# 9

# Attacks on collaborative recommender systems

When we discussed collaborative filtering techniques in previous chapters of this book, we made the implicit assumption that everyone in the user community behaves honestly and is fair and benevolent. If this assumption holds, all the participants profit: customers are receiving good buying proposals, well-appreciated items get some extra promotion, and the recommendation service itself will be appreciated by web site visitors if its proposals are of high quality and correctly reflect the opinions of the user community.

In the real world, however, the assumption of having only honest and fair users may not hold, in particular when we consider that the proposals of a recommender system can influence the buying behavior of users and real money comes into play. A malevolent user might, for instance, try to influence the behavior of the recommender system in a such way that it includes a certain item very often (or very seldom) in its recommendation list. We shall call this an attack on the recommender system. When a person expresses his or her genuine negative opinion – which can be based on any reasons – this is not seen as an attack. An attack occurs when an agent tries to influence the functioning of the system intentionally.

In general, attackers might have different goals, such as to increase the sales of an item, to cause damage to a competitor, or to sabotage the system as a whole so it is no longer able to produce helpful recommendations. In this chapter we focus on situations in which the goal of the attacker is to promote a certain item or bias the system not to recommend items of a competitor. Note that the manipulation of the "Internet opinion" – for instance, in product review forums – is not a new problem and is therefore not limited to automated recommender systems as discussed in this book. Recent research in marketing supplies evidence not only that consumers' buying behavior can be influenced by community reviews (Chevalier and Mayzlin 2006) but also that the strategic manipulation of opinion forums has a measurable impact on customers and

firms (Dellarocas 2006, Mayzlin 2006). Lam and Riedl (2004), report examples of real-world manipulation attempts on such systems, including attacks on one of Amazon.com's buying tip features[1] or eBay's trust and reputation system[2].

Technically, an "attacker" on a community-based recommender system could try to bias the system's behavior by creating several fake user accounts (profiles) and give the target item of the attack a particularly good or bad rating value. However, such a simple strategy will not work very well in most real systems, which are based on nearest-neighbors methods. Profiles that consist only of one particularly good or bad rating will not be similar to any other user's profile, meaning that their rating values for the target item will never have a significant impact. An attacker therefore needs to find better ways to shill the system (Lam and Riedl 2004) and to make sure that the opinions carried in the fake profiles will take effect.

In the following sections, we discuss the different ways in which collaborative filtering systems can be attacked by malevolent users and summarize recent research that analyzes the vulnerability of different recommendation approaches. Afterward, possible countermeasures are discussed before we briefly review the privacy aspects of recommender systems. Our discussion will naturally be limited to community-based approaches, because since only their behavior can be influenced by a manipulated set of user ratings. Content- and knowledge-based systems can be manipulated only by those involved in the setup of the system unless their knowledge sources are mined from public sources.

## 9.1 A first example

In the following simple example, we sketch the general idea of a profile injection attack; see Mobasher et al. (2007) for a more detailed example. In this simplified scenario, we assume that a simplified version of a memory-based collaborative filtering method is used, which uses Pearson correlation as a similarity measure and a neighborhood size of 1 – that is, only the opinion of the most similar user will be used to make a prediction. Without the fake profile in the last row of the ratings matrix, *User2* is the most similar user, and this user's rating value 2 (dislike) for the target item will be taken as a prediction for *Alice*. However, in the situation of a successful attack, as shown in Table 9.1, the fake profile becomes the most similar one, which means that the particularly high rating for the target item will be taken as a prediction for Alice.

[1] news.com.com/2100-1023-976435.html.
[2] http://www.auctionbytes.com/cab/abn/y03/m09/i17/s01.

Table 9.1. *A profile injection attack.*

| | Item1 | Item2 | Item3 | Item4 | ... | ... | Target | *Pearson* |
|---|---|---|---|---|---|---|---|---|
| Alice | 5 | 3 | 4 | 1 | ... | ... | **?** | |
| User1 | 3 | 1 | 2 | 5 | ... | ... | 5 | −0.54 |
| User2 | 4 | 3 | 3 | 3 | ... | ... | 2 | 0.68 |
| User3 | 3 | 3 | 1 | 5 | ... | ... | 4 | −0.72 |
| User4 | 1 | 5 | 5 | 2 | ... | ... | 1 | −0.02 |
| **Attack** | **5** | **3** | **4** | **3** | | | **5** | 0.87 |

In realistic settings, however, attacking a recommender system by inserting fake profiles to influence its predictions is not that easy. Consider only the following two aspects of that point. First, to be taken into account in the neighborhood formation process, a fake profile must be similar to an existing profile. In general, however, an attacker has no access to the ratings database when trying to determine good values for the selected items (see Table 9.1) that are used to establish the similarity with existing profiles. On the other hand, one single attack profile will not influence the prediction of the system very much when a larger neighborhood size is used. Therefore, several fake profiles must be inserted, which, however, might be also not so easy for two reasons: First, the automatic insertion of profiles is prohibited by many commercial systems (e.g., by using a so-called Captcha, see also Section 9.5 on countermeasures). Second, attack situations, in which many user profiles are created in a relatively short time, can easily be detected by an attack-aware recommender system.

In the following section, we discuss the different possible dimensions of attacks according to Lam and Riedl (2004), as well as more elaborate attack methods, and, finally, show how they influence the predictions of a recommender system as discussed by Mobasher et al. (2007).

