Chapter 25
Robust Collaborative Recommendation

Robin Burke, Michael P. O'Mahony and Neil J. Hurley

Abstract Collaborative recommender systems are vulnerable to malicious users who seek to bias their output, causing them to recommend (or not recommend) particular items. This problem has been an active research topic since 2002. Researchers have found that the most widely-studied memory-based algorithms have significant vulnerabilities to attacks that can be fairly easily mounted. This chapter discusses these findings and the responses that have been investigated, especially detection of attack profiles and the implementation of robust recommendation algorithms.

25.1 Introduction

Collaborative recommender systems are dependent on the goodwill of their users. There is an implicit assumption – note the word "collaborative" – that users are in some sense "on the same side", and at the very least, that they will interact with the system with the aim of getting good recommendations for themselves while providing useful data for their neighbors. Herlocker et al. [10] use the analogy of the "water-cooler chat", whereby co-workers exchange tips and opinions.

However, as contemporary experience has shown, the Internet is not solely inhabited by good-natured collaborative types. Users will have a range of purposes

Robin Burke
Center for Web Intelligence, School of Computer Science, Telecommunication and Information Systems, DePaul University, Chicago, Illinois, USA e-mail: rburke@cs.depaul.edu

Michael P. O'Mahony
CLARITY: Centre for Sensor Web Technologies, School of Computer Science and Informatics, University College Dublin, Ireland e-mail: michael.p.omahony@ucd.ie

Neil J. Hurley
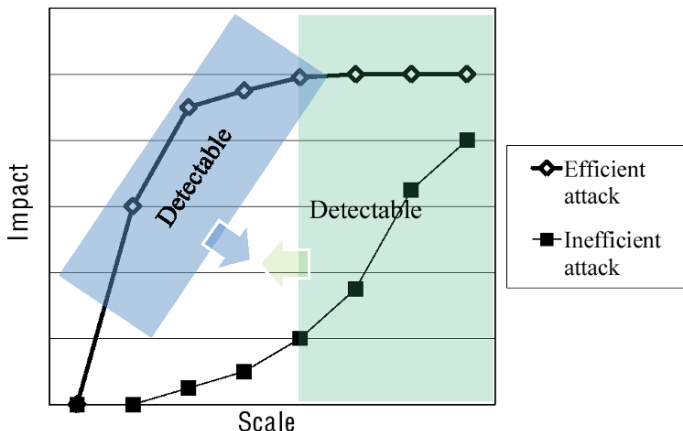School of Computer Science and Informatics, University College Dublin, Ireland e-mail: neil.hurley@ucd.ie

Fig. 25.1: Curves show the theoretical impact of attacks of different degrees of efficiency. The shaded areas shows attacks that can be detected.

in interacting with recommender systems, and in some cases, those purposes may be counter to those of the system owner or those of the majority of its user population. To cite a well-known example, the Google search engine finds itself engaging in more-or-less continual combat against those who seek to promote their sites by "gaming" its retrieval algorithm.

In search engine spam, the goal for an attacker is to make the promoted page "look like" a good answer to a query in all respects that Google cares about. In the case of collaborative recommendation, the goal for an adversary is to make a particular product or item look like a good recommendation for a particular user (or maybe all users) when really it is not. Alternatively, the attacker might seek to prevent a particular product from being recommended when really it is a good choice. If we assume that a collaborative system makes its recommendations purely on the basis of user profiles, then it is clear what an attacker must do – add user profiles that push the recommendation algorithm to produce the desired effect. A single profile would rarely have this effect, and in any case, fielded systems tend to avoid making predictions based on only a single neighbor. What an attacker really needs to do is to create a large number of psuedonomous profiles designed to bias the system's predictions. Site owners try to make this relatively costly, but there is an inherent tension between policing the input of a collaborative system and making sure that users are not discouraged from entering the data that the algorithm needs to do its work. The possibility of designing user rating profiles to deliberately manipulate the recommendation output of a collaborative filtering system was first raised in [24]. Since then, research has focused on attack strategies, detection strategies to combat attacks and recommendation algorithms that have inherent robustness against attack.

A framework for understanding this research is sketched in Figure 25.1. First, we demonstrate the extent of the problem by modeling *efficient* attacks, attacks that can

with relatively low cost produce a large impact on system output. This enables us to understand the shape of the impact curve for efficient attacks. Research on detection attempts to identify groups of profiles that make up an attack and to eliminate them from the database. Attacks that are not efficient are more difficult to detect, but because they are inefficient, must be very large to have an impact. A large influx of ratings for a particular item is easy to detect with standard system monitoring procedures. Research on detection therefore focuses on how to detect efficient attacks and variants of them, seeking to increase the size of the "detectable" boxes in the diagram, and thereby limiting the impact that an attacker can have. At the same time, researchers have studied a number of algorithms that are intended to be robust against attack, having lower impact curves relative to efficient attacks. With the combination of these techniques, researchers have sought, not to eliminate attacks, but to control their impact to the point where they are no longer cost-effective.

This chapter looks at each of these points in turn. In Section 25.3, we look at research that aims to identify the most efficient and practical attacks against collaborative recommender systems, establishing the shape of the impact curve suggested above. Section 25.5 looks at the problem of detection: in particular, the left-most shaded area for detecting efficient attacks. Lastly, in Section 25.6, we examine attempts to reduce the impact of attacks through robust algorithms.

25.2 Defining the Problem

A collaborative recommender is supposed to change its recommendations in response to the profiles that users add. It is somewhat counter-intuitive to suppose that "robustness" or "stability" is a desirable property in a system that is supposed to be adaptive. The goal of robust recommendation is to prevent attackers from manipulating the system through large-scale insertion of user profiles, a *profile injection* attack.

We assume that any user profile is feasible. That is, we do not want to demand that users' ratings fit with those that have been entered previously or that they make any kind of objective sense. Users are entitled to their idiosyncratic opinions and there is always the possibility that what is an unusual user today may be more typical tomorrow as new users sign up. So, a profile, taken by itself, cannot constitute an attack. Also, it is important to note that some web phenomena that look like attacks are not considered such within this definition. For example, in the Fall of 2008, numerous videogame fans converged on the page for the game *Spore* on Amazon.com, using it as a vehicle for airing their complaints about the digital rights management software included with the game. Presumably these were a large number of authentic individuals, and while their ratings no doubt skewed the recommendations for *Spore* for some time, their actions would not be considered an attack as we define it here. It is not clear that any automated technique can identify when a real user posts

a rating to make a political statement or as a prank, rather than to reflect an honest preference.[1]

For the purposes of this research, an attack is a concerted effort to bias the results of a recommender system by the insertion of a large number of profiles using false identities. Each of the separate identities assumed by the attacker are referred to as an *attack profile*. Once created, these profiles are used to insert preference data into the system. The most dangerous attacks are those that are crafted for maximum impact on the system, so much research has been devoted to finding the most effective and practical attacks against different algorithms.

While random vandalism surely does occur, research in this area has concentrated on attacks designed to achieve a particular recommendation outcome. The objectives of *product push* and *product nuke* attacks are to promote or demote the recommendations made for items, respectively. For example, the goal of an attacker might be to force a system to output poor recommendations for his competitors' products (nuke) while attempting to secure positive recommendations for his own (push).

From the perspective of the attacker, the best attack against a system is one that yields the biggest impact for the least amount of effort. There are two types of effort involved in mounting an attack. The first is the effort involved in crafting profiles. On of the crucial variables here is the amount of knowledge that is required to put together an attack. A *high-knowledge attack* is one that requires the attacker to have detailed knowledge of the ratings distribution in a recommender system's database. Some attacks, for example, require that the attacker know the mean rating and standard deviation for every item. A *low-knowledge attack* is one that requires system-independent knowledge such as might be obtained by consulting public information sources.

We assume that the attacker will have a general knowledge of the type of algorithm being employed to produce recommendations. An attacker that has more detailed knowledge of the precise algorithm in use would be able to produce an *informed attack* that makes use of the mathematical properties of the algorithm itself to produce the greatest impact.

The second aspect of effort is the number of profiles that must be added to the system in order for it to be effective. The ratings are less important since the insertion of ratings can be easily automated. Most sites employ online registration schemes requiring human intervention, and by this means, the site owner can impose a cost on the creation of new profiles. This is precisely why, from an attacker's perspective, attacks requiring a smaller number of profiles are particularly attractive.

---

[1] It could be argued that even such a technique did exist, it would not be in the interest of a collaborative system to deploy it.

