Model-Based Collaborative Filtering as a Defense Against Profile Injection Attacks [*]

Bamshad Mobasher and **Robin Burke** and **JJ Sandvig**
Center for Web Intelligence
School of Computer Science, Telecommunication and Information Systems
DePaul University, Chicago, Illinois
{mobasher, rburke, jsandvig}@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, have been shown to be quite vulnerable to such attacks. In this paper, we examine the robustness of model-based recommendation algorithms in the face of profile injection attacks. In particular, we consider two recommendation algorithms, one based on k-means clustering and the other based on Probabilistic Latent Semantic Analysis (PLSA). These algorithms aggregate similar users into user segments that are compared to the profile of an active user to generate recommendations. Traditionally, model-based algorithms have been used to alleviate the scalability problems associated with memory-based recommender systems. We show, empirically, that these algorithms also offer significant improvements in stability and robustness over the standard k-nearest neighbor approach when attacked. Furthermore, our results show that, particularly, the PLSA-based approach can achieve comparable recommendation accuracy.

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 (Burke *et al.* 2005; Burke, Mobasher, & Bhaumik 2005; Lam & Reidl 2004; O'Mahony *et al.* 2004). 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 (for example, a number of profiles that assign high ratings to a particular item), these biased profiles may be considered peers for genuine users and result in biased recommendations. This is precisely the effect found in (Lam & Reidl 2004) and (O'Mahony *et al.* 2004).

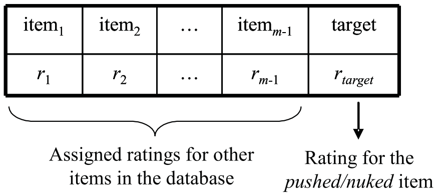
A widely accepted approach to user-based collaborative filtering is the k-nearest neighbor algorithm. However, memory-based algorithms such as k-NN do not scale well to commercial recommender systems. For such data-intensive systems, model-based algorithms provide a scalable solution. Building a model 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 (O'Conner & Herlocker 1999).

A potentially positive side-affect of a model-based approach is that it may provide improved robustness against profile injection attacks. Previous work has shown the vulnerability of the basic k-nearest neighbor algorithm to attack (Burke *et al.* 2005). However, a model-based approach is an abstraction of the detailed user profiles. We hypothesize that this abstraction will minimize the influence of an attack, because the attack profile is not directly used in recommendation.

In our study, we have focused on two model-based algorithms that cluster similar users into segments. Each segment represents an aggregate profile that is used for recommendation, rather than the original user data. Our benchmark standard applies k-means clustering to create the user segments. In (O'Conner & Herlocker 1999) a similar approach was used to improve scalability of k-NN, but resulted in somewhat lower prediction accuracy. We present a more successful approach based on probabilistic latent semantic analysis (PLSA) to infer hidden relationships among groups of users. PLSA is a "fuzzy" approach, in that each user has a degree of association with every user segment. This allows particularly authoritative users to exercise greater influence on recommendation.

The primary contribution of this paper is to demonstrate that model-based algorithms, and in particular the PLSA-based algorithm, offer significant improvement in robustness against profile injection attacks when compared with the standard memory-based k-NN. In addition, this improvement in robustness does not come at a significant cost in terms of recommendation accuracy.

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



item_1	item_2	…	$\text{item}_{m\text{-}1}$	target
r_1	r_2	…	$r_{m\text{-}1}$	r_{target}

Assigned ratings for other items in the database
Rating for the *pushed/nuked* item

Figure 1: General form of an attack profile

Attack Types

A profile injection attack against a collaborative recommender system consists of a number of *attack profiles* added to the database of real user profiles. The goal of a push attack is to increase the system's predicted rating on a *target item* for a given user (or group of users). 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.

The generic form of an attack profile is depicted in Figure 1. Specific attack types define the method for assigning ratings to the set of *filler items* and the target item. The set of filler items represent a group of randomly selected items in the database that are assigned ratings within the attack profile. In certain attack types, a subset of filler items may be pre-selected for a precise impact. The target item in a push attack is generally given the maximum allowed rating.

