pr(z_k|u,i')}$$

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.

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 previous sections, where the neighborhood V is the set of user cluster aggregate profiles most similar to the target user.

4. EXPERIMENTAL EVALUATION

To evaluate the robustness of our recommendation algorithm based on association rule mining, we compare the results of push attacks using different parameters. In each case, we report the relative improvement over the k-nearest neighbor, k-means, and PLSA 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 [6]. 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 [14].

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

Table 1: Normalized Mean Absolute Error

	run 1	run 2	run 3	mean	stdev
apriori	0.3293	0.3292	0.3287	0.3291	0.0003
k-nn	0.3332	0.3337	0.3339	0.3336	0.0003

Table 2: Coverage

	run 1	run 2	run 3	mean	stdev
apriori	0.4701	0.4716	0.4718	0.4712	0.0009
k-nn	0.9942	0.9941	0.9942	0.9942	0.0002

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:

$$HitRatio_i = \sum_{u \in U} H_{ui} / |U| .$$

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 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 analyze the accuracy of our association rule recommender compared to k-nearest neighbor. To evaluate the recommendations, we performed 10-fold cross-validation on the entire dataset and no attack profiles were injected.

Determining a suitable evaluation metric was challenging because the two algorithms are based on fundamentally different approaches. The k-nn algorithm predicts a rating value for each target item and ranks all items based on this score. However, 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 obvious how to directly compare the recommendation scores of the two algorithms, because k-nn uses a predicted value and association rules use a confidence measure. It is also 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.

In order to compare the accuracy of our association rule recommender and k-nn, we transform both recommendation lists into binary "like" and "dislike" categories for each item. For the association rule recommender, it is simply a matter of using the sign of the confidence value, as discussed in the previous paragraph. For k-nn, we categorize an item as "like" if the predicted rating is greater than the user's mean rating, and "dislike" otherwise.

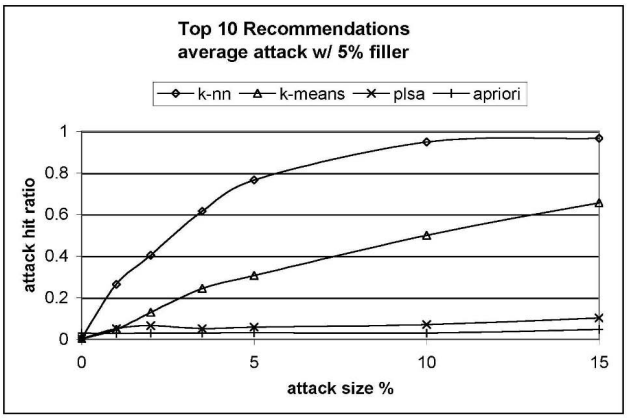

Figure 2: Average attack hit ratio at 5% filler size

It is now possible to compare the accuracy of the two algorithms. We use the normalized mean absolute error (NMAE) metric. For the Apriori algorithm, we use a minimum support of 0.1 and a maximum of 6000 rules with the greatest lift. In neighborhood formation for k-nn, we achieved optimal results using $k = 20$ users.

As shown in Table 1, the difference in accuracy between the association rule recommender and k-nn is statistically insignificant. This is a very promising result, as the scalability of model-based algorithms often come at the cost of lower recommendation accuracy [13].

Because Apriori selects recommendations from only among those item sets that have met the support threshold, it will by necessity have lower coverage than our baseline algorithms. There will be some items that do not appear in the Frequent Itemset Graph, 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 as Table 2 shows. We see that 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 our association rule recommender, we compare the results of push attacks on k-nearest neighbor, k-means clustering, and PLSA.

4.4.1 Average Attack

Figure 2 presents hit ratio results for top 10 recommendations at different attack sizes, using a 5% filler. All model-based techniques show notable improvement in stability over k-nn. However, the performance of Apriori and PLSA are superior to k-means at large attack sizes. Under a 15% at-