Items

Users	1	2	3	4	5	6	7	
a	+	−		+	+		+	Authentic profiles
b	−	+	+	−	−		−	
c	+	−	+		−	−	−	
d	−	+	+	−				
e	−		−	−	−		−	
f	+	−	+	+	+		+	
g		−	+	+	−	−	+	
h	+	−	+	+	+		?	Target profile
i	+	−	+		−	−	−	Attack profiles
j	−	+	+	−			−	
k	−		−	−	−		−	
l	+	−	+	+	+		−	
m		−	+	+	−	−	−	

Fig. 25.2: Simplified system database showing authentic user profiles and a number of attack profiles inserted. In this example, user *h* is seeking a prediction for item 7, which is the subject of a product nuke attack.

25.2.1 An Example Attack

To illustrate the basic idea of a profile injection attack, consider the simplified recommender system database that is presented in Figure 25.2. In this example, the objective is to demote the recommendations that are made for item 7 (i.e. a product nuke attack), and a number of attack profiles (users *i* through *m*) have been inserted into the system to target this item.

In particular, consider the binary recommendation problem in which the task is to predict whether or not user *h* likes item 7. In the first instance, let the attack profiles be ignored and consider only the authentic profiles (users *a* through *g*) as possible neighbours for the target user, *h*. Regardless of the specific recommendation algorithm used, presumably the algorithm would determine that users *a* and *f* have similar tastes to the active user, and since both of these users like item 7, a positive recommendation for the item follows.

When the attack profiles are also considered as possible neighbours, the situation is significantly altered. Several of these attack profiles are also similar to user *h*, and, since all of these profiles rate item 7 poorly, the system is now likely to recommend a negative rating for the item. Thus, the objective of the attack is realised. The next section discusses how these attack profiles must be crafted to work well in a realistic setting.

25.3 Characterising Attacks

A profile-injection attack against a recommender system consists of a set of profiles added to the system by the attacker. A profile consists of a set of rating/item pairs, or alternately, we can think of the profile being a vector of all items, with a rating value for each item, but allowing the *null* value for unrated items. For the attacks that we are discussing, there will always be a target item i_t that the attacker is interested in promoting or demoting. There will generally also be a set of *filler items*, that are chosen randomly from those available. We will denote this set I_F. Some attack models also make use of a set of items that are selected out of the database. The small set usually has some association with the target item (or a targeted segment of users). For some attacks, this set is empty. This will be the set I_S. Finally, for completeness, the set $I_\emptyset$ contains those items not rated in the profile. Since the selected item set is usually small, the size of each profile (total number of ratings) is determined mostly by the size of the filler item set. Some of the experimental results report filler size as a proportion of the size of I (i.e., the set of all items).

25.3.1 Basic Attacks

Two basic attack models, introduced originally in [12], are the random and average attack models. Both of these attacks involve the generation of profiles using randomly assigned ratings to the filler items in the profile.

25.3.1.1 Random Attack

Random attack profiles consist of random ratings distributed around the overall mean assigned to the filler items and a prespecified rating assigned to the target item. In this attack model, the set of selected items is empty. The target item i_t is assigned the maximum rating (r_{max}) or the minimum rating (r_{min}) in the case of push or nuke attacks, respectively.

The knowledge required to mount such an attack is quite minimal, especially since the overall rating mean in many systems can be determined by an outsider empirically (or, indeed, may be available directly from the system). However, this attack is not particularly effective [12, 6].

25.3.1.2 Average Attack

A more powerful attack described in [12] uses the individual mean for each item rather than the global mean (except for the pushed item). In the average attack, each assigned rating for a filler item corresponds (either exactly or approximately) to the mean rating for that item, across the users in the database who have rated it.

As in the random attack, this attack can also be used as a nuke attack by using r_{min} instead of r_{max}. It should also be noted that the only difference between the average attack and the random attack is in the manner in which ratings are computed for the filler items in the profile.

The average attack might be considered to have considerable knowledge cost of order $|I_F|$ (the number of filler items in the attack profile) because the mean and standard deviation of these items must be known. Experiments, however, have shown that the average attack can be just as successful even when using a small filler item set. Thus the knowledge requirements for this attack can be substantially reduced, but at the cost of making all profiles contain the same items, possibly rendering them conspicuous [4].

25.3.2 Low-knowledge attacks

The average attack requires a relatively high degree of system-specific knowledge on the part of attackers. A reasonable defense against such attacks would be to make it very difficult for an attacker to accumulate the required distribution data. The next set of attack types are those for which the knowledge requirements are much lower.

25.3.2.1 Bandwagon Attack

The goal of the bandwagon attack is to associate the attacked item with a small number of frequently rated items. This attack takes advantage of the Zipf's distribution of popularity in consumer markets: a small number of items, bestseller books for example, will receive the lion's share of attention and also ratings. The attacker using this model will build attack profiles containing those items that have high visibility. Such profiles will have a good probability of being similar to a large number of users, since the high visibility items are those that many users have rated. It does not require any system-specific data, because it is usually not difficult to independently determine what the "blockbuster" items are in any product space.

The bandwagon attack uses selected items which are likely to have been rated by a large number of users in the database. These items are assigned the maximum rating value together with the target item i_t. The ratings for the filler items are determined randomly in a similar manner as in the random attack. The bandwagon attack therefore can be viewed as an extension of the random attack.

As we show in Section 25.4, the bandwagon attack is nearly as effective as the average attack against user-based collaborative filtering algorithms[2], but without the knowledge requirements of that attack. Thus it is more practical to mount. However, as in the case of the average attack, it falls short when used against an item-based algorithm [12].

---

[2] Refer to Chapter 4 for details on the user-based and item-based collaborative filtering algorithms.

25.3.2.2 Segment Attack

Mobasher et al. [19] introduced the segment attack and demonstrated its effectiveness against the item-based algorithm. The basic idea behind the segment attack is to push an item to a targeted group of users with known or easily predicted preferences. For example, the producer of a horror movie might want to get the movie recommended to viewers who have liked other horror movies. In fact, the producer might prefer not to have his movie recommender to viewer who do not enjoy the horror genre, since these users might complain and thereby reveal his attack.

To mount this attack, the attacker determines a set of segment items that are likely to be preferred by his intended target audience. Like the bandwagon attack, it is usually fairly easy to predict what the most popular items in a user segment would be. These items are assigned the maximum rating value together with the target item. To provide the maximum impact on the item-based algorithm, the minimum rating is given to the filler items, thus maximising the variations of item similarities.

25.3.3 Nuke Attack Models

All of the attack models described above can also be used for nuking a target item. For example, as noted earlier, in the case of the random and average attack models, this can be accomplished by associating rating r_{min} with the target item instead of r_{max} . However, the results presented in Section 25.4 suggest that attack models that are effective for pushing items are not necessarily as effective for nuke attacks. Thus, researchers have designed additional attack models designed particularly for nuking items.

25.3.3.1 Love/Hate Attack

The love/hate attack is a very simple attack, with no knowledge requirements. The attack consists of attack profiles in which the target item it is given the minimum rating value, r_{min}, while other ratings in the filler item set are the maximum rating value, r_{max}. This can be seen as a very low-knowledge version of the Popular Attack below. Surprisingly, this is one of the most effective nuke attacks against the user-based algorithm.

25.3.3.2 Reverse Bandwagon Attack

The reverse bandwagon attack is a variation of the bandwagon attack, discussed above, in which the selected items are those that tend to be rated poorly by many users. These items are assigned low ratings together with the target item. Thus the target item is associated with widely disliked items, increasing the probability that

the system will generate low predicted ratings for that item. This attack was designed to reduce the knowledge required by selecting only a handful of known disliked items. For example, in the movie domain, these may be box office flops that had been highly promoted prior to their openings.

In Section 25.4, we show that although this attack is not as effective as the more knowledge-intensive average attack for nuking items in the user-based system, it is a very effective nuke attack against item-based recommender systems.

25.3.4 Informed Attack Models

The low-knowledge attacks above work by approximating the average attack, concentrating on items that are expected to be rated because of their popularity. The average attack in turn is a natural choice for an attacker with a basic intuition about collaborative recommendation, namely that users will be compared on the basis of similarity, so the incentive is to make the profiles similar to the average user. If, on the other hand, the attacker has more detailed knowledge of the precise algorithm, a more powerful attack can be mounted.

25.3.4.1 Popular Attack

Let us assume that the recommender system uses the widely studied user-based algorithm proposed in [27], where similarities between users are calculated using Pearson correlation[3]. In a similar manner to the bandwagon attack, attack profiles are constructed using popular (i.e. frequently rated) items from the domain under attack.

A high degree of overlap does not, however, guarantee high similarities between attack and authentic profiles. The bandwagon attack used random filler items to generate variation among ratings with the aim of producing at least some profiles that correlate correctly with any given user. The Popular Attack makes use of average rating data and rates the filler items either $r_{min} + 1$ and r_{min}, according to whether the average rating for the item is higher or lower. Linking the rating value to the average rating is likely to result in positive correlations between attack and authentic profiles and furthermore also maximises the prediction shift (see Section 25.4) of attack profiles as computed by the algorithm under consideration (see [25] for details).[4]

---

[3] See [25] for a discussion on informed attacks in cases where alternative similarity metrics are employed. Note that none of the metrics considered provided robustness against attack.