The *random attack* and *average attack* are basic attack types introduced in (Lam & Reidl 2004) and further generalized in (Burke, Mobasher, & Bhaumik 2005). In both cases, filler items of an attack profile are assigned random ratings. For a random attack, the ratings are distributed around the global rating mean. For an average attack, the ratings are distributed around the individual mean for each filler item.

In practice, an average attack is much more effective than a random attack. However, it requires greater knowledge about the system's rating distribution. This knowledge cost is minimized by the fact that 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 extension of the random attack, the *bandwagon attack* (Burke *et al.* 2005; Burke, Mobasher, & Bhaumik 2005) is nearly as effective as the average attack. The goal of a bandwagon attack is to associate the target item with a small number of frequently rated items. This takes advantage of the Zipf 's law distribution: a small number of items will receive the lion's share of ratings. In a bandwagon attack, a small set of frequently rated items are selected along with the set of random filler items. Attack profiles give maximum rating to those items that have high visibility, and therefore have a good probability of being similar to a large number of users.

Random, average, and bandwagon attack types are not particularly effective against item-based collaborative filtering. In response, the *segment attack* was introduced (Burke *et al.* 2005; Burke, Mobasher, & Bhaumik 2005). It turns out that a segment attack is also quite effective against user-based algorithms. A segment attack attempts to target a specific group of users who may already be predisposed toward the target item. 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 group of users, in addition to the random filler items. This differs from a bandwagon attack in that the selected items are expected to be highly rated within the targeted user group, rather than frequently rated. The selected segment items are assigned the maximum rating value along with the target item. To provide the greatest impact on item-based algorithms, all remaining filler items are given the minimum allowed rating.

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 two model-based recommendation algorithms that cluster user profiles into user segments. The first is based on k-means clustering and the second on Probabilistic Latent Semantic Analysis.

k-Nearest Neighbor

k-nearest neighbor is a memory-based algorithm dependent on direct user-to-user similarity (Herlocker *et al.* 1999). It operates by selecting the k most similar users to a target user, and formulates a prediction for a target item by combining the preferences of these users. k-NN is widely used and reasonably accurate. The similarity between the target user, u, and a neighbor, v, is computed using the standard Pearson's correlation coefficient:

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

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.

Once similarities are calculated, the k most similar users that have rated the target item are selected as the neighborhood. Note that this implies a target user may have a different neighborhood for each target item. We also filter out all neighbors with a 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:

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

where V is the set of k similar neighbors; $r_{v,i}$ 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 described above.

The formula in essence computes the degree of preference for all neighbors, weighted by their similarity, and then adds this to the target user's average rating: the idea being that different users may have different "baselines" around which their ratings are distributed. Note that if the denominator of the above equation is zero, the prediction is simply the target user's average rating.

k-Means Clustering

A standard model-based collaborative filtering algorithm uses k-means to cluster similar users into segments. 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. We have used the Pearson correlation described above. The discovered user segments are then applied to the user-based neighborhood formation task, rather than individual profiles.

In order to determine similarity between a target user and a user segment, we must aggregate each cluster into a comparable form with the target user. Profile Aggregation based on Clustering Transactions (PACT) provides an effective method for the derivation of aggregate representations from the user profile clusters (Mobasher, Dai, & T. Luo 2002). For each cluster C_k, we compute the mean vector $\vec{v_k}$ of common item ratings corresponding to each user segment:

$$\vec{v_k} = \frac{1}{|C_k|} \sum \vec{u_n}$$

where $\vec{u_n}$ is the rating vector for a user profile $u_n \in C_k$.

To make a recommendation for a target user u and target item i, we select a neighborhood of user segments 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. Given the aggregate profile of a user segment contains the average rating for each item within the segment, we can make a prediction for item i as described in the previous section, where the neighborhood V is the set of user segment aggregate profiles most similar to the target user.

