Chapter 12

Attack-Resistant Recommender Systems

"*The truth is incontrovertible. Malice may attack it, ignorance may deride it, but in the end, there it is.*" – Winston Churchill

12.1 Introduction

The input to recommender systems is typically provided through open platforms. Almost anyone can register and submit a review at sites such as Amazon.com and Epinions.com. Like any other data-mining system, the effectiveness of a recommender system depends almost exclusively on the quality of the data available to it. Unfortunately, there are significant motivations for participants to submit incorrect feedback about items for personal gain or for malicious reasons:

- The manufacturer of an item or the author of a book might submit fake (positive) reviews on Amazon in order to maximize sales. Such attacks are also referred to as *product push* attacks.
- The competitor of an item manufacturer might submit malicious reviews about the item. Such attacks are also referred to as *nuke* attacks.

It is also possible for an attack to be designed purely to cause mischief and disrupt the underlying system, although such attacks are rare relative to attacks motivated by personal gain. This chapter studies only attacks that are *motivated to achieve a particular outcome* in the recommendation process. The person making the attack on the recommender system is also referred to as the *adversary*.

By creating a concerted set of fake feedbacks from many different users, it is possible to change the predictions of the recommender system. Such users become *shills* in the attack process. Therefore, such attacks are also referred to as *shilling attacks*. It is noteworthy that the addition of a single fake user or rating is unlikely to achieve the desired outcome. In most

C.C. Aggarwal, *Recommender Systems: The Textbook*,
DOI 10.1007/978-3-319-29659-3_12

cases, an adversary would need to create a large number of fake users (or fake *profiles*) to achieve the desired outcome. For the purpose of this chapter, a profile refers to a set of ratings corresponding to a fake user created by the adversary. Of course, the number of injected profiles may depend on the specific recommendation algorithm being attacked, and the specific approach used to attack it. An attack that requires a smaller number of injected profiles is referred to as an *efficient* attack because such attacks are often difficult to detect. On the other hand, if an attack requires a large number of injected profiles, then such an attack is *inefficient* because most systems should be able to detect a sudden injection of a large number of ratings about a small number of items. Furthermore, the effectiveness of the attacks may depend on the specific recommendation algorithm being used. Some recommendation algorithms are more robust to attacks than others. Furthermore, different attacks may work more or less efficiently with different algorithms.

Attacks can also be classified based on the amount of knowledge required to mount them successfully. Some attacks require only limited knowledge about the ratings distribution. Such attacks are referred to as *low-knowledge* attacks. On the other hand, attacks that require a large amount of knowledge about the ratings distribution are referred to as *high-knowledge attacks*. As a general rule, a trade-off exists between the amount of knowledge required to make an attack and the efficiency of the attack. If adversaries have more knowledge about the ratings distribution, then they can generally make more efficient attacks.

This chapter is organized as follows. In the next section, we will discuss the nature of the trade-offs between the required knowledge and the efficiency of the attack. We will also discuss the impact of using a specific recommendation algorithm on the effectiveness of the attack. The various types of attacks are discussed in section 12.3. The problem of attack detection in recommender systems is discussed in section 12.4. The design of robust recommender systems is discussed in section 12.5. A summary is given in section 12.6.

12.2 Understanding the Trade-Offs in Attack Models

Attack models have a number of natural trade-offs between the efficiency of the attack and the amount of knowledge required to mount a successful attack. Furthermore, the effectiveness of a particular attack may depend on the specific recommendation algorithm being used. In order to understand this point, we will use a specific example.

Consider the toy example illustrated in Table 12.1 where we have 5 items and 6 (real) users. The ratings are all drawn from the range of 1 to 7, where 1 indicates extreme dislike and 7 indicates extreme like. Furthermore, an attacker has injected 5 fake profiles, which are denoted with the labels *Fake-1*, *Fake-2*, *Fake-3*, *Fake-4*, and *Fake-5*. The goal of this attacker is to inflate the ratings of item 3. Therefore, this attacker has chosen a rather naive attack, in which they have inserted fake profiles containing a single item corresponding to the item 3. However, such an attack is not particularly efficient. It is highly detectable because only a single item is included in every injected profile with a very similar rating. Furthermore, such ratings injections are unlikely to have a large impact on most recommendation algorithms. For example, consider the *non-personalized* recommendation algorithm in which the highest rated item is recommended. In such a case, the *naive* attack algorithm will increase the predicted rating of item 3, and it will be more likely to be recommended. The attack might also increase the predicted rating of item 3 in cases where item bias is explicitly used as a part of the model construction. There is, however, little chance that such an attack will significantly affect a neighborhood-based algorithm. Consider, for example, a user-based neighborhood algorithm in which the profiles are used to make predictions for Mary. None of the injected profiles will be close to the rating profile of Mary; therefore, Mary's predicted

Table 12.1: A naive attack: injecting fake user profiles with a single pushed item

Item ⇒ / **User ⇓**	1	2	3	4	5
John	1	2	1	6	7
Sayani	2	1	2	7	6
Mary	1	1	?	7	7
Alice	7	6	5	1	2
Bob	?	7	6	2	1
Carol	7	7	6	?	3
Fake-1	?	?	7	?	?
Fake-2	?	?	6	?	?
Fake-3	?	?	7	?	?
Fake-4	?	?	6	?	?
Fake-5	?	?	7	?	?

Table 12.2: Slightly better than naive attack: injecting fake user profiles with a single pushed item and random ratings on other items

Item ⇒ / **User ⇓**	1	2	3	4	5
John	1	2	1	6	7
Sayani	2	1	2	7	6
Mary	1	1	?	7	7
Alice	7	6	5	1	2
Bob	?	7	6	2	1
Carol	7	7	6	?	3
Fake-1	2	4	7	6	1
Fake-2	7	2	6	1	5
Fake-3	2	1	7	6	7
Fake-4	1	7	6	2	4
Fake-5	3	5	7	7	4

ratings of item 3 will not be affected by the injection of the fake profiles. This particular injection of ratings is, therefore, not particularly efficient because it is hard to affect the predicted ratings, even when a large number of fake users are injected. Furthermore, such a profile injection can be detected in most cases because of the injection of ratings involving a single item.

Consider a second example of an attack, shown in Table 12.2, in which the attacker is trying to promote item 3, but he or she also adds random ratings to other items in order to reduce the detectability. Note that the genuine ratings in this second example are the same as the first, but the fake profiles are different. Such an attack is more effective than the one shown in Table 12.1. Consider the case where a user-based neighborhood is used to perform the recommendation. When only genuine profiles are used, John and Sayani are among Mary's neighbors and item 3 will have a low predicted rating for Mary. When fake profiles are also injected before user-based recommendation, most of the fake profiles are not close to Mary because the ratings are randomly chosen. However, the profile *Fake-3* happens

Table 12.3: Highly knowledgeable attacker injects fake user profiles

Item ⇒ **User** ⇓	1	2	3	4	5
John	1	2	1	6	7
Sayani	2	1	2	7	6
Mary	1	1	?	7	7
Alice	7	6	5	1	2
Bob	?	7	6	2	1
Carol	7	7	6	?	3
Fake-1	6	7	7	2	1
Fake-2	7	7	6	1	1
Fake-3	1	1	7	6	7
Fake-4	1	1	6	7	6
Fake-5	2	1	7	7	7

to be close to Mary by chance. As a result, the predicted rating of Mary for item 3 will increase. Therefore, from the point of view of an adversary, this type of attack does a better job than a completely naive attack with a single item. Nevertheless, this attack is quite inefficient because a large number of injected profiles would be required to affect the results of a neighborhood-based algorithm. In general, it is hard to ensure that randomly injected ratings will be close enough to a particular target user to whom the recommendation is to be made. After all, it is important for fake profiles to be close to target users in order to significantly affect the recommendation in any way.

In order to understand the impact of greater knowledge on the attack process, consider the example of an attacker who has a significant knowledge of the underlying ratings distributions. Therefore, we have illustrated an example in Table 12.3 in which the genuine ratings are the same as those in Table 12.1. However, the injected ratings are designed to reflect the underlying item correlations and also push the ratings for item 3 higher. For example, the attacker is aware that the ratings of items 1 and 2 are positively correlated in the ratings database, and the ratings of items 4 and 5 are also positively correlated. Furthermore, these two groups of items are negatively correlated with one another. Therefore, the attacker injects the ratings, while respecting these correlations. Correspondingly, it is evident that these correlations are respected in the fake profiles of Table 12.3. In this case, it is evident that the predicted ratings of Mary for item 3 are affected more significantly, than in the earlier examples of Tables 12.1 and 12.2. This is because the three profiles *Fake-3*, *Fake-4*, and *Fake-5* are all very close to Mary, and they may be included among her peers in a neighborhood-based algorithm. Therefore, this attack is very efficient because it requires a small number of profiles in order to cause a significant shift in the underlying ratings. On the other hand, such an attack requires a significant amount of knowledge, which may not always be available in practical settings.