[4] Note that an optimal push attack strategy is also presented in [18]. In this case, it is concluded that maximising the correlation between authentic and attack profiles is the primary objective. While this conclusion makes sense, it is important to select attack profile ratings that also maximise prediction shift, as is the case with the popular attack described here.

The ratings strategy described above applies to push attacks; this strategy can easily be adjusted for nuke attacks. For example, positive correlations but negative prediction shifts can be achieved by assigning the target item a rating of r_{min}, and ratings of r_{max} and $r_{max} - 1$ to the more- and less-liked selected items.

The knowledge requirement here is intermediate between the bandwagon attack and the average attack. Like the bandwagon attack, the popular items can usually be easily estimated from outside the system; but because there are no filler items, the Popular Attack will need more popular items. The attacker then needs to guess at the relative average preferences between these items in order to provide the correct rating. It might be possible to extract such distinctions from the system itself, or if not, to mine them from external sources; for example, by counting the number of positive and negative reviews for particular items to find general trends.

25.3.4.2 Probe Attack Strategy

Although there are no studies that look at the detectability of the popular attack, it seems likely that it would be easy to detect since all of the attack profiles are identical and also because in many rating databases $r_{min} + 1$ and r_{min} ratings are relatively rare.

A less conspicuous strategy is to obtain items and their ratings from the system itself via the Probe Attack. To perform this strategy, the attacker creates a seed profile and then uses it to generate recommendations from the system. These recommendations are generated by the neighboring users and so they are guaranteed to be rated by at least some of these users and the predicted ratings will be well-correlated with these users' opinions. One could imagine probing narrowly in order to influence a small group as in the segment attack, or probing more broadly to construct an average attack. In a sense, the probe attack provides a way for the attacker to incrementally learn about the system's rating distribution.

This strategy also has another advantage over the popular attack, since less domain knowledge is required by an attacker. Only a small number of seed items need to be selected by the attacker, thereafter the recommender system is used to identify additional items and ratings. In the experiments conducted in Section 25.4, seed items are selected and assigned ratings in a similar manner as in the popular attack.

25.4 Measuring Robustness

Collaborative recommendation algorithms can be categorised into two general classes, which are commonly referred to as memory-based and model-based algorithms [2]. Memory-based algorithms utilise all available data from a system database to compute predictions and recommendations. In contrast, model-based algorithms operate by first deriving a model from the system data, and this model is subsequently used in the recommendation process.

A wide range of collaborative recommendation algorithms have been proposed in the literature, and a comprehensive analysis of the robustness of all of these algorithms is beyond the scope of this chapter. Here, we focus on two widely implemented and studied algorithms, the *user-based* and *item-based* algorithms [27, 32]. The reader is referred to [21, 20, 31] for a robustness analysis of some other collaborative recommendation algorithms.

25.4.1 Evaluation Metrics

Since the objective of push and nuke attacks is to promote and demote target items, we need to evaluate how successfully they do so. Evaluation metrics for robustness need to capture the differences in the predicted ratings and recommended status (i.e. whether or not the target item in included in a top N recommended list) of target items pre- and post-attack.

Many researchers have used average prediction shift to evaluate the changes in predicted ratings. Let U_T and I_T be the sets of users and items, respectively, in the test data. 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' are the pre- and post-attack predictions, respectively. A positive value means, for example, that the attack has succeeded in making a pushed item more positively rated. The average prediction shift for an item i over all users can be computed as $\Delta_i = \sum_{u \in U_T} \Delta_{u,i}/|U_T|$. Similarly the average prediction shift for all items tested can be computed as $\bar{\Delta} = \sum_{i \in I_T} \Delta_i/|I_T|$.

Prediction shift is a good indicator that an attack is having the desired effect of making a pushed (or nuked) item appear more (or less) desirable. However, it is possible that a pushed item, for example, could be strongly shifted on average but still not make it onto a recommendation list. Such a situation could arise if the item's initial average prediction is so low that even a strong boost is insufficient. To capture the impact of an attack on prediction lists, another metric has been proposed: hit ratio. Let R_u be the set of top N recommendations for user u. If the target item appears in R_u, for user u, the scoring function H_{ui} has value 1; otherwise it is zero. Hit ratio for an item i is given by $HitRatio_i = \sum_{u \in U_T} H_{ui}/|U_T|$. Average hit ratio can then calculated as the sum of the hit ratio for each item i following an attack on i across all items divided by the number of items: $\overline{HitRatio} = \sum_{i \in I_T} HitRatio_i/|I_T|$.

Many experimenters make use of the publicly available MovieLens 100K dataset[5]. This dataset consists of 100,000 ratings made by 943 users on 1,682 movies. Ratings are expressed on an integer rating scale of 1 to 5 (the higher the score, the more liked an item is). Results below should be assumed to be relative to this dataset unless otherwise stated.

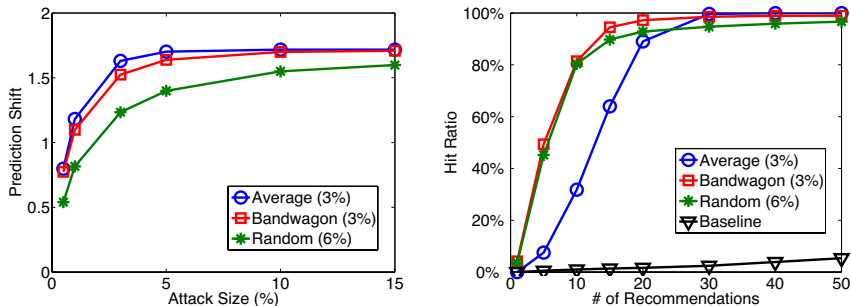[5] http://www.cs.umn.edu/research/GroupLens/data/.

Fig. 25.3: Prediction shift (left) and hit ratio (right) for product push attacks mounted against the user-based collaborative recommendation algorithm. Hit ratio results relate to a 10% attack size.

25.4.2 Push Attacks

To get a sense for the impact that a push attack can have, we will look at results originally reported in [20]. In these figures, the user-based algorithm is subjected to various attacks of different sizes (attack size is measured as a percentage of the total number of authentic profiles in the system; thus an attack of 1% equates to the insertion of 10 attack profiles into the MovieLens dataset). Figure 25.3 (left) shows the average attack (3% filler size), the bandwagon attack (using one frequently rated item and 3% filler size), and the random attack (6% filler size). These parameters were selected as they are the versions of each attack that were found to be most effective. Not surprisingly, the most knowledge-intensive average attack achieved the best performance in terms of prediction shift. This attack works very well. It is capable of moving an average-rated movie (3.6 is the mean) to the top of the five point scale. The performance of the bandwagon attack was quite comparable, despite having a minimal knowledge requirement. In addition, the bandwagon attack was clearly superior to the random attack, which highlights the significance of including the selected items that are likely to be rated by many users.

Interestingly, Figure 25.3 (right) shows that the largest hit ratios were achieved by the bandwagon attack, indicating that prediction shift does not necessarily translate directly into top N recommendation performance. This result is particularly encouraging from the attacker's perspective, given that the required knowledge to implement such attacks is low. Note that all attacks significantly outperform the pre-attack hit ratio results (indicated by "baseline" in the figure).

The item-based algorithm was shown in [12] to be relatively robust against the average attack. The segment attack, introduced in [19], was specifically crafted as a limited-knowledge attack for the item-based algorithm. It aims to increase the column-by-column similarity of the target item with the users preferred items. If the
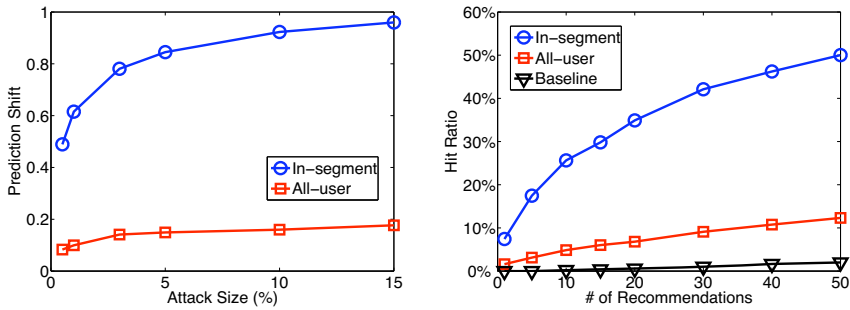
Fig. 25.4: Prediction shift (left) and hit ratio (right) for product push attacks mounted against the item-based collaborative recommendation algorithm. Hit ratio results relate to a 10% attack size.

target item is considered similar to something that the user likes, then its predicted rating will be high – the goal of the push attack. The task therefore for the attacker is to associate her product with popular items that are considered to be similar. The users who have a preference for these similar items are considered the target segment. The task for the attacker in crafting a segment attack is therefore to select items similar to the target item for use as the segment portion of the attack profile I_S. In the realm of movies, we might imagine selecting films of a similar genre or those containing the same actors.