## 9.2 Attack dimensions

A first differentiation between possible attack types can be made with respect to the goal of the attack – that is, whether the goal is to increase the prediction value of a target item (*push attack*) or decrease it (called a *nuke attack*). Although by intuition there seems to be no technical difference between these types of goals, we will see later on that push and nuke attacks are actually not always equally effective. Finally, one further possible intent of an attacker can simply be to make the recommender system unusable as a whole.

https://doi.org/10.1017/CBO9780511763113.011 Published online by Cambridge University Press

Another differentiation factor between attacks is whether they are focused only on particular users and items. Targeting a subset of the items or users might be less suspicious. Such more focused (segmented) attacks may also be more effective, as the attack profiles can be more precisely defined in a way that they will be taken into account for the system's predictions with a higher probability. Existing research on vulnerabilities of recommender systems has therefore focused on attack models in single items.

Whether one is able to attack a recommender system in an effective way also depends on the amount of knowledge the attacker has about the ratings database. Although it is unrealistic to assume that the ratings database is publicly available, a good estimate of the distribution of values or the density of the database can be helpful in designing more effective attacks.

Finally, further classification criteria for recommender system attacks include

- *Cost*: How costly is it to insert new profiles? Can the profile injection task be automated, or do we need manual interaction? How much knowledge about the existing ratings in the database is required to launch an attack of a certain type?
- *Algorithm dependence*: Is the attack designed for a specific algorithm, such as a memory-based filtering technique, or is it independent of the underlying algorithm?
- *Detectability*: How easily can the attack be detected by the system administrator, an automated monitoring tool, or the users of the system itself?

## 9.3 Attack types

Consider the most common profile injection attack models (for formal definitions, see Mobasher et al. 2007). The general form of an attack profile, consisting of the target item, a (possibly empty) set of selected items that are used in some attack models, a set of so-called filler items, and a set of unrated items is shown in Table 9.2. The attack types discussed on the following pages differ from each other basically in the way the different sections of the attack profiles are filled.

Note that in the subsequent discussion, we follow the attack type classification scheme by Mobasher et al. (2007); other attack types, which partially also require detailed knowledge about the ratings database, are described by O'Mahoney et al. (2005) and Hurley et al. (2007).

Table 9.2. *Structure of an attack profile.*

| Item1 | ... | ItemK | ... | ItemL | ... | ItemN | Target |
|---|---|---|---|---|---|---|---|
| r_1 | ... | r_k | ... | r_l | ... | r_n | X |
| selected items | | | filler items | | unrated items | | |

### 9.3.1 The random attack

In the *random attack*, as introduced by Lam and Riedl (2004), all item ratings of the injected profile (except the target item rating, of course) are filled with random values drawn from a normal distribution that is determined by the mean rating value and the standard deviation of all ratings in the database.

The intuitive idea of this approach is that the generated profiles should contain "typical" ratings so they are considered as neighbors to many other real profiles. The actual parameters of the normal distribution in the database might not be known: still, these values can be determined empirically relatively easily. In the well-known MovieLens data set, for example, the mean value is 3.6 on a five-point scale (Lam and Riedl 2004). Thus, users tend to rate items rather positively. The standard deviation is 1.1. As these numbers are publicly available, an attacker could base an attack on similar numbers, assuming that the rating behavior of users may be comparable across different domains. Although such an attack is therefore relatively simple and can be launched with limited knowledge about the ratings database, evaluations show that the method is less effective than other, more knowledge-intensive, attack models (Lam and Riedl 2004).

### 9.3.2 The average attack

A bit more sophisticated than the random attack is the *average attack*. In this method, the average rating per item is used to determine the rating values for the profile to be injected. Intuitively, the profiles that are generated based on this strategy should have more neighbors, as more details about the existing rating datasets are taken into account.

In fact, experimental evaluations and a comparison with the random attack show that this attack type is more effective when applied to memory-based user-to-user collaborative filtering systems. The price for this is the additional knowledge that is required to determine the values – that is, one needs to estimate the average rating value for every item. In some recommender systems,

these average rating values per item can be determined quite easily, as they are explicitly provided when an item is displayed. In addition, it has also been shown that such attacks can already cause significant harm to user-based recommenders, even if only a smaller subset of item ratings is provided in the injected profile – that is, when there are many unrated items (Burke et al. 2005).

### 9.3.3 The bandwagon attack

The *bandwagon attack* exploits additional, external knowledge about a rating database in a domain to increase the chances that the injected profiles have many neighbors. In nearly all domains in which recommenders are applied, there are "blockbusters" – very popular items that are liked by a larger number of users. The idea, therefore, is to inject profiles that – besides the high or low rating for the target items – contain only high rating values for very popular items. The chances of finding many neighbors with comparable mainstream choices are relatively high, not only because the rating values are similar but also because these popular items will also have many ratings. Injecting a profile to a book recommender with high rating values for the *Harry Potter* series (in the year 2007) would be a typical example of a bandwagon attack. Another noteworthy feature of this attack type is that it is a low-cost attack, as the set of top-selling items or current blockbuster movies can be easily determined.

In an attack profile as shown in Table 9.2, we therefore fill the slots for the selected items with high rating values for the blockbuster items and add random values to the filler items to ensure that a sufficient overlap with other users can be reached. As discussed by Mobasher et al. (2007), the bandwagon attack seems to be as similarly harmful as the average attack but does not require the additional knowledge about mean item ratings that is the basis for the average attack.

### 9.3.4 The segment attack

The rationale for segment attack (Mobasher et al. 2005) is straightforwardly derived from the well-known marketing insight that promotional activities can be more effective when they are tailored to individual market segments. When designing an attack that aims to push item $A$, the problem is thus to identify a subset of the user community that is generally interested in items that are similar to $A$. If, for example, item $A$ is the new *Harry Potter* book, the attacker will include positive ratings for other popular fantasy books in the injected profile. This sort of attack will not only increase the chances of finding many neighbors in the database, but it will also raise the chances that a typical fantasy book reader will actually buy the book. If no segmentation is done, the new

*Harry Potter* book will also be recommended to users who never rated or bought a fantasy novel. Such an uncommon and suspicious promotion will not only be less effective but may also be easier to detect; see Section 9.5 for an overview of automated attack detection techniques.