It is noteworthy that the efficiency of a particular attack also depends on the particular algorithm being attacked. For example, user-based and item-based neighborhood algorithms have very different levels of propensity of being attacked. If an item-based algorithm were to be applied to the high-knowledge case of Table 12.3, then the predicted ratings of Mary for item 3 would not be affected very significantly. This is because the item-based algorithm uses the ratings of other users only in the item-item similarity computation. The fake

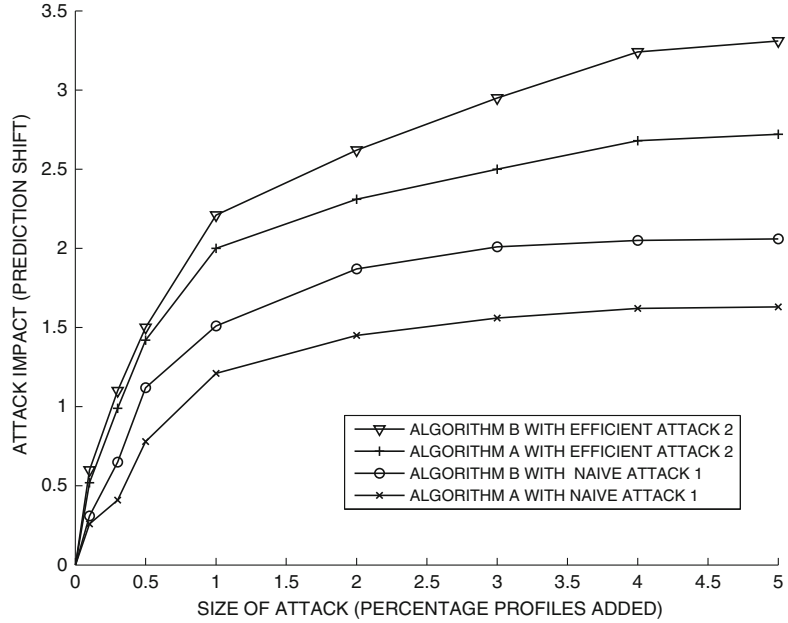

Figure 12.1: Typical examples of the effect of the combination of the specific push attack algorithm and the specific recommendation algorithm

profiles affect the similarity computation used to discover the most similar items to item 3; subsequently Mary's *own ratings* on these items are used to make the prediction. In order to change the most similar items to item 3, one must typically inject a large number of ratings, which makes the attack more detectable. Furthermore, it is much harder to change the predictions in a particular direction for the target item by changing its similar items; after all, Mary's *own ratings* on these items are used to make the prediction rather than those in the fake profiles. Algorithms that are less prone to attacks are referred to as *robust* algorithms. It is one of the goals of recommender systems to design algorithms that are more robust to attacks.

The aforementioned examples lead us to the following observations:

1. Carefully designed attacks are able to affect the predictions with a small number of fake profile insertions. On the other hand, a carelessly injected attack may have no effect on the predicted ratings at all.

2. When more knowledge about the statistics of the ratings database is available, an attacker is able to make more efficient attacks. However, it is often difficult to obtain a significant amount of knowledge about the ratings database.

3. The effectiveness of an attack algorithm depends on the specific algorithm being attacked.

In order to understand the nature of these trade-offs, consider a recommendation algorithm A that is robust to attacks, and another algorithm B that is not robust. Similarly, consider a naive attack (labeled 1), and an efficient attack (labeled 2), both of which are push attacks. Therefore, there are four different combinations of algorithm and attack-type. In Figure 12.1,

we have shown some typical examples of how a specific recommender system might respond to a particular attack type. The X-axis illustrates the fraction of fake profiles that are inserted in the attack, and the Y-axis illustrates the amount of shift in the predicted ratings. In each case, the shift in predicted ratings is positive because it is a push attack. Intuitively, the shift is defined as the amount by which the predicted ratings have moved on the average over all users. The shift may be computed over a particular (pushed) item, or it may be computed over a subset of (pushed) items. More details of the methodology for shift computation are provided in section 12.2.1.

The higher the curve is, the more efficient the attack will be. Efficient attacks are more desirable for an adversary because they are more difficult to detect. It is evident that the combination of recommendation algorithm B and attack type 2 leads to the highest curve, both because of the weakness of the recommendation algorithm and the efficiency of the attack. It is also possible to measure the impact of an attack in terms of other evaluation metrics such as the hit-ratio rather than through the prediction shift. In all cases, the impact of adding the fake profiles to a specific evaluation metric is quantified.

However, it is sometimes not possible to easily extrapolate the effectiveness of a particular attack into a concrete statement about the robustness of the recommendation algorithm at hand. This is because adversaries might tailor the attack to a specific recommendation algorithm, and therefore the robustness of the recommendation algorithm depends on the type of attack. For example, an attack algorithm that works well for a user-based neighborhood algorithm might not work well for an item-based neighborhood technique, and vice versa. By tailoring the attack to the specific recommendation algorithm at hand, more efficient attacks can be constructed. Fortunately, it is often difficult for an adversary to achieve this goal, unless she is aware of the specific recommendation algorithm being used.

An adversarial relationship perpetually exists between recommender systems and attackers. Attackers try to design increasingly clever algorithms to influence the recommender systems, whereas the designers of recommender systems try to propose more robust algorithms. Although the goal of this chapter is to learn how to design robust algorithms, it is important to understand attack strategies in order to be able to design robust algorithms. Therefore, we will first introduce the various types of attacks before discussing the design of robust algorithms.

12.2.1 Quantifying Attack Impact

In order to analyze the impact of various types of attacks, it is important to be able to quantify their impact. For example, the respective impacts of the attacks shown in Figures 12.1, 12.2, and 12.3 are quantified with a measure abstractly referred to as the "prediction shift." This measure is shown on the Y-axis in Figure 12.1. It is useful to examine in greater detail how the prediction shift is actually computed.

Consider a ratings matrix R with user set U and item set I. The first step is to select a subset $U_T \subseteq U$ of test users. Furthermore, let $I_T \subseteq I$ be the set of test items pushed in the testing process. Then, the attack is performed one at a time for each item $j \in I_T$, and the effect on predicted rating of users in U_T on item j is measured. The average prediction shift over all users and items is measured. Therefore, the attack needs to be performed $|I_T|$ times in order to measure the prediction shit over all test items.

Let $\hat{r}_{ij}$ be the predicted rating of user $i \in U_T$ for item $j \in T_T$ before the attack, and $\hat{r}'_{ij}$ be the corresponding predicted rating after the attack on item j. Then, the prediction shift of user i for item j is given by $\delta_{ij} = \hat{r}'_{ij} - \hat{r}_{ij}$. Note that δ_{ij} can be either positive or negative. A positive value indicates that the push attack has been successful, and therefore

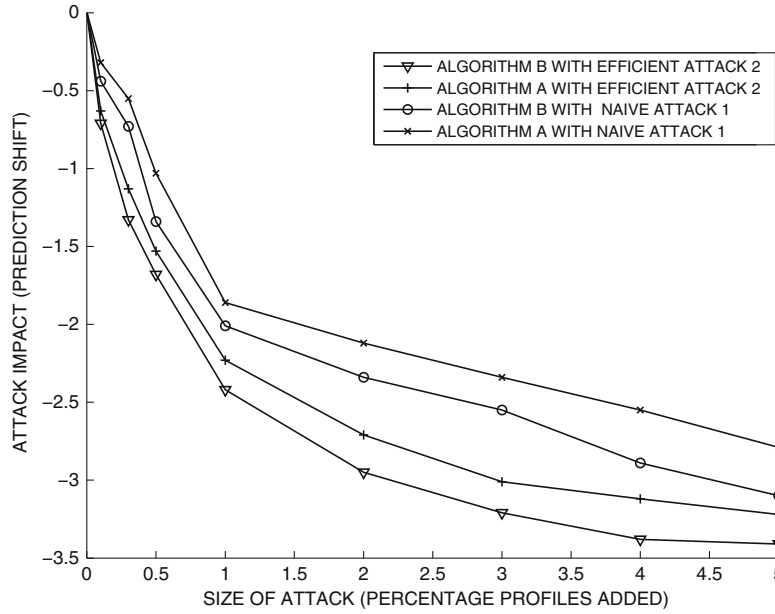

Figure 12.2: Typical examples of prediction shifts in case of nuke attack (Compare with push attacks of Figure 12.1)

the item j is more positively rated. If the attack is a nuke attack, then negative values of the prediction shift are indicative of success. Then, the average shift $\Delta_j(U_T)$ for test-user set U_T and item j is computed as follows:

$$\Delta_j(U_T) = \frac{\sum_{i \in U_T} \delta_{ij}}{|U_T|} \tag{12.1}$$

Then, the overall prediction shift $\Delta^{all}(U_T, I_T)$ over all items in I_T is equal to the average value of the per-item shift over all test items:

$$\Delta^{all}(U_T, I_T) = \frac{\sum_{j \in I_T} \Delta_j(U_T)}{|I_T|} \tag{12.2}$$

The prediction shift is a way of quantifying how well a pushed (or nuked) item has been shifted in a direction favoring its goal. Note that δ_{ij} can be either positive or negative; therefore, shifts in a direction opposite to the desired outcome are penalized by this measure. Furthermore, the prediction shift curves will be upward sloping in the case of a push attack, whereas they will be downward sloping in the case of a nuke attack. For example, typical curves for prediction shifts in the case of nuke attacks are illustrated in Figure 12.2. It is evident that these plots have trends opposite to those shown in Figure 12.1.