In [19], user segments are constructed by looking at popular actors and gen-res. For the results shown in Figure 25.4, the segment is all users who gave above average ratings (4 or 5) to any three of the five selected horror movies, namely, Alien, Psycho, The Shining, Jaws, and The Birds. For this set of five movies, the researchers selected all combinations of three movies that had at least 50 users support, and chose 50 of those users randomly and averaged the results.

The power of the segmented attack is demonstrated in the figure, which contrasts the horror movie fans against the set of all users. While the segmented attack shows some impact against all users, it is clearly very successful in pushing the attacked movie precisely to those users defined by the segment. Further, in the context of the item-based algorithm, the performance of this attack compares very favourably to that of the high-knowledge average attack. For example, the average attack achieved a hit ratio of 30% against all users for top N lists of size 10 and an attack size of 10%. In contrast, the segmented attack achieved approximately the same hit ratio for the same size top N list, but using an attack size of only 1%.

It should also be noted that, although designed specifically as an attack against the item-based algorithm, the segment attack is also effective against the user-based algorithm. Due to limitations of space, we do not show these results here – refer to [20] for details.
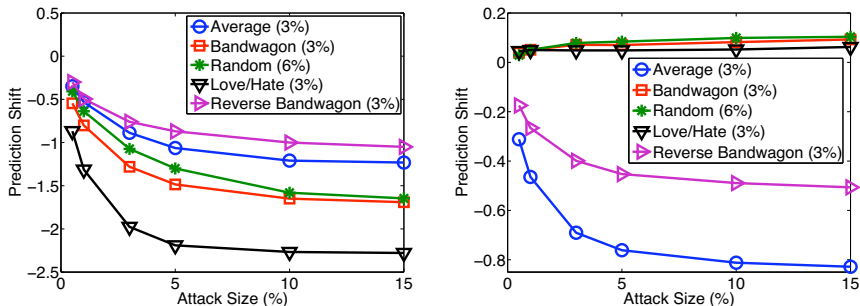
Fig. 25.5: Prediction shifts achieved by nuke attacks against the user-based (left) and item-based (right) algorithms.

25.4.3 Nuke Attacks

It might be assumed that nuke attacks would be symmetric to push attacks, with the only difference being the rating given to the target item and hence the direction of the impact on predicted ratings. However, our results show that there are some interesting differences in the effectiveness of models depending on whether they are being used to push or nuke an item. In particular, the rating distribution should be taken into account: there are in general relatively few low ratings in the MovieLens database, so low ratings can have a big impact on predictions. Furthermore, if we look at the top N recommendations, the baseline (the rate at which an average movie makes it into a recommendation list) is quite low, less than 0.1 even at a list size of 50. It does not take much to make an item unlikely to be recommended.

In the love/hate attack, the randomly selected 3% of filler items were assigned the maximum rating while the target item was given the minimum rating. For the reverse bandwagon attack (designed to attack the item-based algorithm), items with the lowest average ratings that meet a minimum threshold in terms of the number of user ratings in the system are selected as the selected item set, as described in detail in Section 25.3. The experiments were conducted using $|I_S| = 25$ with a minimum of 10 users rating each movie.

Results are shown in Figure 25.5 for all attack models. Despite the minimal knowledge required for the love/hate attack, this attack proved to be the most effective against the user-based algorithm. Among the other nuke attacks, the bandwagon attack actually surpassed the average attack, which was not the case with the push results discussed above.

The asymmetry between these results and the push attack data is somewhat surprising. For example, the love/hate attack produced a positive prediction shift slightly over 1.0 for a push attack of 10% against the user-based algorithm, which is much less effective than even the random attack. However, when used to nuke

an item against the user-based algorithm, this model was by far the most effective model we tried, with a prediction shift of almost twice that of the average attack. For pushing items, the average attack was the most successful, while it proved to be one of the least successful attacks for nuking items. The bandwagon attack, on the other hand, performed nearly as well as the average attack in pushing items, and had superior overall performance for nuking, despite its lower knowledge requirement.

Overall, the item-based algorithm proved to be far more robust. The average attack was the most successful nuke attack here, with reverse bandwagon close behind. The asymmetries between push and nuke continue as we examine the item-based results. The random and love/hate attacks were poor performers for push attacks, but as nuke attacks, they actually failed completely to produce the desired effect. Reverse bandwagon (but not bandwagon) proved to be a reasonable low-knowledge attack model for a nuke attack against the item-based algorithm.
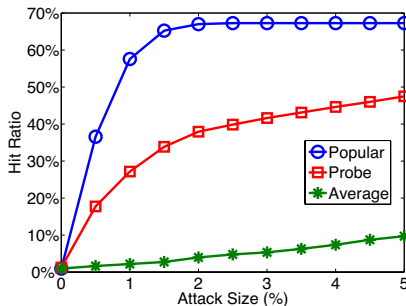
25.4.4 Informed Attacks

Finally, we turn to the evaluation of the informed attack strategies against the user-based algorithm. In particular, we compare the performance of the informed popular and probe push attacks to the average attack as seen above.

The attacks were implemented as follows. Popular attack profiles consisting of a total of 100 items (including the target item) were selected and assigned ratings as described in Section 25.3. For the probe attack, 10 seed items were selected at random from the 100 most frequently rated items from the system. Thereafter the system was interrogated to discover additional profile items and ratings. In total, probe attack profiles consisted of 100 items. Likewise, the benchmark average attack profiles consisted of 100 items, which corresponds to a filler size of approximately 1.7%. For the purposes of comparison, the 100 most frequently-rated items were chosen for average attack profiles (and not selected randomly, as before).

Figure 25.6 shows the hit ratios achieved by the three attacks. It is clear from the figure that the impact of the informed attacks was significantly greater than that of the average attack. For example, for an attack size of only 2%, the hit ratios achieved by the popular, probe and average attacks were 65%, 34% and 3%, respectively, for top N lists of size 10. Thus the advantage of creating attacks that consider particular features of the algorithm under attack is clearly demonstrated.

The main drawback associated with the informed attacks lies in the high degree of domain knowledge that is required in order to select the appropriate items and ratings with which to create the attack profiles. As discussed in Section 25.3, however, such knowledge is often made directly available to attackers by recommender system applications. Further, the knowledge required can often be obtained from other sources, e.g. by examining best seller lists and the number of positive and negative reviews received by items, etc. Even in situations where such data is only partially available, previous work demonstrates that these informed attacks retain their strong performance [26].

Fig. 25.6 Hit ratios achieved
by the popular, probe and
average push attacks against
the user-based algorithm.

25.4.5 Attack impact

It is clear from the research summarized above that the memory-based algorithms
that form the core of collaborative recommendation research and practice are highly
vulnerable to manipulation. An attacker with fairly limited knowledge can craft at-
tacks that will make any item appear well liked and promote it into many users'
recommendation lists. The "efficient" attacks that have been developed clearly are
a threat to the stability and usability of collaborative systems and thus we see the
justification for the low-scale / high-impact portion of the theoretical curve shown
in Figure 25.1.

    To respond to this threat, researchers have examined two complementary re-
sponses. The shaded "detection" areas in Figure 25.1 point towards the first re-
sponse, which is to detect the profiles that make up an attack and eliminate them.
The second approach is to design algorithms that are less susceptible to the types of
attacks that work well against the classic algorithms.

25.5 Attack Detection

Figure 25.7 summarises the steps involved in attack detection. This is a binary clas-
sification problem, with two possible outcomes for each profile, namely, *Authentic*,
meaning that the classifier has determined that the profile is that of a genuine system
user or *Attack*, meaning that the classifier has determined that this is an instance of
an attack profile. One approach to the detection problem, followed by work such as
[7, 1], has been to view it as a problem of determining independently for each profile
in the dataset, whether or not it is an attack profile. This is the 'single profile' input
shown in Figure 25.7. The input is a single rating vector $\mathbf{r}_u$, for some user u from
the dataset. Before processing by the classifier, a *feature extraction* step may extract
a set of features, $\mathbf{f}_u = (f_1, \ldots, f_k)$ from the raw rating vector $\mathbf{r}_u$. The classifier takes
$\mathbf{f}_u$ as input and outputs, "*Attack*" or "*Authentic*". If the classifier is a *supervised* clas-
sifier, then a training phase makes use of annotated dataset of profiles, i.e. a set of
profiles labelled as Authentic or Attack, in order to learn the classifier parameters.