Probabilistic Latent Semantic Analysis

Probabilistic latent semantic analysis (PLSA) models (Hofmann 1999) provide a probabilistic approach for characterizing latent or hidden semantic associations among co-occurring objects. In (Jin, Zhou, & Mobasher 2004) PLSA was applied to the creation of user segments based on web usage data. We have adapted this approach to the context of collaborative filtering.

The process of discovering user segments via PLSA is as follows. 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 (Dempster, Laird, & Rubin 1977) 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 (Hofmann 2001) 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 can now identify segments of users that have similar underlying interests. For each latent variable z_k, we create a user segment 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 segment of highest probability. Thus, every user profile will be associated with at least one user segment, but may be associated with multiple segments. This allows authoritative users to have broader influence over predictions, without adversely affecting coverage in sparse rating data.

For each user segment C_k, we can aggregate the associated user profiles into a weighted profile vector $\vec{v_k}$:

$$\vec{v_k} = \frac{\sum \vec{u_n} \bullet Pr(u_n|z_k)}{\sum |Pr(u_n|z_k)|}$$

where $\vec{u_n}$ is the rating vector for a user profile $u_n \in C_k$.

To make a recommendation for a target user u and target item i, we select a neighborhood of user segments 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 segment aggregate profiles most similar to the target user.

Evaluation Metrics

There has been considerable research in the area of recommender system evaluation focused on accuracy and performance (Herlocker *et al.* 2004). Our goal, on the other hand, is to measure the effectiveness of an attack, i.e., the "win" for the attacker. In the experiments reported below, we follow the lead of (O'Mahony *et al.* 2004) in measuring stability via prediction shift. In addition, we measure hit ratio for a push attack. Hit ratio measures the average likelihood that a top N recommender will recommend a pushed item, compared to all other items (Sarwar *et al.* 2001).

Prediction shift measures the change in an item's predicted rating after being attacked. Let U and I be the sets of test users and attacked items, respectively. For each user-item pair (u, i) the prediction shift denoted by $\Delta_{u,i}$, can be measured as $\Delta_{u,i} = p'_{u,i} - p_{u,i}$, where p and p' represent the prediction before and after attack, respectively. A positive value means that the attack has succeeded in raising the predicted rating for the item. The average prediction shift for an item i over all users in the test set can be computed as $\sum_{u \in U} \Delta_{u,i} / |U|$. The average prediction shift is then computed by averaging over individual prediction shifts for all attacked items.

Note that a strong prediction shift does not guarantee an item will be recommended - it is possible that other items' scores are also affected by an attack, or that the item score is so low that even a prodigious shift does not promote it to "recommended" status.

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

Experimental Evaluation

To evaluate the robustness of the model-based k-means and PLSA algorithms, we compare the results of push attacks using different parameters. In each case we report the relative improvement over the standard k-nearest neighbor approach.

Data and Test Sets

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. An exception is made for segment attacks when evaluating in-segment users. In this case, the test set contains only user profiles that have rated every segment movie with a four or five. If there are more than 50 in-segment users, a random sample of 50 is used as the test set. The remainder of user profiles after removing the test set is designated as the training set. All collaborative filtering models and 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.

Results

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 the mean absolute error (MAE) metric. MAE is a statistical measure for comparing predicted values to actual user ratings (Herlocker *et al.* 1999). 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.

As Figure 2 shows, k-means and PLSA are not remarkably less accurate than k-NN, with PLSA being somewhat more accurate than k-means. For the remainder of the experiments, we apply 30 user segments for both k-means and PLSA. Although a greater number of user segments result in

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

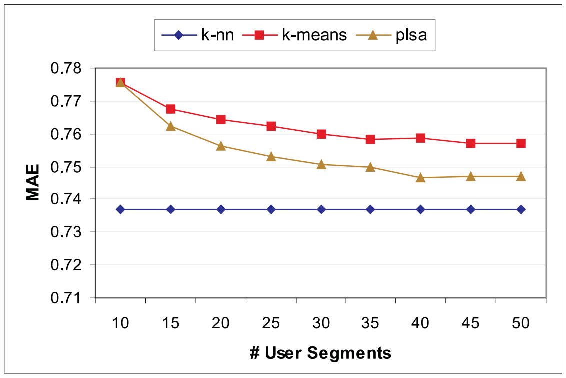


Figure 2: Comparison of MAE

improved MAE, 30 seems to be the point of diminishing returns. Larger segments require considerably more processing time in order to build a model, with marginal improvement for our purposes.