Although the prediction shift is a good way of quantifying the changes in the ratings, it may often not measure the true impact from the perspective of the end user. The end user only cares about whether her pushed item made it to the top-k list (or was removed from the top-k list). In many cases, a large prediction shift may not be sufficient to move an item into the top-k list. Therefore, a more appropriate measure is the *Hit-Ratio* $h_j(U_T)$, which is defined for item j and test user set U_T. The hit-ratio $h_j(U_T)$ is defined as the fraction of

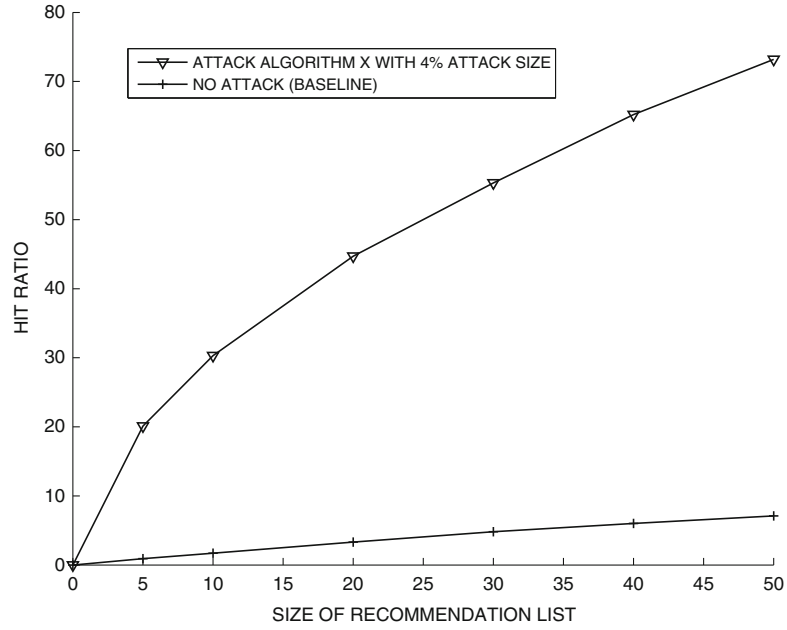

Figure 12.3: Effect of push attack on hit-ratio

users in U_T for which item j appears among the top-k recommendations. Then, the overall hit-ratio $h^{all}(U_T, I_I)$ over all test users and items is averaged over all the test items in I_T:

$$h^{all}(U_T, I_T) = \frac{\sum_{j \in I_T} h_j(U_T)}{|I_T|} \tag{12.3}$$

It is noteworthy that the hit-ratio is not a differential measure, as it does not compute the shift in the ratings. Therefore, unlike the prediction shift, one needs to plot the hit ratio both before and after the attack. In this type of plot, the X-axis depicts the size of the recommendation list, and the Y-axis depicts the hit-ratio; the size of the attack (i.e., the number of injected profiles) is fixed. An example of such a plot is illustrated in Figure 12.3 in which the hit-ratio of both the original algorithm and attacked algorithm is shown. The distance between these two curves provides an idea of the level of success of an adversary in making the pushed items appear on the recommendation list. It is also possible to fix the size of the recommendation list and plot the hit-ratio with the size of the attack. Such a plot would be somewhat similar to Figure 12.1 because it provides an idea of how the hit-ratios are affected with increasing attack size.

12.3 Types of Attacks

Although the rating of a particular item may be targeted in an attack, it is important to inject ratings of other items in order to make the attack effective. If fake profiles of only a single (pushed or nuked) item are inserted, they generally do not significantly affect the outcome of many recommendation algorithms. Furthermore, such attacks are generally easy to detect using automated methods. Therefore, the ratings of additional items are included in the injected profile. Such items are referred to as *filler* items. The importance of including

filler items is particularly emphasized by the example of Table 12.1, where the addition of only a single item containing a rating is not sufficient to create an effective attack.

The ratings of most of the items will not be specified in the fake user profiles, just as in the case of genuine user profiles. Such unspecified items are also referred to as *null* items. It is also evident from the example of Table 12.3 that the attacks are most effective when the filler items are correlated with the target item in terms of the underlying rating patterns. For example, if a target movie, such as *Gladiator,* is rated frequently with another movie, such as *Nero,* then it is generally beneficial to add filler ratings of *Nero* when trying either to use either a push attack or a nuke attack on *Gladiator*. It would not be quite as beneficial to add filler ratings for a completely unrelated item like *Shrek*. However, such attacks require a greater knowledge of the ratings distribution because correlated sets of items need to be identified. Therefore, there is a natural trade-off between the efficiency and knowledge requirements of different types of attacks.

Some attacks are specifically designed to be push attacks or to be nuke attacks. Although many attacks can be used in both capacities, each attack is generally more effective in one of the two settings. There are also subtle differences in the evaluation of these two types of attacks. The two types of attacks often show very different behavior in terms of the prediction shift and the hit-ratio. For example, it is much easier to nuke an item with a few bad ratings, given that only a few top items are recommended in any given setting. In other words, the effects on the hit-ratio can be more drastic than the effects on the prediction shift in the case of a nuke attack. Therefore, it is important to use multiple measures while evaluating push attacks and nuke attacks.

In the following, we will discuss the various types of attacks that are commonly used, and also discuss their common use case as either push attacks or as nuke attacks. These attacks require various levels of knowledge from the adversary. We will study these different attacks, starting with the ones that require the least knowledge.

12.3.1 Random Attack

In the random attack, the filler items are assigned their ratings from a probability distribution that is distributed around the *global* mean of all ratings across all items. Because the global mean is used, the ratings of the various filler items are drawn from the same probability distribution. The filler items are chosen randomly from the database, and therefore the selection of items to rate is also not dependent on the target item. However, in some cases, the same set of filler items may be used for each profile. There is no advantage in choosing the same set of filler items for each profile because it does not reduce the required level of knowledge to mount the attack, but it only makes the attack more conspicuous.

The target item is either set to the maximum possible rating value r_{max} or the minimum possible rating value r_{min}, depending on whether it is a push attack or a nuke attack. The main knowledge required to mount this attack is the mean values of all ratings. It is not very difficult to determine the global mean of the ratings in most settings. The limited knowledge required for a random attack comes at a disadvantage for the attacker, because such attacks are often not very efficient.

12.3.2 Average Attack

The average attack is similar to the random attack in terms of how the filler items are selected for rating. The same set of filler items are selected for each profile. However, the average attack differs from the random attack in terms of how the ratings are *assigned* to

the selected items. In the average attack, the ratings that are assigned to the filler items have values that are specified at or approximately at the average of the *specific* item. The target item is assigned either the maximum rating or the minimum rating depending on whether the attack is a push attack or a nuke attack. Note that the average attack requires a greater amount of knowledge than the random attack, because knowing the global mean is not sufficient. One also needs to know the mean of *each* filler item. Furthermore, the attack is somewhat conspicuous because the same set of filler items is used for each fake profile.

In order to reduce the possibility of detection, one can also use randomly selected filler items for each injected user profile. The drawback of doing this is that a greater amount of knowledge will be required for making the attack. For example, the global mean of *each* of the injected filler items will be required. However, this can sometimes be reasonable in settings where the ratings are public. For example, the ratings on Amazon.com are public, and the average values can be computed easily. In other systems, such as IMDb, the average rating of each item is often directly advertised. Alternatively, one can randomly select items out of a small set of candidate items in order to determine the fillers for each fake profile. Such a strategy requires much less knowledge. Furthermore, it has been shown [123] that it does not lose a significant amount of knowledge.

12.3.3 Bandwagon Attack

The main problem with many of the aforementioned attacks is the inherent sparsity of the ratings matrix, which prevents the injected profiles from being sufficiently similar to the existing profiles. When too many items are selected as filler items, the attack becomes conspicuous. On the other hand, when a small number of filler items are selected randomly for a fake profile, then it might not have an inadequate number of observed ratings in common with other users. In user-based collaborative filtering, a fake profile has no impact when it does not have any rated items in common with the target user to whom the recommendation is being made. The efficiency of the attack reduces as a result.