Because most attack scenarios consist of groups of profiles working *in concert* to push or nuke a particular item, work such as [16, 23] has suggested that there is benefit to considering groups of profiles *together* when making the classification. This is represented by the 'Group of Profiles' input, in which the classifier considers an entire group of profiles, possibly after some feature extraction, and outputs a label for each profile in the group. Note that not all steps may take place in any particular scenario. For instance, there may be no feature extraction, in which case, $\mathbf{f} = \mathbf{r}$ and if *unsupervised* classifiers are used, then there is no need for a training phase.
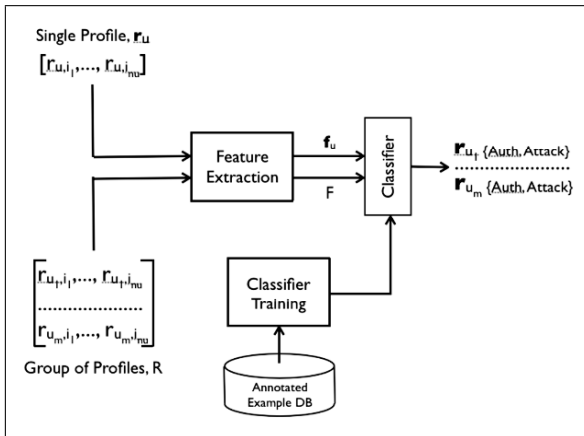

Fig. 25.7: The detection process.

25.5.1 Evaluation Metrics

To compare different detection algorithms, we are interested primarily in measures of classification performance. Taking a 'positive' classification to mean the labeling of a profile as Attack, a confusion matrix of the classified data contains four sets, two of which – the true positives and true negatives – consist of profiles that were correctly classified as Attack or Authentic, respectively; and two of which – the false positives and false negatives – consist of profiles that were incorrectly classified as Attack or Authentic, respectively. Various measures are used in the literature to compute performance based on the relative sizes of these sets. Unfortunately, different researchers have used different measures, making direct comparison of results sometimes difficult.

Precision and *recall* are commonly used performance measures in information retrieval. In this context, they measure the classifier's performance in identifying at-

tacks. Each measure counts the number of attack profiles correctly classified. Recall which is also called *sensitivity* presents this count as a fraction of the total number of actual attacks in the system. Precision, which is also called the *positive predictive value* (PPV), presents this count as a fraction of the total number of profiles labelled as Attack:

$$recall \equiv sensitivity = \frac{\text{\# true positives}}{\text{\# true positives} + \text{\# false negatives}}, \quad (25.1)$$

$$precision \equiv PPV = \frac{\text{\# true positives}}{\text{\# true positives} + \text{\# false positives}}.$$

Analogous measures can be given for performance in identifying authentic profiles. *Specificity* presents the count of authentic profiles correctly classified as a fraction of the total number of authentic profiles in the system. *Negative predictive value* (NPV), presents the count as a fraction of the total number of profiles labelled Authentic:

$$specificity = \frac{\text{\# true negatives}}{\text{\# true negatives} + \text{\# false positives}}, \quad (25.2)$$

$$NPV = \frac{\text{\# true negatives}}{\text{\# true negatives} + \text{\# false negatives}}.$$

In detection results below, we use the terms *precision*, *recall*, *specificity* and *NPV*.

25.5.1.1 Impact on Recommender and Attack Performance

The misclassification of authentic profiles results in the removal of good data from the ratings database, which has the potential to impact negatively on the overall performance of the recommender system. One way to assess this impact is to compute the MAE of the system before and after detection and filtering. On the positive side, the removal of attack profiles reduces attack performance. Assuming the attack is a push or nuke attack, the degree to which attack performance is affected can be assessed by computing the prediction shift on the targeted item before and after detection and filtering.

25.5.2 Single Profile Detection

The basis of individual profile detection is that the distribution of ratings in an attack profile is likely to be different to that of authentic users and therefore each attack profile can be distinguished by identification of these differences. As such, individual profile detection is an instance of a *statistical detection* problem. It should be noted that it is in the interest of the attacker to minimise the statistical differences between attack and authentic profiles, in order to minimise the probability of de-

tection. On the other hand, a cost-effective attack is likely to consist of unusually influential profiles – e.g., a targeted pushed item will have unusualy high ratings and filler items may have been chosen to support the influence of the profile towards high ratings for the target. As a result, distinctive characteristics are likely to exist and may be manifested in many ways, including an abnormal deviation from the system average rating, or an unusual number of ratings in a profile [1].

25.5.2.1 Unsupervised Detection

An unsupervised individual profile detection algorithm is described in [7]. Detection is based on certain common generic attributes of attack profiles, for example that there is a higher than usual rating deviation from mean in such profiles and that such profiles are likely to have a higher than usual similarity to their closest neighbours. Measures of these attributes are proposed and these are applied to compute a probability that a profile is an attack profile.

25.5.2.2 Supervised Detection

Supervised detection algorithms have focussed on the selection of attributes of attack profiles from which to build a feature vector for input to a classifier. Generally, such features have been selected by observation of *generic* attributes that are common across attack profiles of a number of different attack strategies and also *model specific* attributes that are common across profiles that have been generated for a specific type of attack.

In [5] profile attributes based to those proposed in [7] and others along similar lines were developed into features for inclusion in a feature vector input to a supervised classifer. Moreover, other features based on the statistics of the filler and target items in the user profile, rather than the entire profile, were proposed. For example, the *filler mean variance* feature is defined as the variance of the ratings in the filler partition of the profile and is used to detect average attacks; the *filler mean target difference* feature, defined as the difference between the means of the target items and the means of the filler items, is used to detect bandwagon attacks.

The authors looked at three supervised classifiers: kNN, C4.5, and SVM. The kNN classifier uses detection attributes of the profiles to find the $k = 9$ nearest neighbors in the training set using Pearson correlation for similarity to determine the class. The C4.5 and SVM classifiers are built in a similar manner such that they classify profiles based on the detection attributes only. The results for the detection of a 1% average attack over various filler sizes are reproduced in Figure 25.8. SVM and C4.5 have near perfect performance on identifying attack profiles correctly, but on the other hand, they also misclassify more authentic profiles than kNN. SVM has the best combination of recall and specificity across the entire range of filler sizes for a 1% attack.
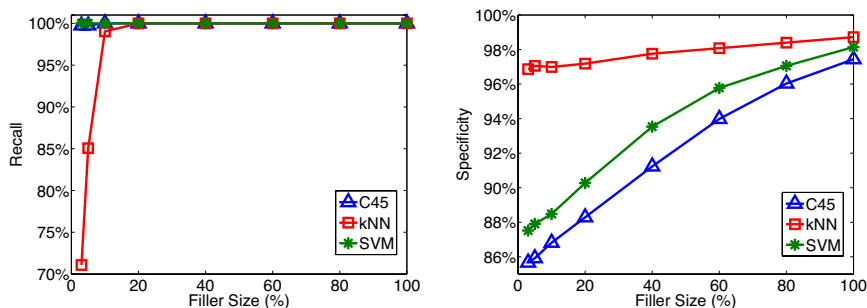
Fig. 25.8: Recall (left) and specificity (right) vs filler size for three classifiers trained on a 1% average attack.

The effect of misclassification of authentic profiles is assessed by examining the MAE of the system before and after detection and filtering. The increase in MAE is observed to be less than 0.05 on a rating scale of 1–5. Finally the effectiveness of the attack as measured by the prediction shift on the targeted item is shown to be significantly reduced when detection is used. All three classifiers reduce the range of attacks that are successful, particularly at low attack sizes. The SVM algorithm, in particular, dominates for attack sizes less than 10%, allowing no resulting prediction shift over that entire range.
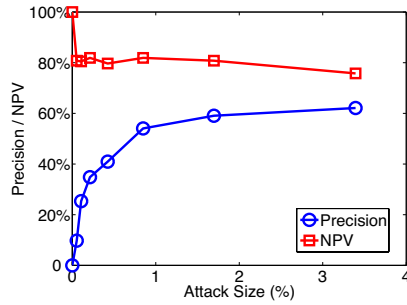
25.5.3 Group Profile Detection

A number of unsupervised algorithms that try to identify groups of attack profiles have been proposed [23, 33, 18]. Generally, these algorithms rely on clustering strategies that attempt to distinguish clusters of attack profiles from clusters of authentic profiles.

25.5.3.1 Neighbourhood Filtering

In [23] an unsupervised detection and filtering scheme is presented. Rather than filtering profiles from the dataset in a preprocessing step, in this method, filtering is applied *to the profiles in the active user's neighbourhood* during prediction for a particular item. This approach has the advantage of identifying just those attack profiles that are targeting the active item. The strategy is based on an algorithm proposed in [8] in the context of reputation reporting systems that aims to provide a reputation estimate for buyers and sellers engaged in on-line marketplaces that is robust to malicious agents who attempt to fradulently enhance their own reputations. The ap-

Fig. 25.9 Precision and NPV for the neighbourhood filtering algorithm vs attack size.

proach involves the clustering of neighbourhoods into two clusters. Analysing the statistics of the clusters, a decision is made as to whether an attack is present and, if so, which cluster contains the attack profiles. *All* profiles in the cluster are removed.