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. We use the metrics of prediction shift and hit ratio to measure the relative performance of various attack models.

To evaluate the sensitivity of filler size, we have tested 5%, 10%, 25%, and 100% filler items on each attack type. 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.

Figure 3 presents the prediction shift results of an average attack at different attack sizes using a 5% filler. Both k-means and PLSA show notable improvement in stability over k-NN. However, the performance of PLSA is superior to k-means at larger attack sizes. At a 15% attack, prediction shift for PLSA is 0.8 - less than half that of k-means (1.7), and one third that of k-NN (2.6).

Figure 4 presents hit ratio for an average attack using a 5% filler size and 15% attack size. Clearly, PLSA is more robust than k-means or k-NN. Even at a top 50 recommendation, PLSA includes an attacked item just over 25% of the time, compared to nearly 100% for each of the other algorithms.

Figure 5 depicts prediction shift for an average attack at different filler sizes using a 15% attack size. Although less remarkable, PLSA continues to outperform k-means and k-NN at larger filler sizes. At 25% filler, prediction shift for PLSA is just over 1.0, compared to 1.9 and 2.1 for k-means and k-NN, respectively. In fact, it is not until 100% filler that either of the other algorithms are able to match the performance of PLSA. Note that as filler size is increased, k-NN prediction shift goes down. This is because an attack profile with many filler items has greater chance of being dissimilar to the active user.

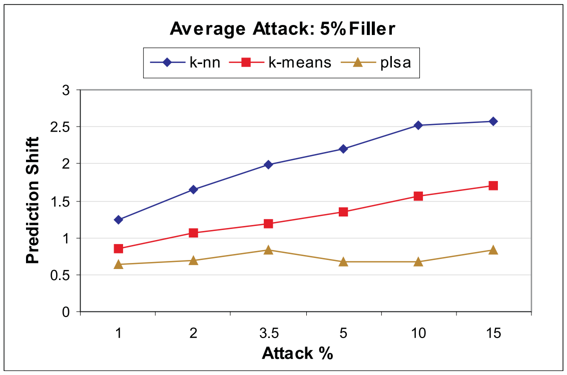


Figure 3: Average attack prediction shift at 5% filler

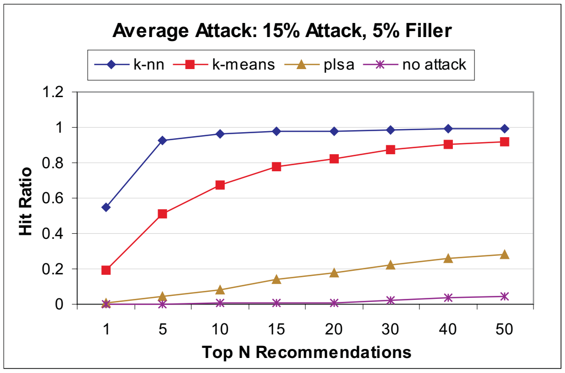


Figure 4: Average attack hit ratio at 5% filler and 15% attack

We have shown results for average attack because it is more effective than random or bandwagon attacks; however, PLSA has also exhibited improved robustness over k-means and k-NN against these attacks. We next present results for 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 4 or 5. This would be an ideal target market to which other horror movies could be promoted, and so we measure the impact of the attack on recommendations made to them. Note that in-segment in context of a segment attack should not be confused with a "user segment", which we use as a model for prediction.