The basic idea of the bandwagon attack is to leverage the fact that a small number of items are very popular in terms of the number of ratings they receive. For example, a blockbuster movie or a widely used textbook might receive many ratings. Therefore, if these items are always rated in the fake user profile, it increases the chance of a fake user profile being similar to the target user. In such cases, the predicted ratings of the target user are more likely to be affected by the attack. Therefore, the knowledge about the popularity of the items is used to improve the efficiency of the attack. In addition to the popular items, a set of random items is used as additional filler items.

In the bandwagon attack, the ratings of popular items are set to their maximum possible rating value r_{max}. The other filler items are rated randomly. The reason for assigning the maximum rating value to the most popular items is to the increase the chances that more users will be found within the fake profiles, which are close to any particular target user to whom the recommendation is being made. This is because the popular items are more likely to be assigned positive ratings in real settings. The target item is set to the maximum possible rating r_{max} or the minimum possible rating r_{min}, depending on whether it is a push attack or a nuke attack.

It is noteworthy that the notion of "popular" items in this particular case does not necessarily refer to the most frequently rated items, but rather refers to widely liked items. Such items are likely to be frequently rated *in a positive way* in the ratings database. One does not need to use the ratings matrix in order to determine the most popular items. It is usually easy to determine the most popular products of any type from sources independent

of the ratings matrix. This is the main reason that the bandwagon attack requires much less knowledge as compared to the average attack. Bandwagon attacks can often perform almost as well as the average attack, in spite of their smaller knowledge requirements. In general, bandwagon attacks can influence user-based collaborative filtering algorithms significantly, but they have greater difficulty in influencing item-based algorithms.

12.3.4 Popular Attack

The popular attack shares a number of similarities with the bandwagon attack in that it also uses popular items in order to create the filler items. However, the popular items might be either widely liked or widely disliked items, but they must have many ratings. The popular attack also assumes more knowledge about the ratings database to set the ratings of these popular items. Furthermore, it does not assume the existence of an additional set of filler items. Therefore, more popular items have to be used in this attack than in the case of the bandwagon attack.

In order to set the ratings on the popular items in an intelligent way, more knowledge needs to be assumed about the underlying rating database. In particular, it is assumed that average values of the ratings of the popular items are known. In order to achieve a push attack, the ratings of the various filler items in a fake user profile are set as follows:

1. If the average rating of a filler item in the ratings matrix is less than the global rating average over all items, then the rating of that item is set to its minimum possible value r_{min}.

2. If the rating of a filler item is greater than the overall average rating of all items, then the rating of the item is set to $r_{min} + 1$.

3. The rating of the target item is always set to r_{max} in the fake user profile.

The reason for setting the ratings in this unusual way is (a) to increase the likelihood of finding a profile similar to that of the target user within the fake profiles by choosing differential ratings of r_{min} and $r_{min} + 1$ for filler items; and (b) to increase the ratings gap between the target item and the filler items to push the item more effectively. This attack can also be used for the case of a nuke attack with minor modifications. In a nuke attack, the ratings of filler items are set to $r_{max} - 1$ for low-rated popular items, r_{max} for highly rated popular items, and r_{min} for the target item.

As in the case of the bandwagon attack, one does not need to assume that the popular items need to be inferred from the ratings database. Such information can be easily inferred from other data sources. However, one does need to know the average values of the ratings, albeit only for the popular items. It is also possible to use external sources to *estimate* the popular items with lower or higher ratings. For example, one might use the text of the reviews to determine the items with positive or negative sentiment. Nevertheless, the knowledge requirements of a popular attack are always greater than those of a bandwagon attack.

12.3.5 Love/Hate Attack

The love/hate attack is specifically designed to be a nuke attack, and its main advantage is that it requires very little knowledge to mount this attack. In the love/hate attack, the nuked item is set to the minimum rating value r_{min}, whereas the other items are set to the maximum rating value r_{max}. In spite of the minimal knowledge requirements, this

attack is very effective. As discussed earlier, nuke attacks are generally easier to mount than push attacks. Therefore, such low-knowledge attacks often have a better chance of being successful in the case of nuke attacks as compared to push attacks. For example, a symmetrically designed attack in which the ratings of the filler items are set to r_{min} and the rating of the target item is set to r_{max}, is not quite as successful for pushing items. The love/hate attack is highly specific to user-based collaborative filtering algorithms, and it is almost completely ineffective with item-based collaborative filtering algorithms.

12.3.6 Reverse Bandwagon Attack

This attack is specifically designed to nuke items. The reverse bandwagon attack is a variation of the bandwagon attack, in which widely disliked items are used as filler items to mount the attack. The fact that such items are "widely disliked" means that they have received many ratings. For example, if a movie is highly promoted before its release but then turns out to be a box-office failure, then it will receive many low ratings. These items are selected as filler items. Such filler items are assigned low ratings together with the nuked item. As in the case of the bandwagon attack, it is often not very difficult to discover such items from other channels. This attack works very well as a nuke attack when an item-based collaborative filtering algorithm is used for recommendation. Although it can also be used in the case of user-based collaborative filtering algorithms, many other attack methods, such as the average attack, are generally more effective.

12.3.7 Probe Attack

An important aspect of many of the aforementioned methods is that the ratings are often artificially set to values, such as r_{min} and $r_{min}+1$, in an identical way across many profiles. The use of such ratings tends to make the attack rather conspicuous, and therefore easily detectable. The probe attack tries to obtain more realistic ratings for items *directly from a user-based recommender system* in order to use these values in the attack. In other words, the operation of a recommender system is probed to mount the attack.

In the probe attack, a seed profile is created by the attacker, and the predictions generated by the recommender system are used to learn related items and their ratings. Since these recommendations have been generated by user-neighbors of this seed profile, they are highly likely to be correlated with the seed profile. One can also use this approach to learn the ratings of items within a specific genre. For example, in a movie recommendation scenario, consider the case where the target item to be pushed or nuked corresponds to action movies. The seed profile might contain the ratings of a set of popular action movies. The seed profile can then be extended further by observing the operation of a user-based collaborative filtering algorithm when the seed profile is used as the target user. The recommended items and their predicted ratings can be used to augment the seed profile in a realistic way. The rating of the target item is set to r_{max} or r_{min}, depending on whether it is pushed or nuked, respectively. The ratings of other filler items learned from the probing approach are set to the average values predicted by the recommender system.

12.3.8 Segment Attack

Almost all the aforementioned attack methods work effectively with user-based collaborative filtering algorithms, but they do not work quite as effectively with item-based algorithms. The only exception is the reverse bandwagon attack, which is designed only to nuke items,

but not to push items. It is generally harder to attack item-based collaborative filtering algorithms. One of the reasons for this phenomenon is that an item-based algorithm leverages the target user's *own* ratings in order to make attacks. The target user is always an authentic user. Obviously, one cannot use fake profile injection to manipulate a genuine user's specified ratings.

However, it is possible to change the *peer* items with the use of fake profiles. Changing the peer items has an effect on the quality of the predicted ratings. In the segment attack, the attacker uses their domain knowledge to identify a targeted set of users (i.e., users with specific interests) to which they push the item. For example, the attacker might decide to push a historical movie, such as *Gladiator*, to users who have liked historical movies in the past. Note that the relevant genre for a particular movie is often common knowledge, and it does not require any specific information from the ratings matrix. Therefore, the first step for the attacker is to determine which *segment* (i.e., category or genre) of items are the closest to a given item. Such items are assigned the maximum possible rating together with the pushed item. An additional set of sampled filler items are assigned the minimum rating. This maximizes the variations in the item similarities towards items of the same genre. The basic idea is for the attacker to make sure that only very similar items are used in the item-based recommendation process. It is generally assumed that increasing the likelihood of using items of a similar genre in the prediction process for the target item will give it an additional advantage over other items in the recommendation process. After all, users tend to rate similar items in a similar way. Therefore, for users who have liked movies of this genre in the past, the predicted ratings of the targeted item will be pushed up more than other items of the same genre because of greater relevance. Therefore, it is more likely that such users will be recommended the targeted item. Although one might also use a variation of the segment attack for nuke attacks, it is mostly effective for push attacks. Furthermore, the segment attack may also be used effectively in the context of user-based collaborative filtering algorithms.

The segment attack is a generalization of the notion of the *favorite item* attack [123]. The favorite item attack is designed only with a specific *user* in mind. Filler items are selected to be a set of items, such that their ratings are greater than the average user rating. In such a case, the ratings of these items and the pushed item are set to their maximum value, and the ratings of filler items are set to the minimum value. Although the favorite item attack works well for both user-based and item-based collaborative filtering algorithms, the attack is restricted to a specific user. Furthermore, the attack requires a significant amount of knowledge of the values of the ratings. These characteristics tend to make this attack rather impractical. Its main utility is in establishing an upper bound on the effectiveness of other attacks.