For this type of attack, again, additional knowledge – for example, about the genre of a book – is required. Once this knowledge is available, the attack profiles can be filled with high ratings for the selected items and low filler ratings. The segment attack was particularly designed to introduce bias toward item-based collaborative filtering approaches, which – as experiments show (Mobasher et al. 2007) – are, in general, less susceptible to attacks than their user-based counterparts. In general, however, this type of attack also works for user-based collaborative filtering.

### 9.3.5 Special nuke attacks

Although all of the aforementioned attack types are, in principle, suited to both push and nuke individual items, experimental evaluations show that they are more effective at pushing items. Mobasher et al. (2007) therefore also proposed special nuke attack types and showed that these methods are particularly well suited to bias a recommender negatively toward individual items.

- *Love/hate attack*: In the corresponding attack profiles, the target item is given the minimum value, whereas some other randomly chosen items (those in the filler set) are given the highest possible rating value. Interestingly, this simple method has a serious effect on the system's recommendations when the goal is to nuke an item, at least for user-based recommenders. However, it has been shown that if we use the method the other way around – to push an item – it is not effective. A detailed analysis of the reasons for this asymmetry has not been made yet, however.
- *Reverse bandwagon*: The idea of this nuke attack is to associate the target item with other items that are disliked by many people. Therefore, the selected item set in the attack profile is filled with minimum ratings for items that already have very low ratings. Again, the amount of knowledge needed is limited because the required small set of commonly low-rated items (such as recent movie flops) can be identified quite easily.

### 9.3.6 Clickstream attacks and implicit feedback

The attack types described thus far are based on the assumption that a "standard" recommender system is used that collects explicit ratings from registered users.

Although most of today's recommender systems fall into this category, we also briefly discuss attacks on systems that base their recommendations on implicit feedback, such as the user's click behavior.

In the Amazon.com example, a clickstream-based recommender would probably inform you that "users who viewed this book also viewed these items". In such scenarios, the personalization process is typically based on mining the usage logs of the web site. In the context of attacks on recommender systems, it is therefore interesting to know whether and how easily such systems can be manipulated by a malevolent user. This question is discussed in detail by Bhaumik et al. (2007), who describe two possible attack types and analyze how two different recommendation algorithms react to such attacks.

Let us briefly sketch the basic idea of recommending web pages based on nearest-neighbor collaborative filtering and usage logs. The required steps are the following: first, the raw web log data is preprocessed and the individual user sessions are extracted. A user session consists of a session ID and a set of visited pages; time and ordering information are often neglected. Based on these sessions, typical navigation patterns are identified in the mining step, which is commonly based on algorithms such as clustering or rule mining. At run time, a recommender system then compares the set of viewed pages of the current user with these navigation patterns to predict the pages in which the active user most probably will be interested. To calculate the predictions, a procedure similar to the one commonly used for user-based collaborative filtering with cosine similarity measure can be employed.

Attacks on such systems can be implemented by employing an automated crawler that simulates web browsing sessions with the goal of associating a target item (the page to be "pushed") with other pages in such a way that the target items appear on recommendation lists more often. Bhaumik et al. (2007) devise and evaluate two attack types in their experiments. In the segment attack, the target page is visited by the crawler, together with a specific subset of pages that are of interest to a certain subcommunity of all site users. In the *popular page* attack, which is a sort of bandwagon attack, the target page is visited together with the most popular pages.

A first evaluation of these attack types and two recommendation algorithms (kNN and one based on Markov models) showed that recommenders (or, more generally, personalization techniques) based on usage logs are susceptible to crawling attacks and also that – at least in this evaluation – small attack sizes are sufficient to bias the systems. Until now, no advanced countermeasures have been reported; further research in the broader context of usage-based web personalization is required.

## 9.4 Evaluation of effectiveness and countermeasures

Mobasher et al. (2007) present the result of an in-depth analysis of the described attack types on different recommender algorithms. To measure the actual influence of an attack on the outcome of a recommender system, the measures "robustness" and "stability" have been proposed by O'Mahony et al. (2004). *Robustness* measures the shift in the overall accuracy before and after an attack; the *stability* measure expresses the attack-induced change of the ratings predicted for the attacked items. Mobasher et al. (2007) introduce two additional measures. For the push attack, they propose to use the "hit ratio", which expresses how often an item appeared in a top-$N$ list; see also the *ExpTopN* metric of Lam and Riedl (2004). For nuke attacks, the change in the predicted rank of an item is used as a measure of the attack's effectiveness.

### 9.4.1 Push attacks

**User-based collaborative recommenders.** When applied to user-based collaborative systems, the evaluation of the different attacks on the MovieLens data set shows that both the average and the bandwagon attacks can significantly bias the outcome of the recommender system. In particular, it was shown that in both these approaches, a relatively small number of well-designed item ratings in the attack profiles (selected items or filler items) are required to achieve a change in the bias of the recommender. In fact, having too many items in the filler set in the average attack even decreases the achievable prediction shift.

Besides a good selection of the item ratings in the profile, the size of the attack is a main factor influencing the effectiveness of the attack. With an attack size of 3 percent – that is, 3 percent of the profiles are faked after the attack – a prediction shift of around 1.5 points (on a five-point scale) could be observed for both attack types. The average attack is a bit more effective; however, it requires more knowledge about average item ratings than the bandwagon attack. Still, an average increase of 1.5 points in such a database is significant, keeping in mind that an item with a "real" average rating of 3.6 will receive the maximum rating value after the attack.