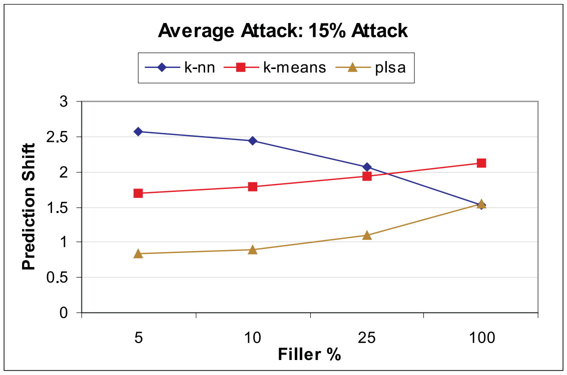


Figure 5: Comparison of filler size for an average attack

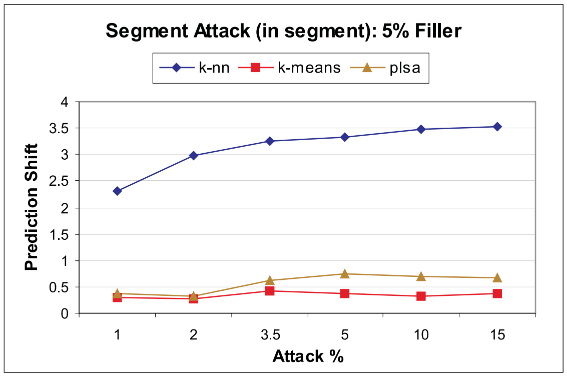


Figure 6: Segment attack prediction shift at 5% filler (for in-segment users)

Figure 6 depicts prediction shift for a segment attack using a 5% filler. Clearly, the attack is extremely effective against the k-NN algorithm. A minimal 1% attack shows a prediction shift of almost 2.5 - the same result as a 15% average attack. Although not displayed, our hit ratio results confirm these findings.

By contrast, k-means and PLSA are less affected by a segment attack. Even at 100% filler size, k-means continues to hold at 0.5 and PLSA never surpasses a prediction shift of 1. Although k-means is slightly more robust than PLSA against a segment attack, this is offset by the poor performance against the other attack types.

Conclusions

Recent research has shown the vulnerability of the standard memory-based k-nearest neighbor algorithm to profile injection attacks. An attacker is able to bias a memory-based 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 focused on a recommendation algorithm based on Probabilistic Latent Semantic Analysis in order to discover segments of similar users that are compared to the profile of an active user to generate recommendations. This level of abstraction from the original user profiles allows PLSA to make recommendations that are relatively accurate, while removing much of the influence of biased attack profiles.

References

Burke, R.; Mobasher, B.; Zabicki, R.; and Bhaumik, R. 2005. Identifying attack models for secure recommendation. In *Beyond Personalization: A Workshop on the Next Generation of Recommender Systems*.

Burke, R.; Mobasher, B.; and Bhaumik, R. 2005. Limited knowledge shilling attacks in collaborative filtering systems. In *Proceedings of the 3rd IJCAI Workshop in Intelligent Techniques for Personalization*.

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

Herlocker, J.; Konstan, J.; Borchers, A.; and Riedl, J. 1999. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd ACM Conference on Research and Development in Information Retrieval (SIGIR'99)*.

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

Hofmann, T. 1999. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*.

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

Jin, X.; Zhou, Y.; and Mobasher, B. 2004. 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)*.

Lam, S., and Reidl, J. 2004. Shilling recommender systems for fun and profit. In *Proceedings of the 13th International WWW Conference*.

Mobasher, B.; Dai, H.; and T. Luo, M. N. 2002. Discovery and evaluation of aggregate usage profiles for web personalization. *Data Mining and Knowledge Discovery* 6:61–82.

O'Conner, M., and Herlocker, J. 1999. Clustering items for collaborative filtering. In *Proceedings of the ACM SIGIR Workshop on Recommender Systems*.

O'Mahony, M.; Hurley, N.; Kushmerick, N.; and Silvestre, G. 2004. Collaborative recommendation: A robustness analysis. *ACM Transactions on Internet Technology* 4(4):344–377.

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