12.3.9 Effect of Base Recommendation Algorithm

As discussed earlier, the choice of attack is highly specific to the particular recommendation algorithm at hand. In general, user-based recommendation algorithms are more susceptible to attacks as compared to item-based algorithms. Only a few attacks, such as the reverse bandwagon attack and the segment attack are specifically designed for item-based algorithms. Most of the other attack methods are effective for user-based algorithms, but are able to affect the item-based algorithms only to a limited degree. Some attack methods such as the love/hate attack are completely ineffective against item-based algorithms.

Interestingly, much of the work on attack algorithms is largely focussed on neighborhood-based methods and there are only a few studies on the effectiveness with respect to model-based algorithms. Some recent work [446, 522] has analyzed the vulnerability of model-based algorithms against attacks. Examples of the analyzed algorithms included clustering-based algorithms, PCA-based methods, LSA-based methods, and association rule methods. The experiments showed that model-based algorithms are generally more robust to attacks as compared to user-based collaborative filtering algorithms, although there were some variations among different algorithms. Hybridizing the algorithm tends to make the approach more robust, especially when external domain knowledge is used. This is because domain knowledge cannot be influenced by profile injection mechanisms. A summary of the effects of various attacks on various model-based collaborative filtering algorithms may be found in [523].

Although this chapter is primarily focussed on explicit ratings, a few attack methods have also been designed for implicit feedback data sets [79]. Just as explicit data sets require the injection of fake *profiles*, implicit feedback data sets require the injection of fake *actions*. The basic idea is to correlate the fake actions with other popular actions so as to give the impression that the fake actions are similar to these popular actions. Consider a Web site that wants to increase the likelihood of recommendation a particular page by injecting fake actions in the click-stream. The mechanism for injecting a fake action is to use an automated crawler that simulates Web browsing sessions. The crawler visits carefully selected Web pages in combination, so that the target item is pushed effectively. An example of such an attack is the *popular page* attack [79] in which the target page is crawled together with other popular pages. Such an attack can be viewed as an implicit version of the bandwagon attack.

12.4 Detecting Attacks on Recommender Systems

An adversarial relationship exists between attackers and the designers of recommender systems. From the point of view of maintaining a robust recommender system, the best way to thwart attacks is to detect them. Detection allows corrective measures (such as the removal of fake user profiles) to be taken. Accordingly, the detection of fake user profiles is a pivotal element in the design of a robust recommender system. However, the removal of fake profiles is a mistake-prone process, in which genuine profiles might be removed. It is important not to make too many mistakes, because the removal of authentic profiles can be counter-productive. On the other hand, the inability to remove fake profiles is also undesirable. This results in a natural trade-off between the precision and recall of fake profile removal. Correspondingly, attack detection algorithms are often measured in terms of this precision and recall. In fact, one can also use[1] the receiver operating characteristic (ROC) curve (see Chapter 7), which plots the trade-off between the true positive rate (TPR) and the false positive rate (FPR). An alternative way of evaluating the effectiveness of attack

[1]The ROC curve is used in a different context here than in Chapter 7. In Chapter 7, the ROC curve measures the effectiveness of ranking items for recommendations. Here, we measure the effectiveness of ranking user profiles based on their likelihood of being fake. However, the general principle of using the ROC curve is similar in both cases, because a ranking is compared with the binary ground-truth in both cases.

removal is by measuring the impact of profile removal on recommender system accuracy. For example, one can measure the mean absolute error both before and after the filtering of the profiles. The various detection algorithms can be compared in terms of this measure.

Almost all attacks use multiple profiles in order to undermine the recommender system. Therefore, the profiles may be removed either individually or as a group. Different attack algorithms have been designed for each case. Furthermore, attack detection algorithms may be either supervised or unsupervised. The difference between these two types of detection algorithms is as follows:

1. *Unsupervised attack detection algorithms:* In this case, ad hoc rules are used to detect fake profiles. For example, if a profile (or significant portion of it) is identical to many other profiles, then it is likely that all these profiles have been injected for the purpose of creating an attack. The basic idea in this class of algorithms is to identify the key characteristics of attack profiles that are not similar to genuine profiles. Such characteristics can be used to design unsupervised heuristics for fake profile detection.

2. *Supervised attack detection algorithms:* Supervised attack detection algorithms use classification models to detect attacks. Individual user profiles or groups of user profiles are characterized as multidimensional feature vectors. In many cases, these multidimensional feature vectors are derived using the same characteristics that are leveraged for the unsupervised case. For example, the number of profiles to which a given user profile is identical can be used as a feature for that user profile. Multiple features can be extracted corresponding to various characteristics of different types of attacks. A binary classifier can then be trained in which known attack profiles are labeled as $+1$, and the remaining profiles are labeled as -1. The trained classifier is used to predict the likelihood that a given profile is genuine.

Supervised attack detection algorithms are generally more effective than unsupervised methods because of their ability to learn from the underlying data. On the other hand, it is often difficult to obtain examples of attack profiles.

Attack detection methods are either *individual* profile detection methods or *group* profile detection methods. When detecting individual attack profiles, each user profile is assessed independently to determine whether or not it might be an attack. In the case of group detection, a set of profiles is assessed as a group. Note that both the unsupervised and supervised methods can be applied to either individual or group profile detection. In the following, we will discuss various methods for detecting attack profiles as individuals, and for detecting attack profiles as groups.

12.4.1 Individual Attack Profile Detection

Individual attack-profile detection is also referred to as *single attack-profile detection*. An unsupervised method for individual attack-profile detection is discussed in [158]. In this technique, a set of features is extracted from each user profile. The features are such that unusually high or unusually low values are indicative of an attack, depending on the feature at hand. In many cases, these features measure the *consistency* of a particular profile with other profiles in the system. Therefore, the fraction of features that take on abnormal values can be used as a measure to detect attacks. Other heuristic functions can also be used in conjunction with these features, which can enumerated as follows:

1. *Number of prediction differences (NPD):* For a given user, the *NPD* is defined as the number of prediction changes after removing that user from the system. Generally,

attack profiles tend to have larger prediction differences than usual, because the attack profiles are designed to manipulate the system predictions in the first place.

2. *Degree of disagreement with other users (DD):* For the ratings matrix $R = [r_{ij}]_{m \times n}$, let ν_j be the mean rating of item j. Then, the degree to which the user i differs from other users on item j is given by $|r_{ij} - \nu_j|$. This value is then averaged over all the $|I_i|$ ratings observed for user i to obtain the degree of disagreement $DD(i)$ of user i:

$$DD(i) = \frac{\sum_{j \in I_i} |r_{ij} - \nu_j|}{|I_i|} \tag{12.4}$$

Users with a larger degree of disagreement with other users are more likely to be attack profiles. This is because attack profiles tend to be different from the distribution of the other ratings.

3. *Rating deviation from mean agreement (RDMA):* The rating deviation from mean agreement is defined as the average absolute difference in the ratings from the mean rating of an item. The mean rating is biased with the inverse frequency if_j of each item j while computing the mean. The inverse frequency if_j is defined as the inverse of the number of users that have rated item j. Let the biased mean rating of an item j be ν_j^b. Let I_i be the set of items rated by user i. Then, the value $RDMA(i)$ for user i is defined as follows:

$$RDMA(i) = \frac{\sum_{j \in I_i} |r_{ij} - \nu_j^b| \cdot if_j}{|I_i|} \tag{12.5}$$

Note the presence of the inverse frequency if_j in the aforementioned equation, so that rare items are given greater importance. It is instructive to compare this equation with Equation 12.4, which does not use such weightings at any stage of the computation. Larger values of this metric indicate the possibility that the user profile might represent an attack.

4. *Standard deviation in user ratings:* This is the standard deviation in the ratings of a particular user. If μ_i is the average rating of user i, and I_i is the set of items rated by that user, then the standard deviation σ_i is computed as follows:

$$\sigma_i = \frac{\sum_{j \in I_i} (r_{ij} - \mu_i)^2}{|I_i| - 1} \tag{12.6}$$

Even though the ratings of fake profiles differ significantly from *other* users, they are often quite *self*-similar because many filler items are set to the same rating value. As a result, the standard deviation σ_i tends to be small for fake profiles.

5. *Degree of similarity with top-k neighbors (SN):* In many cases, attack profiles are inserted in a coordination fashion, with the result being that the similarity of a user with her closest neighbors is increased. Therefore, if w_{ij} is the similarity between the users i and j, and $N(i)$ is the set of neighbors of user i, then the degree of similarity $SN(i)$ is defined as follows:

$$SN(i) = \frac{\sum_{j \in N(i)} w_{ij}}{|N(i)|} \tag{12.7}$$

The value of w_{ij} can be computed with any standard user-user similarity metric, such as the Pearson correlation coefficient.

It is noteworthy that most of these metrics, with the exception of *RDMA*, have also been proposed [43] in the context of finding influential users in a recommender system. This coincidence is because fake profiles are designed by attackers to manipulate the predicted ratings as unusually influential entities in the recommender system. Furthermore, all these metrics, with the exception of the standard deviation, take on larger values in the case of an attack profile. The algorithm in [158] declares a profile to be an attack when all these metrics take on abnormal values in the direction indicative of an attack. Many variations of these basic principles are also possible for designing attack detection methods. Other features may be extracted as well. For example, the presence of an unusually large number of ratings in a profile may be considered suspicious [630].