Although an attack size of only 3 percent seems small at first glance, this of course means that one must inject 30,000 fake profiles into a one-million-profile rating database to reach the desired effect, which is something that probably will not remain unrecognized in a real-world setting. The same holds for the 3 percent filler size used in the experiments. Determining the average item

ratings for 3 percent of the items in a several-million-item database may be a problematic task as well.

A detailed analysis of the effects of different attacks on user-based collaborative filtering systems and parameterized versions, as well as augmented variants thereof (using, e.g., significance weighting or different neighborhood sizes), can be found in O'Mahony et al. (2004).

**Model-based collaborative recommenders.** When attacking a standard item-based algorithm (such as the one proposed by Sarwar et al. (2001)) with the same sets of manipulated profiles, it can be observed that such algorithms are far more stable than their user-based counterparts. Using the same datasets, a prediction shift of merely 0.15 points can be observed, even if 15 percent of the database entries are faked profiles.

The only exception here is the segment attack, which was designed specifically for attacks on item-based methods. As mentioned previously, an attacker will try to associate the item to be pushed with a smaller set of supposedly very similar items – the target segment. Although the experiments of Mobasher et al. (2007) show that the prediction shift for all users is only slightly affected by such an attack, these types of attacks are very effective when we analyze the prediction shift for the targeted users in the segment. Interestingly, the impacts of a segment attack are even higher than those of an average attack, although the segment attack requires less knowledge about the ratings database. Furthermore, although it is designed for item-based systems, the segment attack is also effective when user-based collaborative filtering is employed.

The results of further experiments with additional model-based collaborative filtering techniques are reported by Sandvig et al. (2007) and Mobasher et al. (2006); a summary of the findings is given by Sandvig et al. (2008). The effects of different attacks have been evaluated for a $k$-means clustering method, for clustering based on probabilistic latent semantic analysis (pLSA) (Hofmann and Puzicha 1999), for a feature reduction technique using principal component analysis (PCA), as well as for association rule mining based on the Apriori method. The evaluation of different attacks showed that all model-based approaches are more robust against attacks when compared with a standard user-based collaborative filtering approach. Depending on the attack type and the respective parameterizations, slight differences between the various model-based algorithms can be observed.

Finally, Mobasher et al. (2007) also report on an attack experiment of a hybrid recommender system based on item-based filtering and "semantic similarity" (see Mobasher et al. 2004). Not surprisingly, it can be observed that such combined approaches are even more stable against profile injection attacks,

because the system's predictions are determined by both the ratings of the user community and some additional domain knowledge that cannot be influenced by fake profiles.

### 9.4.2 Nuke attacks

Another observation that can be derived from the experiments by Mobasher et al. (2007) is that most attack types are efficient at pushing items but have a smaller impact when they are used to nuke items. The specifically designed nuke methods are, however, quite effective. The very simple love/hate method described previously, for instance, causes a higher negative prediction shift than the knowledge-intensive average method, which was one of the most successful ones for pushing items. Also, the bandwagon attack is more efficient than other methods when the goal is to nuke items, which was not the case when the purpose was to push items. A detailed explanation of the reasons of this asymmetry of attack effectiveness has not yet been found.

Item-based methods are again more stable against attacks, although some prediction shifts can also be observed. Interestingly, only the love/hate attack type was not effective at all for nuking items in an item-based recommender; the reverse bandwagon turns out to be a method with a good nuke effect.

Overall, the questions of possible attacks on recommender systems and the effectiveness of different attack types have been raised only in recent years. A definite list of attack models or a ranking with respect to effectiveness cannot be made yet, as further experiments are required that go beyond the initial studies presented, for example, by Mobasher et al. (2007).

## 9.5 Countermeasures

Now that we are aware of the vulnerabilities of current recommender system technology, the question arises: how we can protect our systems against such attacks?

**Using model-based techniques and additional information.** So far, our discussion shows that one line of defense can be to choose a recommendation technique that is more robust against profile injection attacks. Most model-based approaches mentioned in the preceding sections not only provide recommendation accuracy that is at least comparable with the accuracy of memory-based kNN approaches, but they are also less vulnerable.

In addition, it may be advisable to use a recommender that does not rely solely on rating information that can be manipulated with the help of fake

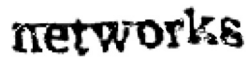


Figure 9.1. Captcha to prevent automated profile creation.

profiles. Additional information, for instance, can be a semantics-based similarity measure, as described above. Alternatively, the recommender could also exploit information about the trust among the different participants in the community, as proposed by Massa and Avesani (2007). Such trust networks among users can not only help to improve the recommendation accuracy, in particular for users who have only a few neighbors, but they could also be used to increase the weights of the ratings of "trusted" friends, thus making it at least harder to attack the recommender system because additional information must be injected into the system's database.

**Increasing injection costs.** A straightforward defense measure is to simply make it harder to automatically inject profiles. The aforementioned experiments show that a certain attack size must be chosen to achieve the desired push or nuke effects. This typically means that the profile creation task must be automated in realistic scenarios because thousands of profiles cannot easily be entered by hand. Standard mechanisms to prevent the automatic creation of accounts include the usage of a Captcha (Von Ahn et al. 2003), as shown in Figure 9.1. A Captcha (Completely Automated Public Turing test to tell Computers and Humans Apart) is a challenge-response test designed to find out whether the user of a system is a computer or a human. A very common test is to present the user with a distorted image showing a text, as in Figure 9.1, and asking the user to type the letters that are shown in the image. Although such techniques are relatively secure as of today, advances are also being made in the area of automatic analysis of such graphical images (see, e.g., Chellapilla and Simard 2004). In addition, it is of course possible to have the Captchas solved by low-cost outsourced labor. Given today's labor cost in some regions, the cost of resolving one Captcha by hand can be as low as one cent per piece.