Clustering is performed using the Macnaughton-Smith [13] divisive clustering algorithm. The rating distributions for the active item over each of the clusters are then compared. Since the goal of an attacker is to force the predicted ratings of targeted items to a particular value, it is reasonable to expect that the ratings for targeted items that are contained in any attack profiles are centered on the attack value, which is likely to deviate significantly from the mean of the authentic neighbours' ratings. Thus an attack is deemed to have taken place if the difference in the means for the two clusters is sufficiently large. The cluster with the smaller standard deviation is determined to be the attack cluster.

Results for this algorithm (using *precision* and *NPV*) applied to an informed nuke attack on the Movielens dataset are reproduced in Figure 25.9. The fraction of authentic users contained in the cluster identified as the cluster of authentic users is at least 75% for all attack sizes tested, so attack profiles are being effectively filtered from the system. However, particularly for small attack sizes, a significant proportion of the attack cluster is made up of authentic users. The cost of removing malicious profiles is to also lose authentic profiles that may have contributed to the accuracy of the prediction. Results show that filtering a system that has not been attacked leads to an increase of around 10% in the MAE.

25.5.3.2 Detecting attacks using Profile Clustering

In [18] the observation is made that attacks consist of multiple profiles which are highly correlated with each other, as well as having high similarity with a large number of authentic profiles. This insight motivates the development of a clustering approach to attack detection, using Probabilistic Latent Semantic Analysis (PLSA) and Principal Component Analysis (PCA).

In the PLSA model [11], an unobserved factor variable $Z = \{z_1, \ldots z_k\}$ is associated with each observation. In the context of collaborative recommendation, an observation corresponds to a rating for some user-item pair and ratings are predicted

using

$$\Pr(u, i) = \sum_{i=1}^{k} \Pr(z_i) \Pr(u|z_i) \Pr(i|z_i).$$

The parameters of this expression are chosen to maximise the likelihood of the observed data, using the Expectation Maximisation algorithm. As discussed in [21], the parameters $\Pr(u|z_i)$ can also be used to produce a clustering of the users by assigning each user u to each cluster C_i such that $\Pr(u|z_i)$ exceeds a certain threshold μ or to the cluster that maximises $\Pr(u|z_i)$ if μ is never exceeded.

It is noted in [18] that all or most attack profiles tend to be assigned to a single cluster. Identifying the cluster containing the attack profiles provides an effective strategy for filtering them from the system. Using the intuition that clusters containing attack profiles will be 'tighter' in the sense that the profiles are very similar to each other, the average Mahalanobis distance over the profiles of each cluster is calculated and that with the minimum distance is selected for filtering. Experiments show that PLSA based attack detection works well against strong attacks. However, for weaker attacks the attack profiles tend to be distributed across different clusters.

A second strategy to exploit the high similarity between attack profiles proposed in [18] is to base a clustering on a PCA of the covariance matrix of the user profiles. Essentially this strategy attempts to identify a cluster where the sum of the pair-wise covariances between profiles in the cluster is maximised. PCA has been widely used as a dimension reduction strategy for high-dimensional data. Identifying profiles with dimensions, the method is explained intuitively in [18] as a method of identifying those highly-correlated dimensions (i.e. profiles) that would safely be removed by PCA. Alternatively, a cluster C can be defined by an indicator vector $\mathbf{y}$ such that $y(i) = 1$ if user $u_i \in C$ and $y(i) = 0$ otherwise. With S defined as the covariance matrix, the sum of the pair-wise covariances of all profiles in C, may be written as the quadratic form

$$\mathbf{y}^T S \mathbf{y} = \sum_{i \in C, j \in C} S(i, j).$$

Moreover, for the normalised eigenvectors $\mathbf{x}_i$ of S, associated with eigenvector λ_i such that $\lambda_1 \leq \cdots \leq \lambda_m$, the quadratic form evaluates as

$$\mathbf{y}^T S \mathbf{y} = \sum_{i=1}^{m} (\mathbf{y}.\mathbf{x_i})^2 (\mathbf{x}_i^T S \mathbf{x}_i) = \sum_{i=1}^{m} (\mathbf{y}.\mathbf{x_i})^2 \lambda_i.$$

With this observation, the method described in [18] may be understood as a method that seeks the binary vector $\mathbf{y}$ that maximises the quadratic form by choosing $\mathbf{y}$ so that it has small correlation with those $3 - 5$ eigenvectors corresponding to the smallest eigenvalues and hence correlates strongly with the eigenvectors corresponding to large eigenvalues.

Precision and recall results for the PLSA and PCA clustering strategies are reproduced in Figure 25.10 for an average attack of size 10%. Similar results have been obtained for random and bandwagon attacks. The PLSA and PCA clustering

strategies require that the size of the filtered cluster be specified and, in these results, the cluster size is taken to be the actual number of inserted attack profiles. This point should be taken into account in comparing the results with those obtained with the neighbourhood filtering strategy (Figure 25.9), in which no such control on the cluster size was applied. The 80% maximum recall obtained for the PLSA strategy is due to the fact that the wrong cluster is selected approximately 20% of the time. The PCA clustering strategy shows very good performance, even in the case of attacks consisting of a mixture of random, average and bandwagon profiles.

The UnRAP algorithm [3] also uses clustering to distinguish attack profiles. This algorithm uses a measure called the H_v score which has proved successful in identifying highly correlated biclusters in gene expression data. In the context of attack detection, the H_v score measures for each user, a sum of the squared deviations of its ratings from the user mean, item mean and overall mean ratings:

$$H_v(u) = \frac{\sum_{i \in I}(r_{u,i} - \bar{r}_i - \bar{r}_u + \bar{r})^2}{\sum_{i \in I}(r_{u,i} - \bar{r}_u)^2},$$

where $\bar{r}_i$ is the mean over all users of the ratings for item i, $\bar{r}_u$ is the mean over all items of the ratings for user u and $\bar{r}$ is the mean over users and items.

A H_v score is assigned to all users in the database and users are sorted according to this score. The top $r = 10$ users with highest score are identified as potential attackers and are examined to identify a target item. The target is identified as that which deviates most from the mean user rating. Next, a sliding window of r users is passed along the sorted user list, shifting the window by one user each iteration. The sum of the rating deviation for the target item is calculated over the window and a stopping point is reached when this sum reaches zero. The users traversed during this process become candidate attack profiles, which are then further filtered by removing any that have not rated the item or whose rating deviation is in the opposite direction to the attack. Precision results for this method on an average attack are reproduced in Figure 25.10, compared with the PCA clustering strategy. In general, the authors report that this method performs well particularly for mid-size attacks, in which other methods show a dip in performance.

25.5.4 Detection findings

For both supervised and unsupervised detection, it has proved possible to achieve reasonably good performance against the attack types discussed in 25.3. Perhaps this is not so surprising, since the assumption is that these attacks are crafted according to a fairly regular pattern and thereby vary substantially from the real users of the system. The extent to which real-life attacks against recommender systems correspond to these idealized models is not known, since e-commerce companies
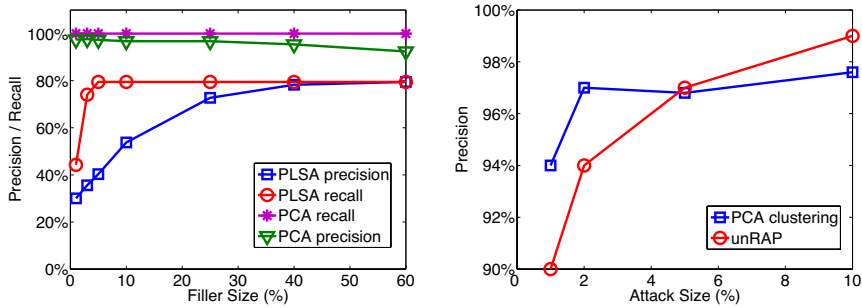
Fig. 25.10: Precision and recall for the PLSA and PCA clustering strategies vs filer size for a 10% average attack (left). Precision vs attack size for PCA clustering and UnRAP on an average attack, with filler size=10% (right).

have been reluctant to reveal vulnerabilities that they have identified in their own systems.

Going back to the framework in Figure 25.1, these findings give us some optimism that the shaded area at the upper left exists. That is, it is possible to detect attacks that are crafted to be optimal against the well-known memory-based algorithms. It remains an open question to what extent these detection measures extend downward and to the right, into regions where attacks differ from the optimal and have correspondingly less impact, but still remain a source of profit for the attacker.