The aforementioned features are useful not only for unsupervised attack detection algorithms, but also for supervised methods. The main difference between supervised and unsupervised methods is that examples of previous attacks are available. In such cases, these features are used to create a multidimensional representation and a classification model is constructed. For a given user profile for which the attack behavior is unknown, these features can be extracted. The classification model, which is built on the training data of example attacks, can be used on these features to assess the likelihood that it is indeed an attack.

An example of such a supervised attack detection algorithm is discussed in [124]. The metrics discussed above are used as features for the attack detection algorithm. In addition to these features, a number of generic and model-specific features were introduced. *Model-specific features* are designed to detect a specific type of attack, such as an average attack or segment attack. The generic features introduced in [124] are as follows:

1. *Weighted deviation from mean agreement (WDMA):* The *WDMA* metric is similar to the *RDMA* metric, but it places greater weight on the ratings of rare items. Therefore, the square of the inverse frequency is used instead of the inverse frequency in the *WDMA* computation. Therefore, using the same notations as Equation 12.5, the *WDMA* feature is computed as follows:

$$WDMA(i) = \frac{\sum_{j \in I_i} |r_{ij} - \nu_j| \cdot if_j^2}{|I_i|} \tag{12.8}$$

2. *Weighted degree of agreement (WDA):* The second variation of the *RDMA* metric uses only the numerator of the *RDMA* metric, defined by the right-hand side of Equation 12.5:

$$WDA(i) = \sum_{j \in I_i} |r_{ij} - \nu_j| \cdot if_j \tag{12.9}$$

3. *Modified degree of similarity:* The modified degree of similarity is computed in a similar way to the degree of similarity defined by Equation 12.7. The main difference is that the similarity value w_{ij} in Equation 12.7 is proportionally discounted by the number of users who rate both items i and j. This discounting is based on the intuition that the computed similarity is less reliable when the number of items in common between users i and j is small.

In addition, a number of model-specific features have been used in [124]. Readers are referred to [124] for details of these features. Three different algorithms corresponding to the k-nearest neighbor classifier, the C4.5 decision tree, and the support vector machine were

tested. It was found that these different classifiers had different trade-offs between precision and recall of fake profile detection, with the support vector machine providing the best overall performance.

12.4.2 Group Attack Profile Detection

In these cases, the attack profiles are detected as groups rather than as individuals. The basic principle here is that the attacks are often based on groups of related profiles, which are very similar. Therefore, many of these methods use clustering strategies to detect attacks. Some of these methods perform the detection at recommendation time [397], whereas others use more conventional preprocessing strategies [427] in which detection is performed *a priori*, with the fake profiles are removed up front.

12.4.2.1 Preprocessing Methods

The most common approach is to use clustering to remove fake profiles. Because of the way in which attack profiles are designed, authentic profiles and fake profiles create separate clusters. This is because many of the ratings in fake profiles are identical, and are therefore more likely to create tight clusters. In fact, the relative tightness of the clusters containing the fake profiles is one way of detecting them. The method proposed in [427] uses *PLSA* to perform the clustering of the user profiles. Note that *PLSA* already creates a soft clustering, in which each user profile has a particular probability of belonging to an aspect. This soft clustering is converted to a hard clustering by assigning each user profile to the cluster with which it has the largest probability of membership. Although the *PLSA* approach is used for clustering in this case, virtually any clustering algorithm can be used in principle. After the hard clusters have been identified, the average Mahalanobis radius of each cluster is computed. The cluster with the smallest Mahalanobis radius is assumed to contain fake users. This approach is based on the assumption of the relative tightness of the clusters containing fake profiles. Such an approach works well for relatively overt attacks, but not necessarily for subtle attacks.

A simpler approach uses only principal component analysis (*PCA*) [425]. The basic idea is that the covariance between fake users is large. On the other hand, fake users often exhibit very low covariances with other users, when the users are treated as dimensions. How can one identify such highly inter-correlated dimensions with *PCA*, which are not correlated with the normal users? This problem is related to that of *variable selection* in *PCA* [285]. Let us examine the transpose of the ratings matrix in which users are treated as dimensions. According to the theory of variable selection in principal component analysis [427], this problem amounts to that of finding the dimensions (users in the transposed ratings matrix) with small coefficients in the small eigenvectors. Such dimensions (users) are likely to be fake profiles.

The ratings matrix is first normalized to zero mean and unit standard deviation and then the covariance matrix of its transpose is computed. The smallest eigenvector of this matrix is computed. Those dimensions (users) with small contributions (coefficients) in the eigenvector are selected. A slightly more enhanced approach is discussed in [427]. In this case, the top (smallest) 3 to 5 eigenvectors are identified instead of using only the smallest eigenvector. The sum of the contributions over these 3 to 5 eigenvectors is used in order to determine the spam users.

Another algorithm to detect the group profiles is the *UnRAP* algorithm [110]. In the *UnRAP* algorithm, a measure called the H_v-score is used. This measure is adapted from

the bioinformatics area, where it is used in the context of biclustering of gene clusters. Let μ_i be the mean rating of user i, ν_j be the mean rating of item j, γ be the mean over all ratings, and I_i be the set of items rated by user i. Then, the H_v-score of user i is defined as follows:

$$H_v(i) = \frac{\sum_{j \in I_i} (r_{ij} - \mu_i - \nu_j + \gamma)^2}{(r_{ij} - \mu_i)^2} \tag{12.10}$$

Larger values of the H_v-score are more indicative of an attack profile. The basic idea is that fake profiles tend to be self-similar in rating values, but they tend to be different from other users. This is captured by the H_v-score because of the way in which the numerator and denominator are constructed. When the ratings are random, the H_v-score will be close to 1. The algorithm first determines the top-10 users with the largest H_v-scores. This set of users is then used to identify the target item that deviates the most from the mean user rating.

The identification of the target item then sets the stage for the next phase of the algorithm. The criterion for considering users to be candidates for being fake is then relaxed, and more than 10 user profiles are considered as candidates for being fake. However, such candidates will contain many false positives. The *UnRAP* algorithm also discusses methods to remove those users that have not rated the target item, or who have rated the target item in the "wrong" direction. Refer to [110] for details of how the larger candidate set is computed with the use of a sliding-window method.

12.4.2.2 Online Methods

In these methods, the fake profiles are detected during recommendation time. Consider a scenario in which a user-based neighborhood algorithm is used during recommendation time. The basic idea is to create two clusters from the neighborhood of the active user [397]. Note that the main goal of the attacker is to either push or nuke a particular item. Therefore, if a sufficiently large difference exists in the average ratings of the active items in the two clusters, it is assumed that an attack has taken place. The cluster in which the active item has the smaller variance of ratings is assumed to be the attack cluster. All profiles in this attack cluster are removed. This detection method has the merit of being able to be directly integrated into attack-resistant recommendation algorithms during the process of neighborhood formation. Therefore, this approach is not just a method to remove fake profiles, but also an online method for providing more robust recommendations. If desired, the fake profiles can be removed incrementally during the operation of the system.

12.5 Strategies for Robust Recommender Design

A variety of strategies are available for building recommenders in a more robust way. These strategies range from the use of better recommender-system design to better algorithmic design. In the following sections, we will discuss the use of some of these strategies.

12.5.1 Preventing Automated Attacks with CAPTCHAs

It is noteworthy that it requires a significant number of fake profiles in order to result in a significant shift in the predicted ratings. It is not uncommon for the adversary to require between 3% to 5% of the number of authentic profiles to be fake profiles to initiate an attack. For example, consider a ratings matrix containing over a million authentic users.

Figure 12.4: An example of a CAPTCHA from the official CAPTCHA site (http://www.captcha.net)

In such a case, as many as 50,000 fake profiles may be required. It is hard to insert so many fake profiles manually. Therefore, attackers often resort to automated systems to interact with the Web interface of the rating system, and insert the fake profiles.

How can one detect such automated attacks? CAPTCHAs are designed [619] to tell the difference between humans and machines in the context of Web interaction. The acronym CAPTCHA stands for "Completely Automated Public Turing test to tell Computers and Humans Apart." The basic idea is to present a human with distorted text, which is hard for a machine to decipher but can still be read by a human. This distorted text serves as a "challenge" text or word that needs to be entered into the Web interface in order to allow further interaction. An example of a CAPTCHA is illustrated in Figure 12.4. The recommender system can prompt for CAPTCHAs to allow the entry of ratings, especially when a large number of them are entered from the same IP address.