As another relatively simple measure, the providers of the recommender service can also increase the costs by simply limiting the number of allowed profile creation actions for a single IP address within a certain time frame. However, with the help of onion routing techniques or other source obfuscation and privacy-enhancing protocols, this protection mechanism can also be defeated.

**Automated attack detection.** Defense measures of this type aim to automatically detect suspicious profiles in the ratings database. Profiles can raise suspicion for various reasons, such as because the given ratings are "unusual" when compared with other ratings or because they have been entered into the system in a short time, causing the prediction for a certain item to change quickly. In this section, we briefly summarize different methods that have been proposed to detect fake profiles.

Su et al. (2005) propose a method to detect group shilling attacks, in which several existing users of the system cooperate to push or nuke certain items, an attack type we have not discussed so far because it typically involves human interaction and is not automated, as are the other attack types. Their approach works by detecting clusters of users who have not only co-rated many items, but also have given similar (and typically unusual) ratings to these items. After such clusters are identified based on an empirically determined threshold value, their recommendations can be removed from the database. In contrast to other shilling attacks, the particular problem of "group shilling" is that it is not based solely on the injection of fake profiles and ratings. Instead, in this scenario, users with "normal" profiles (that also contain fair and honest ratings) cooperate in a particular case, so that simple fake profile detection methods might miss these ordinary-looking profiles.

Like other attack-prevention methods, the technique proposed by Su et al. (2005) is designed to cope with a certain type of attack (i.e., group shilling). In principle, we could try to develop specific countermeasures for all known attack types. Because the attack methods will constantly improve, however, the goal is to find detection methods that are more or less independent of the specific attack types.

An approach to detect fake profiles, which is independent from the attack type, is proposed by Chirita et al. (2005). Their work is based on the calculation and combination of existing and new rating metrics, such as the degree of agreement with other users, degree of similarity with top neighbors, or rating deviation from mean agreement. Depending on an empirically determined probability function, the ratings of users are classified as normal or faked. An evaluation of the approach for both the random and average attacks on the MovieLens dataset showed that it can detect fake "push" profiles quite accurately, in particular, in situations in which items are pushed that had received only few and relatively low ratings by other users in the past.

Zhang et al. (2006) take a different approach that is based on the idea that every attack type (known or unknown) will influence the rating distribution of some items over time. Therefore, instead of analyzing rating patterns statically, they propose to monitor the ratings for certain items over time to detect

anomalies. In particular, time series for the following two properties are constructed and analyzed: *sample average* captures how the likability of an item changes over time; *sample entropy* shows developments in the distribution of the ratings for an item. To detect attacks more precisely, changes in these values are observed within limited time windows. The optimal size of such time windows is determined in a heuristic procedure. An experimental evaluation, based again on the MovieLens dataset and simulated attacks, showed that a relatively good detection rate can be achieved with this method and the number of false alarms is limited. Again, however, certain assumptions must hold. In this case, the assumptions are that (a) attack profiles are inserted in a relatively small time window and (b) the rating distributions for an item do not significantly change over time. If this second assumption does not hold, more advanced methods for time series analysis are required. A knowledgeable attacker or group of attackers will therefore be patient and distribute the insertion of false profiles over time.

Finally, a general option for distinguishing real profiles from fake ones is to use a supervised learning method and train a classifier based on a set of manually labeled profiles. Realistic rating databases are too sparse and high-dimensional, however, so standard learning methods are impractical (Mobasher et al. 2007). Mobasher et al. therefore propose to train a classifier on an aggregated and lower-dimensional data set (Bhaumik et al. 2006). In their approach, the attributes of each profile entry do not contain actual item ratings, but rather describe more general characteristics of the profile, which are derived from different statistics of the data set. Similar to Chirita et al. (2005), one part of this artificial profile contains statistics such as the rating deviation from mean agreement. In addition, to better discriminate between false profiles and real but "eccentric" ones, the training profiles contain attributes that capture statistics that can be used to detect certain attack types (such as the random attack). Finally, an additional intraprofile attribute, which will help to detect the concentration of several profiles on a specific target item, is calculated and included in the profile. The training dataset consists of both correct profiles (in this case, taken from the MovieLens dataset) and fake profiles that are generated according to the different attack types described previously. The evaluation of attacks with varying attack sizes, attack models, and filler sizes shows that in some situations, fake profiles can be detected in a relatively precise manner – that is, with few authentic profiles being excluded from the database and most of the false ones detected. For special attack types, however, such as the segment or love/hate attacks, which in particular do not require large attack sizes, the achievable prediction shifts still seem to be too high, even if automated detection of fake profiles is applied.

## 9.6 Privacy aspects – distributed collaborative filtering

A different aspect of security and trustworthiness of recommender systems is the question of user privacy. Rating databases of collaborative filtering recommender systems contain detailed information about the individual tastes and preferences of their users. Therefore, collaborative filtering recommenders – as with many personalization systems – face the problem that they must store and manage possibly sensitive customer information. Especially in the recommender systems domain, this personal information is particularly valuable in monetary terms, as detailed customer profiles are the basis for market intelligence, such as for the segmentation of consumers. On the other hand, ensuring customer privacy is extremely important for the success of a recommender system. After a particular system's potential privacy leaks are publicly known, many users will refrain from using the application further or at least from providing additional personal details, which are, however, central to the success of a community-based system.

The main architectural assumption of collaborative filtering recommender systems up to now were that

- there is one central server holding the database, and
- the plain (unobfuscated and nonencrypted) ratings are stored.

Given such a system design, there naturally exists a central target point of an attack; even more, once the attacker has achieved access to that system, all information can be directly used. Privacy-preserving collaborative filtering techniques therefore aim to prevent such privacy breaches by either distributing the information or avoiding the exchange, transfer, or central storage of the raw user ratings.

