Chapter 6

Temporal Defences for Robust Recommendations

Recommender systems are vulnerable to attack: malicious users may deploy a set of sybils to inject ratings in order to damage or modify the output of CF algorithms. Previous work focuses on designing *sybil profile* classification algorithms, which operate independently of CF, and aim to find the current sybils each time they are run. These methods, however, assume that the full sybil profiles have already been input to the system. As previously observed, deployed recommender systems, on the other hand, operate over time: recommendations may be damaged as sybils inject profiles (rather than only when all the malicious ratings have been input), and system administrators may not know when their system is under attack. In this chapter, we address the problem of *temporal* sybil attacks, and propose and evaluate methods for monitoring *global, user* and *item* behaviour over time in order to detect rating anomalies that reflect an *ongoing* attack. We conclude by discussing the consequences of our temporal defences, and how attackers may design *ramp-up attacks* in order to circumvent them.

6.1 Problem Setting

When CF algorithms compute recommendations for web users, they do so assuming that the ratings they manipulate are *honest* depictions of user preferences. Unfortunately, this may not be the case: any system that invites participation is also vulnerable to malicious abuse, and the ratings input to CF algorithms may have been fabricated to damage or modify the recommendations the system outputs. Abusing recommender systems this way is often referred to as *shilling*, *profile injection* or *sybil* attacks; Mobasher *et al*. provide an in-depth review of this problem [MBW07]. All of these terms are synonymous: attackers deploy a set of pseudonymous entities (e.g., automated bots) that rate content in a manner that serves the attacker's goals. These attacks are further categorised based on the intent of the attacker: a *random* attack aims to disrupt the system by injecting noise, while *targetted* attacks aim to promote or demote the ranking (and thus, the recommendability) of individual items. There is a growing body of research that addresses the problem of identifying malicious profiles; for example, Williams *et al*. evaluate the potential that a variety of classifiers have to find sybils [WMB09]. In other words, given a matrix of user-item ratings that contains a set of sybil profiles, the problem to be solved is how to divide the hon-

est from the malicious profiles in order to exclude the sybils. The underlying assumption here is that the *full sybil profiles* are already contained within the user-item matrix: all the sybils have rated all the items that they intend to in order to perform their attack. However, recommender systems are subject to change over time, as users input more ratings and CF algorithms are retrained in order to serve the most up-to-date recommendations. This reality of deployed recommender systems presents two challeges to the assumptions held by attack detection algorithms:

1. The sybil profiles may not be fully inserted or may be inserted over an extended period of time; thus reducing their immediate detectability, while not necessarily reducing the damage they may inflict on the system.
2. As the system is updated, the problem of *when to run* expensive detection algorithms arises: how can system administrators know that their system is potentially under attack?

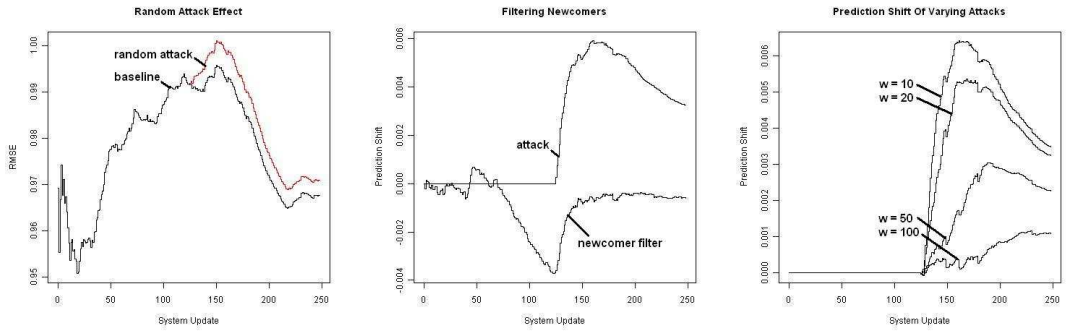
In this chapter, we address these challenges by designing and evaluating algorithms for monitoring recommender system users' behaviour over time. To do so, we make the following contributions:

- We preface this chapter in Section 6.2 by showing how non-temporal profile injection attacks (where the sybils appear, dump malicious ratings over a number of days, and dissapear) can be defeated easily; attackers therefore have an incentive to extend the time taken to inject profiles. We explore this incentive with experiments comparing random profile injection attacks over varying time lengths.
- Based on the previous experiments, we describe the range of potential temporal sybil attacks in Section 6.3. In doing so, we relate the number of sybils and number of ratings input per sybil over time to previously defined (random/targetted) attack models, highlighting the relation between how an attack is carried out and the intent of the attacker.
- In Section 6.4 we describe and evaluate methods of monitoring global, user and item activity over time and identifying anomalies that reflect and flag an ongoing attack.
- Based on the defences we construct, we analyse how attackers may respond, and propose directions for future research by defining adaptive attack models in Section 6.5. We close by discussing related work and concluding in Sections 6.6 and 6.7.