12.5.2 Using Social Trust

The previous chapter reviewed methods for using social trust in the context of a recommender system. In these methods, the social trust between participants is used to influence the ratings. For example, users may specify trust relationships based on their experience with the ratings of other users. These trust relationships are then used to make more robust recommendations. Such methods are able to reduce the effectiveness of attacks, because users are unlikely to specify trust relationships towards fake profiles, which are rather contrived. Detailed discussions of how social trust is used for more effective recommendations are provided in Chapter 11.

The work in [502, 503] proposes an algorithm, referred to as the *influence limiter*, to build trustworthy recommender systems. A global measure of the reputation of each user is used in the recommendation process. Each user is weighted with her reputation score while making the recommendation. The reputation is itself learned on the basis of the accuracy of a user predicting the rating of her neighbors. A theoretical bound on the impact of a negative attack was also shown by this work.

12.5.3 Designing Robust Recommendation Algorithms

It is evident from the discussion in this chapter that different algorithms have different levels of susceptibility to attacks. For example, user-based algorithms are generally much easier to attack than item-based algorithms. Therefore, a number of algorithms have specifically been designed with attack resistance in mind. This section will discuss some of these algorithms.

12.5.3.1 Incorporating Clustering in Neighborhood Methods

It has been shown in [446], how clustering can be used in the context of neighborhood-based methods. This work clusters the user profiles with the use of *PLSA* and k-means techniques. An aggregate profile is created from each cluster. The aggregate profile is based on the average rating of each item in the segment. Then, a similar approach to user-based collaborative filtering is used, except that the aggregate (clustered) profiles are used instead of the individual profiles. For each prediction, the closest aggregated profiles to the target users are used to make recommendations. It was shown in [446] that the clustering-based approach provides significantly more robust results than a vanilla nearest-neighbor method. The main reason for the robustness of this approach is that the clustering process generally maps all the profiles to a single cluster, and therefore limits its influence on the prediction when alternative clusters are available.

12.5.3.2 Fake Profile Detection during Recommendation Time

The attack detection algorithms discussed in the earlier sections can also be used to make robust recommendations, particularly when the detection is done during recommendation time. Such a method is discussed in section 12.4.2.2. In this approach, the neighborhood of the active user is partitioned into two users. An attack is suspected when the active item has very different average values in the two clusters. The cluster that is the most self-similar (i.e., smaller radius) is considered the attack-cluster. The profiles from this cluster are then removed. The recommendations are then performed using the profiles from the remaining cluster. This approach has the dual purpose of being both an attack-detection method and a robust recommendation algorithm.

12.5.3.3 Association-Based Algorithms

Rule-based collaborative filtering algorithms are discussed in section 3.3 of Chapter 3. It was shown in [522] that such algorithms are robust to the average attack when the maximum attack size is less than 15%. The reason for this phenomenon is that there is generally not sufficient support for the attack profiles in order to mount a successful attack. However, such an algorithm is not immune to the segment attack.

12.5.3.4 Robust Matrix Factorization

Matrix factorization methods are generally more robust to attacks because of their natural ability to treat the attack profiles as noise. It has been shown in [424, 427], how *PLSA* methods can be used to detect and remove attacks. Note that many matrix factorization recommenders are themselves based on *PLSA*. Therefore, if the attack profiles are removed in the intermediate step and the probabilistic parameters are renormalized, they can directly be used for recommendation.

Another approach [428] is to modify the optimization function used for matrix factorization to make it more robust to attacks. In matrix factorization, the $m \times n$ ratings matrix R is factorized into user factors and item factors as follows:

$$R \approx UV^T \tag{12.11}$$

Here $U = [u_{is}]$ and $V = [v_{js}]$ are $m \times k$ and $n \times k$ matrices. The predicted value $\hat{r}_{ij}$ of an entry is as follows:

$$\hat{r}_{ij} = \sum_{s=1}^{k} u_{is} v_{js} \tag{12.12}$$

Therefore, the error of predicting an observed entry is given by $e_{ij} = r_{ij} - \hat{r}_{ij}$. As discussed in Chapter 3, the matrix entries of U and V are determined by minimizing the sum of squares of e_{ij} over all the *observed* entries in the matrix R, along with some regularization terms.

How can one change the objective function to de-emphasize the contribution of attacking profiles? The main insight here is attacking profiles often cause outlier entries with large absolute values $|e_{ij}|$ in the *residual matrix* $(R - UV^T)$. Therefore, if one simply used the Frobenius norm of the observed portion of $(R - UV^T)$, the presence of fake profiles would significantly change the user factors and item factors. The natural solution is to de-emphasize the contribution of entries in the residual matrix with large absolute values. Let S be the set of observed entries in the ratings matrix R. In other words, we have:

$$S = \{(i, j) : r_{ij} \text{ is observed}\} \tag{12.13}$$

As discussed in Chapter 3, the objective function of matrix factorization is defined as follows:

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j) \in S} e_{ij}^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2$$

In order to de-emphasize the impact of very large absolute values of e_{ij}, a new set of error terms is defined:

$$\epsilon_{ij} = \begin{cases} e_{ij} & \text{if } |e_{ij}| \leq \Delta \\ f(|e_{ij}|) & \text{if } |e_{ij}| > \Delta \end{cases} \tag{12.14}$$

Here Δ is a user-defined threshold, which defines the case when an entry becomes large. $f(|e_{ij}|)$ is a *damped* (i.e., sublinear) function of $|e_{ij}|$ satisfying $f(\Delta) = \Delta$. This condition ensures that ϵ_{ij} is a continuous function of e_{ij} at $e_{ij} = \pm\Delta$. The damping ensures that large values of the error are not given undue importance. An example of such a damped function is as follows:

$$f(|e_{ij}|) = \sqrt{\Delta(2|e_{ij}| - \Delta)} \tag{12.15}$$

This type of damped function has been used in [428]. The objective function for robust matrix factorization then replaces the error values e_{ij} with the adjusted values ϵ_{ij} as follows:

$$\text{Minimize } J^{robust} = \frac{1}{2} \sum_{(i,j) \in S} \epsilon_{ij}^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2$$

An iterative re-weighted least-squares algorithm, which is described in [426], is used for the optimization process. Here, we describe a simplified algorithm. The first step is to compute the gradient of the objective function J^{robust} with respect to each of the decision variables:

$$\frac{\partial J^{robust}}{\partial u_{iq}} = \frac{1}{2} \sum_{j:(i,j) \in S} \frac{\partial \epsilon_{ij}^2}{\partial u_{iq}} + \lambda u_{iq}, \quad \forall i \in \{1 \ldots m\}, \forall q \in \{1 \ldots k\}$$

$$\frac{\partial J^{robust}}{\partial v_{jq}} = \frac{1}{2} \sum_{i:(i,j) \in S} \frac{\partial \epsilon_{ij}^2}{\partial v_{jq}} + \lambda v_{jq} \quad \forall j \in \{1 \ldots n\}, \forall q \in \{1 \ldots k\}$$

Note that the aforementioned gradients contain a number of partial derivatives with respect to the decision variables. The value of $\frac{\partial \epsilon_{ij}^2}{\partial u_{iq}}$ can be computed as follows:

$$\frac{\partial \epsilon_{ij}^2}{\partial u_{iq}} = \begin{cases} 2 \cdot e_{ij}(-v_{jq}) & \text{if } |e_{ij}| \leq \Delta \\ 2 \cdot \Delta \cdot \text{sign}(e_{ij})(-v_{jq}) & \text{if } |e_{ij}| > \Delta \end{cases}$$

Here, the sign function takes on the value of $+1$ for positive quantities and -1 for negative quantities. The case-wise description of derivative can be consolidated to simplified form as follows:

$$\frac{\partial \epsilon_{ij}^2}{\partial u_{iq}} = 2 \cdot \min\{|e_{ij}|, \Delta\} \cdot \text{sign}(e_{ij}) \cdot (-v_{jq})$$

It is noteworthy that the gradient is damped when the error is larger than Δ. This damping of the gradient directly makes the approach more robust to a few large errors in the ratings matrix. Similarly, we can compute the partial derivative with respect to v_{jq} as follows:

$$\frac{\partial \epsilon_{ij}^2}{\partial v_{jq}} = \begin{cases} 2 \cdot e_{ij}(-u_{iq}) & \text{if } |e_{ij}| \leq \Delta \\ 2 \cdot \Delta \cdot \text{sign}(e_{ij})(-u_{iq}) & \text{if } |e_{ij}| > \Delta \end{cases}$$

As before, it is possible to consolidate this derivative as follows:

$$\frac{\partial \epsilon_{ij}^2}{\partial v_{jq}} = 2 \cdot \min\{|e_{ij}|, \Delta\} \cdot \text{sign}(e_{ij}) \cdot (-u_{iq})$$

One can now derive the update steps as follows, which need to be executed for each user i and each item j:

$$u_{iq} \Leftarrow u_{iq} + \alpha \left(\sum_{j:(i,j) \in S} \min\{|e_{ij}|, \Delta\} \cdot \text{sign}(e_{ij}) \cdot v_{jq} - \lambda \cdot u_{iq} \right) \quad \forall i, \quad \forall q \in \{1 \ldots k\}$$

$$v_{jq} \Leftarrow v_{jq} + \alpha \left(\sum_{i:(i,j) \in S} \min\{|e_{ij}|, \Delta\} \cdot \text{sign}(e_{ij}) \cdot u_{iq} - \lambda \cdot v_{jq} \right) \quad \forall j, \quad \forall q \in \{1 \ldots k\}$$

These updates are performed to convergence. The aforementioned steps correspond to global updates. These updates can be executed within the algorithmic framework of gradient descent (cf. Figure 3.8 of Chapter 3).