### 9.6.1 Centralized methods: Data perturbation

Privacy-preserving variants of CF algorithms that work on centralized, but obfuscated, user data have been proposed for nearest-neighbor methods (Polat and Du 2003), SVD-based recommendation (Polat and Du 2005) and, the Eigentaste method (Yakut and Polat 2007).

The main idea of such approaches can be summarized as follows (Polat and Du 2003). Instead of sending the raw ratings to the central server, a user (client) first obfuscates (disguises) his ratings by applying random data perturbation (RDP), a technique developed in the context of statistical databases to preserve users' privacy. The idea behind this approach is to scramble the original data in such a way that the server – although it does not know the exact values of the

https://doi.org/10.1017/CBO9780511763113.011 Published online by Cambridge University Press

customer ratings but only the range of the data – can still do some meaningful computation based on the aggregation of a large number of such obfuscated data sets. Zhang et al. (2006b) describe this process as preserving privacy by "adding random noise while making sure that the random noise preserves enough of the signal from the data so that accurate recommendations can still be made".

Consider the simple example from Polat and Du (2003) in which the server must do some computation based on the sum of a vector of numbers $A = (a_1, \ldots, a_n)$ provided by the clients. Instead of sending $A$ directly, $A$ is first disguised by adding a vector $R = (r_1, \ldots, r_n)$, where the $r_i$s are taken from a uniform distribution in a domain $[-\alpha, \alpha]$. Only these perturbed vectors $A' = (a_1 + r_1, \ldots, a_n + r_n)$ are sent to the server. The server therefore does not know the original ratings but can make a good estimate of the sum of the vectors if the range of the distribution is known and enough data are available, as in the long run

$$\Sigma_{i=1}^{n}(a_i + r_i) = \Sigma_{i=1}^{n}(a_i) + \Sigma_{i=1}^{n}(r_i) \approx \Sigma_{i=1}^{n}(a_i) \tag{9.1}$$

Similarly, such an estimate can be made for the scalar product, as again, the contribution of the $r_i$s (taken from the uniform distribution $[-\alpha, \alpha]$) will converge to zero. Based on the possibility of approximating the sum and the scalar product of vectors, Polat and Du (2003) devised a nearest-neighbor collaborative filtering scheme that uses $z$-scores for rating normalization. In the scheme, the server first decides on the range $[-\alpha, \alpha]$, which is communicated to the clients. Clients compute $z$-scores for the items they have already rated and disguise them by adding random numbers from the distribution to it. The server aggregates this information based on the above observations for the sum and the scalar product and returns – upon a request for the prediction of an unseen item – the relevant parts of the aggregated information back to a client. The client can then calculate the actual prediction based on the privately owned information about the client's past ratings and the aggregated information retrieved from the server.

The main tradeoff of such an approach is naturally between the degree of obfuscation and the accuracy of the generated recommendations. The more "noise" is included in the data, the better users' privacy is preserved. At the same time, however, it becomes harder for the server to generate a precise enough approximation of the real values. In Polat and Du (2003), results of several experiments with varying problem sizes and parameterizations for the random number generation process are reported. The experimental evaluations show that for achieving good accuracy, when compared with a prediction based on the original data, a certain number of users and item ratings are required,

as the server needs a certain number of ratings to approximate the original data. Given a sufficient number of ratings and a carefully selected distribution function, highly precise recommendations can be generated with this method, in which the original user's ratings are never revealed to others.

Later research, however, showed that relatively simple randomization schemes, such as the one by Polat and Du, may not be sufficiently robust against privacy-breaching attacks, as much of the original information can be derived by an attacker through advanced reconstruction methods; see Zhang et al. (2006a).

Zhang et al. (2006b) therefore propose an extended privacy-preserving CF scheme, in which the main idea is not to use the same perturbation level for all items, as proposed by Polat and Du (2003). Instead, the client and the server exchange more information than just a simple range for the random numbers. In the proposed approach, the server sends the client a so-called perturbation guidance, on which the client can intelligently compute a perturbation that takes the relative importance of individual item ratings into account. The intuition behind this method is that when using an equal perturbation range for all items, this range might be too large for "important" items that are crucial for detecting item similarities; at the same time, the range will be too small for items that are not critical for generating good recommendations (thus causing unnecessary rating disclosure). Technically, the perturbation guidance matrix that captures the importance level for item similarities is a transformation matrix computed by the server based on SVD.

The experiments by Zhang et al. (2006b) show that, with this method, accuracy comparable to the simple randomization method can be achieved even if much more overall "noise" is added to the data, meaning that less of the private information has to be revealed.

### 9.6.2 Distributed collaborative filtering

Another way of making it harder for an attacker to gain access to private information is to distribute the knowledge and avoid storing the information in one central place. Agents participating in such a recommendation community may then decide by themselves with whom they share their information and, in addition, whether they provide the raw data or some randomized or obfuscated version thereof.

**Peer-to-peer CF.** One of the first approaches in that direction was described by Tveit (2001), who proposed to exchange rating information in a scalable peer-to-peer (P2P) network such as Gnutella. In that context, the recommendation

https://doi.org/10.1017/CBO9780511763113.011 Published online by Cambridge University Press

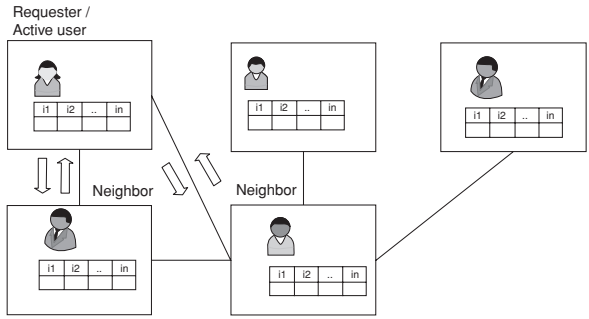


Figure 9.2. Collaborative filtering in P2P environment.

problem was viewed as a search problem, in which the active user broadcasts a query (i.e., a vector of the user's item ratings) to the P2P network. Peers who receive a rating vector calculate the similarity of the received vector with the other known (cached) vectors. If the similarity exceeds a certain threshold, the known ratings are returned to the requester, who can use them to calculate a prediction. Otherwise, the query is forwarded to the neighboring peers and thus spread over the network. Figure 9.2 illustrates a P2P network, in which every user maintains his or her private information and communicates it to his or her neighbors on demand.