25.6 Robust Algorithms

An alternative (or perhaps a complement) to filtering and detection is to develop recommendation algorithms that are intrinsically robust to attack. To date, researchers have largely tried to identify algorithms robust against the attacks that work well on the memory-based algorithms. An open question is whether new attacks can be tailored to exploit vulnerabilities in algorithms that have shown high robustness to standard attacks.

25.6.1 Model-based Recomendation

It has been shown in [21] that model-based recommendation algorithms provide a greater degree of robustness to attack strategies that have proven highly effective on memory-based algorithms. Moreover, this robustness does not come at a significant cost in terms of recommendation accuracy. This work has been followed up in [17,

15], which surveys model-based attack resistant algorithms and proposes a robust matrix factorisation strategy.

A model-based recommendation strategy based on clustering user profiles is analysed in [21]. In this strategy, similar users are clustered into segments and the similarity between the target user and a user segment is calculated. For each segment, an aggregate profile, consisting of the average rating for each item in the segment is computed and predictions are made using the aggregate profile rather than individual profiles. To make a recommendation for a target user u and target item i, a neighbourhood of user segments that have a rating for i and whose aggregate profile is most similar to u is chosen. A prediction for item i is made using the k nearest segments and associated aggregate profiles, rather than the k nearest neighbours. Both k-means clustering and PLSA-based clustering, as described in Section 25.5.3.2, are evaluated. The prediction shift achieved by an average attack on these algorithms, compared with the standard kNN algorithm, is shown in Figure 25.11 (left). The model-based algorithms are considerably more robust and not significantly less accurate, since, according to [21], PLSA and k-means clustering achieve an MAE of 0.75 and 0.76 using 30 segments, in comparison to a value of 0.74 for kNN.

25.6.2 Robust Matrix Factorisation (RMF)

One model-based approach to collaborative recommendation which has proven very successful recently, is the application of matrix factorisation approaches based on singular value decomposition (SVD) and its variants. Recent work in [15, 18] has suggested a robust factorisation strategy in which the clustering strategy of Section 25.5.3.2 is used in conjunction with the training phase of the factorisation procedure. For example, the PLSA clustering strategy can be applied in conjunction with the PLSA recommendation algorithm. In [15], it is proposed that after elimination of attack clusters, the $\Pr(z_i|u)$ distribution of the remaining clusters should be renormalised and the last few steps of training should be re-run, to maintain the predictive accuracy of the standard PLSA algorithm and significantly reduce prediction shift.

Another strategy proposed in [18] is in the context of the application of *Generalized Hebbian Learning* algorithm to compute a rank-1 SVD factorisation:

$$R \approx GH,$$

where R is the rating matrix and G and H are matrices of rank 1. Again, the algorithm is modified so that the contribution of the suspicious users towards the prediction model is zero, once suspicious users have been identified. Results from this strategy are reproduced in Figure 25.11 (right). The MAE for the attacked algorithm is shown when the number of suspicious users r is set to the exact number of attack profiles inserted, and when it is given a fixed value of 7% of the user base. Also shown for
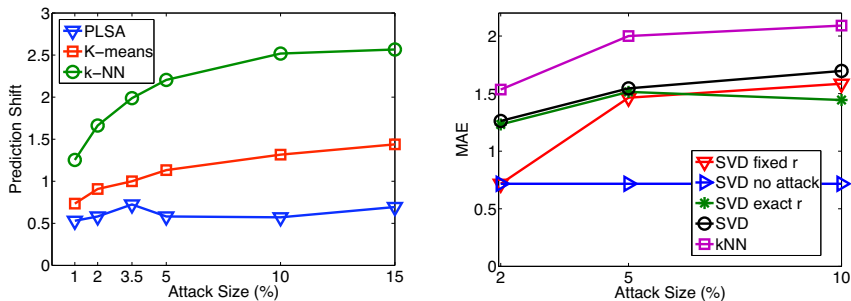
Fig. 25.11: Prediction shift vs attack size for an average attack at 5% filler for segment recommendation (left). MAE on the attacked item vs attack size for filler size of 10% using RMF (right).

reference is the MAE on the kNN algorithm and standard SVD, with and without attack.

Theoretical results are also emerging to support the robustness of particular classes of model-based algorithm. In [35], a manipulation-resistant class of collaborative filtering algorithm is proposed for which robustness is proved, in the sense that the effect of any attack on the ratings provided to an end-user diminishes with increasing number of products rated by the end-user. Here, effectiveness is measured in terms of a measure of the average distortion introduced by the attack to the ratings provided to the user. The class of algorithms for which the proof holds is referred to as a *linear* probabilistic collaborative filtering. In essence, the system is modelled as outputting a probability mass function (PMF) over the possible ratings and in linear algorithms, the PMF of the attacked system can be written as a weighted sum of the PMF obtained considering only genuine profiles and that obtained considering only attack profiles. Robustness is obtained, because, as the user supplies more ratings, the contribution of the genuine PMF to the overall PMF begins to dominate. The authors show that, while nearest neighbour algorithms are not linear in this sense, some well-known model-based algorithms such as the naive-bayes algorithm are asymptotically linear.

25.6.3 Other Robust Recommendation Algorithms

Attack profiles are ineffective if they do not appear in the neighborhoods of authentic users. By avoiding similarity as a criterion for neighbour selection, the recommendation algorithm can be made robust to attacks where the attack profiles are designed to have high similarity with authentic users. In [23] it is argued that the goal of neighbour selection is to select the most *useful* neighbours on which to base

the prediction. While similarity is one measure of usefulness, the notion of neighbour utility can be extended to include other performance measures. A selection criterion is proposed based on a notion of *inverse popularity*. It is shown that, with this selection strategy, the same overall system performance in terms of MAE is maintained. Moreover, cost-effective attacks that depend on popular items to build highly influential profiles are rendered much less effective.

In [31], a robust algorithm is presented based on association rule mining. Considering each user profile as a transaction, it is possible to use the *Apriori* algorithm to generate association rules for groups of commonly liked items. The support of an item set $X \subset I$ is the fraction of user profiles that contain this item set. An association rule is an expression of the form $X \Rightarrow Y(\sigma_r, \alpha_r)$, where σ_r is the support of $X \cup Y$ and α_r is the *confidence* for the rule, defined as $\sigma(X \cup Y)/\sigma(X)$. The algorithm finds a recommendation for a user u by searching for the highest confidence association rules, such that $X \subseteq P_u$ is a subset of the user profile and Y contains some item i that is unrated by u. If there is not enough support for a particular item, that item will never appear in any frequent item set and will never be recommended. This algorithm proves robust to the average attack. For attack sizes below 15%, only 0.1% of users are recommended an attacked item by the association rule algorithm, compared to $80 - 100\%$ of users for the *kNN* algorithm. The trade-off is that coverage of the association rule algorithm is reduced in comparison to kNN. However, the algorithm is not robust against the segment attack.

25.6.4 The Influence Limiter and Trust-based Recommendation

In [28, 29] a recommendation algorithm is presented for which robustness bounds can be calculated. The algorithm introduces two key additional features to the recommendation process, an *influence limiter* and a reputation system. The idea behind the algorithm is to weight the contribution of each user towards a prediction by using a global measure of reputation. The reputation value is boosted when a profile correctly estimates a rating for a neighbor and is reduced which it fails to do so. Within this recommendation model, the authors prove a non-manipulation result that shows that any attack strategy involving up to n attack users, the negative impact due to the attacker is bounded by a small amount. They also show that a user seeking to maximize influence has a strict incentive to rate honestly. Other properties of this algorithm, such as its accuracy, are still under study.

The influence limiter is just one algorithm that takes into account *trust* and *reputation* (see Chapter 20) in order to build recommendations. In recent years, there has been increasing focus on incorporating trust models into recommender systems [14, 22, 9]. In [14], trust propagation is used to increase the coverage of recommender systems while preserving accuracy. In [22] it is argued that the reliability of a profile to deliver accurate recommendations in the past should be taken into account by recommendation algorithms. An algorithm that uses trust as a means of filtering profiles prior to recommendation so that only the top k most trustworthy

profiles participate in the prediction process is presented in [9]. The trust associated with a user for making predictions for an item is computed based on the users' accuracy on predicting their own ratings for that item. The robustness achieved by such algorithms is a function of how difficult it would be for an attacker to become trusted.