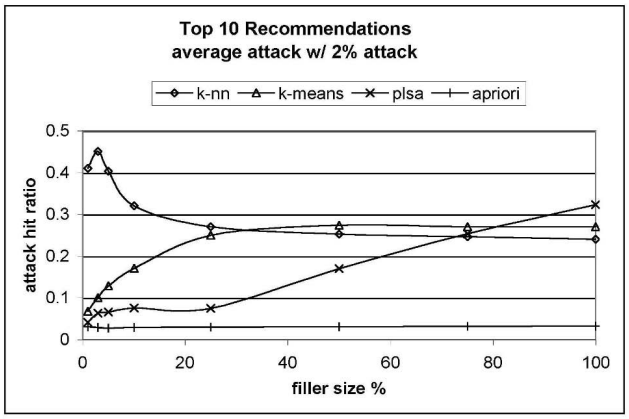

Figure 3: Average attack hit ratio at 2% attack size

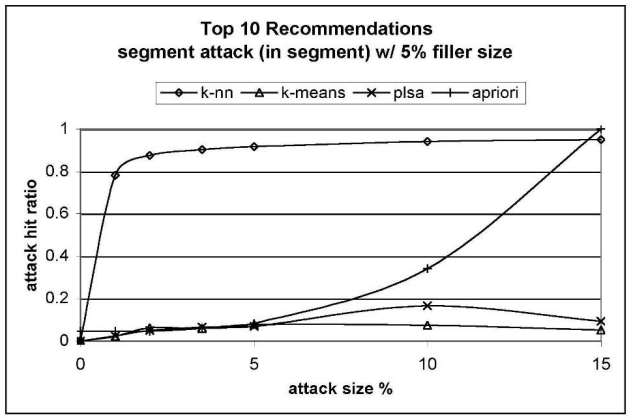

Figure 4: Segment attack hit ratio at 5% filler size

tack, an attacked movie is almost guaranteed to be in a user's top 10 recommended list for k-nn and will be in the top 10 list over 60% of the time for 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.

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 [15]. 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.

Only the Apriori algorithm holds steady at large filler sizes and is essentially unaffected. As with attack size, the reason that filler size does not affect the robustness of the algorithm 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.

We have shown results for average attack because it is more effective than random or bandwagon attacks; however, Apriori has also exhibited improved robustness compared to the other algorithms against these attacks. We next present results for segment attack.

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 and PLSA.

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 and PLSA. 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. In particular, we have introduced a robust recommendation algorithm based on association rule mining that attempts to capture relationships among items based on patterns of co-occurrence across user profiles.

Frequent item sets are generated for the association rules by first discretizing all user profiles such that an item rating is classified as "like" or "dislike". This level of abstraction from the original user profiles acts to short-circuit the mutual reinforcement property found in average and random attacks. It allows the algorithm to make recommendations that are relatively accurate, while removing much of the influence of biased attack profiles.

Overall, the association rule recommender far outperforms the standard k-nearest neighbor algorithm with respect to robustness. However, it is not robust against a segment attack compared to other model-based algorithms. This issue is slightly offset because a segment attack must be relatively large before having an effect on the algorithm. It is unlikely that a malicious user could mount a successful segment attack against a commercial recommender with millions of users.

Future work will study in greater detail the mutual reinforcement of common item sets. We will attempt to discover the ideal attack size threshold necessary for mounting a successful segment attack against the association rule recommender. In addition, we will design an extension to the algorithm that is suitable for recommender systems with a small number of users. We will also compare the accuracy of Apriori with k-nn at lower coverage levels. This might be accomplished by filtering out weak recommendations or weak neighbors in the k-nn algorithm, for example.

6. REFERENCES

[1] 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.

[2] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of Royal Statistical Society*, B(39):1–38, 1977.

[3] 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.

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

[5] T. Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning Journal*, 42(1):177–196, 2001.

[6] J.Herlocker, J. Konstan, L. G. Tervin, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.

[7] X. Jin, Y. Zhou, and B. Mobasher. A unified approach to personalization based on probabilistic latent semantic models of web usage and content. In *Proceedings of the AAAI 2004 Workshop on Semantic Web Personalization (SWP'04)*, San Jose, California, July 2004.

[8] X. Jin, Y. Zhou, and B. Mobasher. Web usage mining based on probabilistic latent semantic analysis. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'04)*, Seattle, Washington, August 2004.

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

[10] 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*, 7(4), 2007.

[11] 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*, pages 1388–1393. AAAI, July 2006.

[12] M. Nakagawa and B. Mobasher. A hybrid web personalization model based on site connectivity. In *WebKDD Workshop at the ACM SIGKKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, August 2003.

[13] 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.

[14] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International World Wide Web Conference*, Hong Kong, May 2001.

[15] C. Williams, R. Bhaumik, R. Burke, and B. Mobasher. The impact of attack profile classification on the robustness of collaborative recommendation. In *Proceedings of the 2006 WebKDD Workshop, held at ACM SIGKDD Conference on Data Mining and Knowledge Discovery (KDD'06)*, Philadelphia, August 2006.