Tveit's early approach to distributed CF was proposed in the context of the then-evolving field of mobile commerce but did not include mechanisms for data obfuscation; furthermore, other questions of scalability, cache consistency, and fraudulent users remained open.

**Distributed CF with obfuscation.** In the work by Berkovsky et al. (2007), therefore, an approach was proposed and evaluated that combines the idea of P2P data exchange and data obfuscation. Instead of broadcasting to the network the "raw" profile and the identification of the target item for which a recommendation is sought, only an obfuscated version is published. Members of the network who receive such a request compute the similarity of the published profile with their own and return a prediction for the target item (if possible) alongside a number expressing the degree of profile similarity. The request-seeking user collects these answers and calculates a prediction using a standard nearest-neighbor method.

In this protocol, disclosure of private profile data occurs in two situations. When the requester sends out his or her profile, he or she inevitably needs to include some personal rating information. On the other hand, the responding agent's privacy is also endangered when he or she returns a rating for the target

https://doi.org/10.1017/CBO9780511763113.011 Published online by Cambridge University Press

item. Although this is merely one single rating, an attacker could use a series of requests (probe attack) to incrementally reconstruct the profile of an individual respondent. Obfuscation (e.g., through randomization) will help to preserve the privacy of the participants. It is advisable, however, to perturb only the profiles of the respondent agents to a significant extent, as obfuscation of the critical requester profile quickly deteriorates recommendation accuracy.

Again, a tradeoff between privacy protection and recommendation accuracy exists: the more the profiles are obfuscated, the more imprecise the results are going to be. Berkovsky et al. (2007) analyze different obfuscation variants and their effects on the recommendation accuracy on the MovieLens dataset. The obfuscation schemes can be varied along different dimensions. First, when replacing some of the original values with fake values, one can use different value distributions. The options range from fixed and predefined values over uniform random distributions to distributions that are similar to the original rating (bell-curve) distributions. Another dimension is the question of whether all ratings should be perturbed or whether it is better to obfuscate only the extreme ratings (see also the earlier discussion of important ratings). Finally, the returned predictions may or may not also be obfuscated.

The following effects of the different obfuscation strategies can be observed. First, it does not seem to matter much whether the fake values are taken from a random or bell-curve distribution or we replace original values with the "neutral" value (e.g., 3 on a 1-to-5 scale). The only major influence factor on the accuracy is the percentage of randomized values. With respect to the effects of obfuscation on "moderate" and "extreme" ratings, the prediction accuracy for extreme ratings (e.g., 1 and 5 on the 1-to-5 scale) quickly deteriorates but remains rather stable for moderate ratings. Still, when the percentage of obfuscated values increases too much, accuracy worsens nearly to the level of unpersonalized recommendations. Overall, the experiments also underline the assumption that obfuscating the extreme ratings, which intuitively carry more information and are better suited to finding similar peers, can quickly degrade recommendation accuracy. Perturbing moderate ratings in the data set does not affect accuracy too much. Unfortunately, however, the unobfuscated extreme ratings are those that are probably most interesting to an attacker.

**Distributed CF with estimated concordance measures.** Lathia et al. (2007) pick up on this tradeoff problem of privacy versus accuracy in distributed collaborative filtering. The main idea of their approach is not to use a standard similarity measure such as Pearson correlation or cosine similarity. Instead, a so-called concordance measure is used, which leads to accuracy results

https://doi.org/10.1017/CBO9780511763113.011 Published online by Cambridge University Press

comparable to those of the Pearson measure but that can be calculated without breaching the user's privacy.

Given a set of items that have been rated by user $A$ and user $B$, the idea is to determine the number of items on which both users have the same opinion (concordant), the number of items on which they disagree (discordant), and the number of items for which their ratings are tied – that is, where they have the same opinion or one of the users has not rated the item. In order to determine the concordance, the deviation from the users' average is used – that is, if two users rate the same item above their average rating, they are concordant, as they both like the item. Based on these numbers, the level of association between $A$ and $B$ can be computed based on Somers' $d$ measure:

$$d_{A,B} = \frac{NbConcordant - NbDiscordant}{NbItemRatingsUsed - NbTied} \tag{9.2}$$

One would intuitively assume that such an implicit simplification to three rating levels (agree, disagree, no difference) would significantly reduce recommendation accuracy, as no fine-grained comparison of users is done. Experiments on the MovieLens dataset (Lathia et al. 2007), however, indicate that no significant loss in recommendation precision (compared with the Pearson measure) can be observed.

The problem of privacy, of course, is not solved when using this new measure, because its calculation requires knowledge about the user ratings. To preserve privacy, Lathia et al. (2007) therefore propose to exploit the transitivity of concordance to determine the similarity between two users by comparing their ratings to a third set of ratings. The underlying idea is that when user $A$ agrees with a third user $C$ on item $i$ and user $B$ also agrees with $C$ on $i$, it can be concluded that $A$ and $B$ are concordant on this item. Discordant and tied opinions between $A$ and $B$ can be determined in a similar way.