25.7 Conclusion

Collaborative recommender systems are meant to be adaptive – users add their preferences to these system and their output changes accordingly. Robustness in this context must mean something different than the classical computer science sense of being able to continue functioning in the face of abnormalities or errors. Our goal is to have systems that adapt, but that do not present an attractive target to the attacker. An attacker wishing to bias the output of a robust recommender system would have to make his attack sufficiently subtle that it does not trigger the suspicion of an attack detector, sufficiently small that it does not stand out from the normal pattern of new user enrollment, and sufficiently close to real user distribution patterns that it is not susceptible to being separated out by dimensionality reduction. If this proves a difficult target to hit and if the payoff for attacks can be sufficiently limited, the attacker may not find the impact of his attack sufficiently large relative to the effort required to produce it. This is the best one can hope for in an adversarial arena.

It is difficult to say how close we have come to this ideal. If an attacker is aware that such detection strategies are being applied, then the attack can be modified to avoid detection. For example, [23] shows that if the attacker is aware of the criteria used to decide if an attack profiles exist in the user's neighbourhood, then the attacker can construct profiles which, although somewhat less effective than the standard attacks, can circumvent detection. In [34] the effectiveness of various types of attack profile obfuscation are evaluated. The general finding is that obfuscated attacks are not much less effective than optimal ones and much harder to detect. More research is needed in this area.

Similar issues apply in the context of attack resistant recommendation algorithms. While model-based algorithms show robustness to attacks that are effective on memory-based algorithms, it is possible to conceive of new attacks that target model-based algorithms. [31], for example, shows that association rule based recommendation is vulnerable to segment attacks.

Another way to view the problem is as a game between system designer and attacker. For each system that the designer creates, an optimal attack against it can be formulated by the attacker, which then requires another response from the designer, etc. What we would like to see is that there are diminishing returns for the attacker, so that each iteration of defense makes attacking more expensive and less effective. One benefit of a detection strategy is that a system with detection cannot be more vulnerable to attack than the original system, since in the worst case, the attacks are not detected. We do not yet know if the robust algorithms that have been proposed

such as RMF have some as-yet-undiscovered flaw that could make them vulnerable to a sophisticated attack, perhaps even more vulnerable than the algorithms that they replace.

Acknowledgements

References

1. A.Williams, C., Mobasher, B., Burke, R.: Defending recommender systems: detection of profile injection attacks. Service Oriented Computing and Applications pp. 157–170 (2007)
2. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence pp. 43–52 (1998)
3. Bryan, K., O'Mahony, M., Cunningham, P.: Unsupervised retrieval of attack profiles in collaborative recommender systems. In: RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems, pp. 155–162. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1454008.1454034
4. Burke, R., Mobasher, B., Bhaumik, R.: Limited knowledge shilling attacks in collaborative filtering systems. In Proceedings of Workshop on Intelligent Techniques for Web Personalization (ITWP'05) (2005)
5. Burke, R., Mobasher, B., Williams, C.: Classification features for attack detection in collaborative recommender systems. In: Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining, pp. 17–20 (2006)
6. Burke, R., Mobasher, B., Zabicki, R., Bhaumik, R.: Identifying attack models for secure recommendation. In: Beyond Personalization: A Workshop on the Next Generation of Recommender Systems (2005)
7. Chirita, P.A., Nejdl, W., Zamfir, C.: Preventing shilling attacks in online recommender systems. In Proceedings of the ACM Workshop on Web Information and Data Management (WIDM'2005) pp. 67–74 (2005)
8. Dellarocas, C.: Immunizing on–line reputation reporting systems against unfair ratings and discriminatory behavior. In Proceedings of the 2nd ACM Conference on Electronic Commerce (EC'00) pp. 150–157 (2000)
9. Fug-uo, Z., Sheng-hua, X.: Analysis of trust-based e-commerce recommender systems under recommendation attacks. In: ISDPE '07: Proceedings of the The First International Symposium on Data, Privacy, and E-Commerce, pp. 385–390. IEEE Computer Society, Washington, DC, USA (2007). DOI http://dx.doi.org/10.1109/ISDPE.2007.55
10. Herlocker, J., Konstan, J., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In Proceedings of the 22nd International ACM SIGIR Conference on Research and Development in Information Retrieval pp. 230–237 (1999)
11. Hofmann, T.: Collaborative filtering via gaussian probabilistic latent semantic analysis. In: SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research

and development in informaion retrieval, pp. 259–266. ACM, New York, NY, USA (2003). DOI http://doi.acm.org/10.1145/860435.860483

12. Lam, S.K., Riedl, J.: Shilling recommender systems for fun and profit. In Proceedings of the 13th International World Wide Web Conference pp. 393–402 (2004)

13. Macnaughton-Smith, P., Williams, W.T., Dale, M., Mockett, L.: Dissimilarity analysis – a new technique of hierarchical sub–division. Nature **202**, 1034–1035 (1964)

14. Massa, P., Avesani, P.: Trust-aware recommender systems. In: RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems, pp. 17–24. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1297231.1297235

15. Mehta, B., Hofmann, T.: A survey of attack-resistant collaborative filtering algorithms. Bulletin of the Technical Committee on Data Engineering **31**(2), 14–22 (2008). URL http://sites.computer.org/debull/A08June/mehta.pdf

16. Mehta, B., Hofmann, T., Fankhauser, P.: Lies and propaganda: Detecting spam users in collaborative filtering. In: Proceedings of the 12th international conference on Intelligent user interfaces, pp. 14–21 (2007)

17. Mehta, B., Hofmann, T., Nejdl, W.: Robust collaborative filtering. In: RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems, pp. 49–56. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1297231.1297240

18. Mehta, B., Nejdl, W.: Unsupervised strategies for shilling detection and robust collaborative filtering. User Modeling and User-Adapted Interaction **19**(1-2), 65–97 (2009). DOI http://dx.doi.org/10.1007/s11257-008-9050-4

19. Mobasher, B., Burke, R., Bhaumik, R., Williams, C.: Effective attack models for shilling item-based collaborative filtering system. In Proceedings of the 2005 WebKDD Workshop (KDD'2005) (2005)

20. Mobasher, B., Burke, R., Bhaumik, R., Williams, C.: Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. ACM Transactions on Internet Technology **7**(4) (2007)

21. Mobasher, B., Burke, R.D., Sandvig, J.J.: Model-based collaborative filtering as a defense against profile injection attacks. In: AAAI. AAAI Press (2006)

22. O'Donovan, J., Smyth, B.: Is trust robust?: an analysis of trust-based recommendation. In: IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces, pp. 101–108. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1111449.1111476

23. O'Mahony, M.P., Hurley, N.J., Silvestre, C.C.M.: An evaluation of neighbourhood formation on the performance of collaborative filtering. Artificial Intelligence Review **21**(1), 215–228 (2004)

24. O'Mahony, M.P., Hurley, N.J., Silvestre, G.C.M.: Promoting recommendations: An attack on collaborative filtering. In: A. Hameurlain, R. Cicchetti, R. Traunmüller (eds.) DEXA, *Lecture Notes in Computer Science*, vol. 2453, pp. 494–503. Springer (2002)

25. O'Mahony, M.P., Hurley, N.J., Silvestre, G.C.M.: An evaluation of the performance of collaborative filtering. In Proceedings of the 14th Irish International Conference on Artificial Intelligence and Cognitive Science (AICS'03) pp. 164–168 (2003)

26. O'Mahony, M.P., Hurley, N.J., Silvestre, G.C.M.: Recommender systems: Attack types and strategies. In Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05) pp. 334–339 (2005)

27. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., J.Riedl: Grouplens: An open architecture for collaborative filtering of netnews. In Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'94) pp. 175–186 (1994)

28. Resnick, P., Sami, R.: The influence limiter: provably manipulation-resistant recommender systems. In: RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems, pp. 25–32. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1297231.1297236

29. Resnick, P., Sami, R.: The information cost of manipulation-resistance in recommender systems. In: RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems,

pp. 147–154. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1454008.1454033

30. Rokach, L.: Mining manufacturing data using genetic algorithm-based feature set decomposition, Int. J. Intelligent Systems Technologies and Applications, 4(1):57-78 (2008).

31. Sandvig, J.J., Mobasher, B., Burke, R.: Robustness of collaborative recommendation based on association rule mining. In: RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems, pp. 105–112. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1297231.1297249

32. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item–based collaborative filtering recommendation algorithms. In Proceedings of the Tenth International World Wide Web Conference pp. 285–295 (2001)

33. Su, X.F., Zeng, H.J., Chen, Z.: Finding group shilling in recommendation system. In: WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web, pp. 960–961. ACM, New York, NY, USA (2005). DOI http://doi.acm.org/10.1145/1062745.1062818

34. Williams, C., Mobasher, B., Burke, R., Bhaumik, R., Sandvig, J.: Detection of obfuscated attacks in collaborative recommender systems. In Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06) (2006)

35. Yan, X., Roy, B.V.: Manipulation-resistnat collaborative filtering systems. In: RecSys '09: Proceedings of the 2009 ACM conference on Recommender systems. ACM, New York, NY, USA (2009)