One can also isolate the gradients with respect to the errors in individual entries and process them in random order. Such an approach corresponds to *stochastic* gradient descent. For each observed entry $(i, j) \in S$, the following update steps are executed:

$$u_{iq} \Leftarrow u_{iq} + \alpha \left(\min\{|e_{ij}|, \Delta\} \cdot \text{sign}(e_{ij}) \cdot v_{jq} - \frac{\lambda \cdot u_{iq}}{n_i^{user}} \right) \quad \forall q \in \{1 \ldots k\}$$

$$v_{jq} \Leftarrow v_{jq} + \alpha \left(\min\{|e_{ij}|, \Delta\} \cdot \text{sign}(e_{ij}) \cdot u_{iq} - \frac{\lambda \cdot v_{jq}}{n_j^{item}} \right) \quad \forall q \in \{1 \ldots k\}$$

Here n_i^{user} denotes the number of observed ratings for user i and n_j^{item} denotes the number of observed ratings for item j. One cycles through the observed entries in the matrix in

random order and performs the aforementioned update steps until convergence is reached. This is based on the framework of Figure 3.9 (cf. Chapter 3) with the modified set of update steps discussed above. These update steps are different from traditional matrix factorization only in terms of capping the absolute values of the gradient components, when the error is larger than Δ. This is consistent with the stated goals of a robust matrix factorization approach, where large errors might be the result of anomalies in the ratings matrix structure. These anomalies might be indicative of attacks.

It is important to note that this approach will work only when the number of attack profiles is small compared to the correct entries in the ratings matrix. On the other hand, if the number of attack profiles is very large, it will significant affect the factor matrices, and the damping approach will not work. Robust matrix factorization and PCA has a rich history in the context of the recovery of the structure of corrupted matrices. Refer to the bibliographic notes for pointers to work in this area.

Intuitively, the notion of robust matrix factorization is not very different from that of robust regression, which is used commonly to reduce the impact of outliers in regression modeling [512]. In this case, the least-squares optimization function is modified in a similar way to robust matrix factorization. Indeed, robust regression modeling can be used to make many of the collaborative filtering methods in section 2.6 of Chapter 2 more robust. Although there are no existing experimental results on such methods, it is reasonable to assume that robust regression modeling methods are likely to be attack-resistant. This would be an interesting direction of future research in the field.

12.6 Summary

Shilling attacks can significantly reduce the effectiveness of recommender systems because of the presence of fake profiles, which distort the recommendations provided to real users. A variety of push attack methods have been designed in an attempt to influence recommender systems. Some of these include the random attack, the average attack, the bandwagon attack, and the segment attack. Another set of tactics exist, such as the reverse bandwagon attack and the love-hate attack, that are designed to "nuke" items (lower their ratings in the system). Nuke attacks are generally easier to carry out than are push attacks. Attack detection methods use a variety of common characteristics of attacks. These characteristics include the self-similarity of injected profiles, and the differences of these profiles from those of other users. Attack detection methods can be used to design robust recommendation systems. Many robust recommendation systems directly incorporate the fake-profile removal process deep into the recommender system. Other techniques use trustworthy recommender systems or increase fake-profile injection costs. The design of robust recommender systems is a perpetual game between attackers and recommender designers, in which increasingly clever measures and countermeasures are developed by both parties over time.

12.7 Bibliographic Notes

Surveys on shilling attacks and attack-resistant recommender systems may be found in [119, 236]. Attack-resistant methods for collaborative filtering are surveyed in [424]. The idea of using fake user profiles for attacking recommendation algorithms was used in [394]. Some of the earliest methods, such as the average attack and random attack, were proposed and evaluated in [122, 329]. The differential behavior of various recommendation algorithms was

discussed in [329]. For example, it was shown that item-item recommendation algorithms are more robust to attack than user-user recommendation algorithms. A related problem is that of asking users to re-rate items to reduce the effect of *noise* [44] in recommender systems. However, noisy ratings are not necessarily the same as fake profiles, which are created intentionally to mislead the recommender system. The approach in [44], therefore, addresses a different scenario from attack-resistant models.

The bandwagon attack works effectively for user-user collaborative filtering algorithms, but it is not quite as effective for item-based algorithms [246, 329, 445]. The main advantage of the bandwagon attack is that it is nearly as effective as the average attack method, but it requires much less knowledge [329]. A discussion of the popular item attack, along with explanations of the prediction shift, is provided in [395]. The effectiveness of this attack is also studied in [396]. The segment attack was proposed in [445], and it was shown to be effective for item-item collaborative filtering algorithms. The segment attack is a generalization of the favorite item attack [123]. The two nuke attack models, namely the reverse bandwagon attack and the love/hate attack, were proposed in [444]. In group-shilling attacks [572], several human agents cooperate together to either push or nuke an item.

Most of the aforementioned attack systems are designed for the case of explicit ratings. Attack systems for implicit ratings require injection of fake *actions* rather than fake *profiles*. Such systems can be implemented with an automated crawler that simulates Web browsing sessions. The crawler visits carefully selected Web pages in combination, so that the target item is pushed effectively. An example of such an attack is the *popular page* attack in which the target page is crawled together with other popular pages. Such an attack can be viewed as an implicit version of the bandwagon attack. Refer to [79] for a discussion of these strategies.

An unsupervised algorithm for individual/single profile attack detection is discussed in [158]. This algorithm is based on the fact that users with undue influence on the ratings are suspicious. The approach uses a number of metrics discussed earlier for detecting influential users [43]. The presence of an unusually large number of ratings for a user profile may also be considered suspicious [630]. These methods were combined with the *RDMA* metric for unsupervised attack detection. These features were further combined with other features for supervised attack detection [124]. An attack-detection algorithm, which monitors the changes in the ratings over time, is proposed in [668]. The basic idea of this approach is that sudden fake profile injections often lead to anomalous temporal changes in the ratings over time, and they can therefore be detected with time-series monitoring. A related method [78] uses anomaly detection to detect attacks. A method for detection of group shilling attacks is discussed in [572]. In this approach, clusters of users are detected who have co-rated many items and provided atypical ratings compared to other ratings in the database; these clusters are generally fake profiles.

A number of methods for group-based attack detection have also been proposed [110, 425, 427]. The use of principal component analysis (*PCA*) for spam detection is discussed in [425]. The work in [427] discusses the use of *PLSA*-based clustering for group-attack detection. Enhancements of the *PCA* approach, originally discussed in [425], are presented in [427]. The *UnRAP* algorithm is discussed in [110].

A variety of methods can be designed to build attack-resistant recommender systems. CAPTCHAs have been designed [619] to tell humans and computers apart. Such CAPTCHAs can be used to increase the costs of injecting fake profiles into the system. The notion of social trust can also be used to reduce the effectiveness of attacks. Such systems are discussed in detail in Chapter 11. The notion of an influence limiter in order to build attack-resistant recommendation algorithms was proposed in [502, 503]. The integration of

attack detection into attack-resistant recommendation algorithms is discussed in [397]. The use of association methods for building robust algorithms is discussed in [522]. A variety of robust matrix factorization methods for designing attack-resistant recommender systems are discussed in [424, 426–428, 609]. Methods for robust PCA and matrix factorization have also been proposed in the traditional machine-learning literature in other settings, where the low rank structure of corrupted data needs to be recovered [132]. A possible avenue for future research in this area is robust regression in order to reduce the impact of outliers on the recommendation process [512].

One of the challenges with attack-resistant recommender systems is that attackers continue to devise ever more sophisticated methods for attacking the recommender system. For example, attackers might use knowledge of the criteria for detecting attack profiles [397], use obfuscated methods to mount attacks [631], or design attack methods targeting specific collaborative filtering models [522]. Therefore, it is important for research to keep up with the advances in attack algorithms in a perpetual game between the attacker and recommendation system designer.

12.8 Exercises

1. For each of the attack methods discussed in this chapter, write a computer program to implement it.

2. Suppose that you are aware that an average attack has been mounted on your recommender system. Discuss a method to remove the fake profiles.

3. Suppose you had perfect knowledge available about the ratings in the recommender system. In other words, all the ratings in the recommender system are available to you. Show how to design an attack that would be hard to detect. [The answer to this question is not unique.]

4. Implement the online neighborhood method for attack detection (see section 12.4.2.2). Refer to the original publication [397] if needed.