The proposed protocol for determining the similarity between users is as follows:

- Generate a set of ratings $r$ of size $N$ using random numbers taken uniformly from the rating scale. Ensure that all values are different from the mean of the rating set $r$.
- Determine the number of concordant, discordant, and tied ratings of user $A$ and user $B$ with respect to $r$.
- Use these value pairs to determine the upper and lower bounds of the real concordance and discordance numbers between $A$ and $B$.
- Use the bounds to approximate the value of Somers' $d$ measure.

https://doi.org/10.1017/CBO9780511763113.011 Published online by Cambridge University Press

The calculations of the upper and lower bounds for the concordance and discordance numbers are based on theoretical considerations of possible overlaps in the item ratings. Consider the example for the calculation of the bounds for the *tied* ratings. Let $T_{A,r}$ be the number of ties of $A$ with $r$ and $T_{B,r}$ be the number of ties of $B$ with $r$. The problem is that we do not know on which items $A$ and $B$ were tied with $r$. We know, however, that if by chance $A$ and $B$ tied with $r$ on exactly the same items, the lower bound is $max(T_{A,r}, T_{B,r})$; accordingly, the upper bound is $(T_{A,r} + T_{B,r})$ if they are tied on different items. If $(T_{A,r} + T_{B,r})$ is higher than the number of items $N$, the value of $N$ is, of course, the upper bound. Similar considerations can be made for the bounds for concordant and discordant pairs.

Given these bounds, an estimate of the similarity based on Somers' $d$ measure can be made by using the midpoint of the ranges and, in addition, weighting the concordance measures higher than the discordance measure as follows (midpoint values are denoted with an overline):

$$predicted(d(A, B)) = \frac{\overline{NbConcordant} - 0.5 \times \overline{NbDiscordant}}{NbItemRatingsUsed - \overline{NbTied}} \tag{9.3}$$

With respect to recommendation accuracy, first experiments reported by Lathia et al. (2007) show that the privacy-preserving concordance measure yields good accuracy results on both artificial and real-world data sets.

With respect to user privacy, although with the proposed calculation scheme the actual user ratings are never revealed, there exist some theoretical worst-case scenarios, in which an attacker can derive some information about other users. Such situations are very unlikely, however, as they correspond to cases in which a user has not only rated all items, but by chance there is also full agreement of the random sets with the user's ratings. The problem of probing attacks, however, in which an attacker requests ratings for all items from one particular user to learn the user model, cannot be avoided by the proposed similarity measure alone.

**Community-building and aggregates.** Finally, we mention a relatively early and a slightly different method to distribute the information and recommendation process for privacy purposes, which was proposed by Canny 2002a, 2002b. In contrast to the pure P2P organization of users described earlier, Canny proposed that the participants in the network form knowledge communities that may share their information inside the community or with outsiders. The shared information, from which the active user can derive predictions, is only an aggregated one, however, based, for example, on SVD (Canny 2002a). Thus, the individual user ratings are not visible to a user outside the community who

https://doi.org/10.1017/CBO9780511763113.011 Published online by Cambridge University Press

requests information. In addition, Canny proposes the use of cryptographic schemes to secure the communication between the participants in the network, an idea that was also picked up by Miller et al. later on (2004) in their PocketLens system.

Overall, Canny's work represents one of the first general frameworks for secure communication and distributed data storage for CF. The question of how to organize the required community formation process – in particular, in the context of the dynamic web environment – is not yet fully answered and can be a severe limitation in practical settings (Berkovsky et al. 2007).

## 9.7 Discussion

Recommender systems are software applications that can be publicly accessed over the Internet and are based on private user data. Thus, they are "natural" targets of attacks by malicious users, particularly because in many cases real monetary value can be achieved – for example, by manipulating the system's recommendation or gaining access to valuable customer data.

First, we discussed attacks on the correct functioning of the systems, in which attackers inject fake profiles into the rating database to make a system unusable or bias its recommendations. An analysis of different attack models showed that, in particular, standard, memory-based techniques are very vulnerable. For model-based methods, which are based, for instance, on item-to-item correlation or association rules, no effective attack models have been developed so far. Intuitively, one can assume that such systems are somehow harder to attack, as their suggestions are based on aggregate information models rather than on individual profiles.

Hybrid methods are even more stable, as they rely on additional knowledge that cannot be influenced by injecting false profiles.

Besides the proper choice of the recommendation technique, other countermeasures are possible. Aside from making it harder (or impossible) to inject a sufficient number of profiles automatically, one option is to monitor the evolution of the ratings database and take a closer look when atypical rating patterns with respect to the values or insertion time appear.

Unfortunately, no reports of attacks on real-world systems are yet publicly available, as providers of recommendations services are, of course, not interested in circulating information about attacks or privacy problems because there is money involved. It is very likely, however, that different attacks have been launched on popular recommender systems, given the popularity of such systems and the amounts of money involved. Future research will require

cooperation from industry to crosscheck the plausibility of the research efforts and to guide researchers in the right direction.

Gaining access to (individual) private and valuable user profiles is the other possible goal of an attack on a recommender system discussed in this chapter. Different countermeasures have been proposed to secure the privacy of users. The first option is to obfuscate the profiles – for instance, by exchanging parts of the profile with random data or "noise". Although this increases privacy, as the real ratings are never stored, it also reduces recommendation accuracy. The other option is the avoidance of a central place of information storage through the distribution of the information. Different P2P CF protocols have therefore been developed; typically, they also support some sort of obfuscation or additional measures that help to ensure the user's privacy.

What has not been fully addressed in distributed CF systems is the question of recommendation performance. Today's centralized, memory-based, and optimized recommendation services can provide recommendations in "near real time." How such short response times, which are crucial for the broad acceptance of a recommender system, can be achieved in a distributed scenario needs to be explored further in future work.