6.2 Defeating Non-Temporal Attacks

We use the same model of recommender system temporal updates as we did in previous chapters: given a dataset at time t, and a window size μ (reflecting how often the system will be updated), we train the algorithm with any data input prior to t and predict any ratings input between t and $(t + \mu)$. We also use the same five subsamples of Netflix user profiles we examined previously. However, in this chapter we reduce the breadth of CF algorithms we consider; we focus only on the item-based kNN algorithm, with the $k = 50$ neighbours weighted as described in [LHC08c] and Section 2.3.2:

$$w(a,b) = \frac{1}{|R_a|}\left(\sum_{i \in R_a} value(a,b,i)\right) \tag{6.1}$$



(a) One-Shot Random Attack (b) Effect of Filtering Newcomers (c) Different Attack Lengths

Figure 6.1: Time-Averaged RMSE Of One-Shot Attack, and Prediction Shift When Pruning Newcomer's Ratings, and Injecting Attacks Over Varying Time Windows

Where the *value* of two ratings is defined as:

$$value(a, b, i) = 1 - \rho|r_{a,i} - r_{b,i}| \tag{6.2}$$

We made this decision for two reasons: (a) comparisons of inter-algorithm attack robustness have already been done [MBW07], and (b) our goal is to design algorithm-independent mechanisms for identifying temporal attacks.

A non-temporal attack would operate as follows: between time t and $(t + \mu)$, a set of sybils S would input a set of ratings X; the non-temporal characteristic of this attack is that all the malicious ratings are input within a single window (while the *actual* time taken to operate the attack may span the size of the entire window). We can then measure the change to the temporal performance with the time-averaged RMSE metric. We visualise the temporal effects of a non-temporal attack with the following example. During the $125th$ week-long window in the Netflix data, we inserted 100 sybils who each rated approximately $10,000$ selected items. In this example, we limit ourselves to exploring the temporal effect of a *random* attack: each sybil randomly picks one of the available items, and then rates it with a random value drawn uniformly from the rating scale. Figure 6.1(a) plots the impact that these ratings have on the time-averaged RMSE.

The random attack has a pronounced effect on the time-averaged RMSE: performance is consistently degraded over the rest of the updates. However, this attack is simple: sybils appear, rate within the window length, and disappear (a "one-shot" attack). Furthermore, at each update the system re-trains using all historic data, regardless of whether the users who have input that data continue to reappear. The natural response is therefore to *distrust newcomers*: any ratings from new users are *suspect*. In Figure 6.1(b) we repeated the previous experiment, but excluded suspect ratings from the item-similarity computation step of the kNN algorithm. By excluding suspect ratings this way, we maintained our ability to formulate recommendations for all users (including sybil and new honest users- should they reappear), while removing the influence that suspect ratings exert on the item neighbourhoods. We plot *prediction shift* values, i.e., the difference between the baseline (predictions with no sybils inserted) and the attack and newcomer-filtered scenarios. While we certainly pruned away the ratings of non-sybil users, the

technique not only eliminates the effect of the attack, but also *improves* upon the baseline RMSE in a number of windows prior to the attack taking place (i.e., the prediction shift is negative). Removing the ratings of users who appear, rate, and do not return thus avoids one-shot attacks and seems to take small steps towards de-noising the data [APTO09].

Given this situation, an attacker may simply respond by widening the number of windows taken to inject ratings. Sybils under the attacker's control would therefore appear in multiple windows and, after the first appearance, no longer be suspect. In order to explore the incentives that attackers have to rate-limit their own sybils, we performed a number of random attacks, where a set of 100 sybils rated the same number of items over a varying number of sequential windows $W_a \in \{10, 20, 50, 100\}$. In each case, the *number* of malicious ratings remained the same, the only difference being the *time* taken to insert them; we compare attacks of the same magnitude that differ only in temporal spread (i.e., the ratings per sybil per window varies, as does the number of windows). The results in Figure 6.1(c) show that injecting ratings over longer time periods deviates the RMSE from the baseline less. This is likely to be an effect of the balance between sybil and non-sybil ratings: longer attacks have less of an effect since, during the time taken to operate the attack, there is a larger influx of non-sybil ratings.

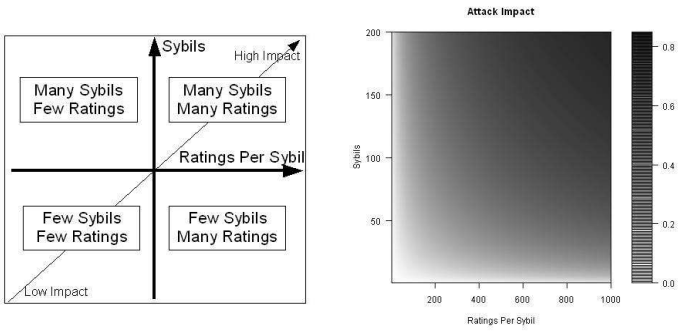
We draw two conclusions from the above experiments: there is an incentive for attackers to (a) inject ratings over more than one window (in order to not have their ratings be suspect), and (b) inject as much data as possible, in order to have the greatest possible effect (since higher volumes of sybil ratings per window has a more pronounced effect). With this in mind, we describe temporal attacks by considering the choices attackers must make when designing a sybil attack.

6.3 Temporal Attack Models

Previous work [MBW07] examines different regions of sybil profiles, looking at what items sybils must rate in order to define different attacks. This structure still holds on the temporal scale; the difference is how long it takes the attacker to construct the sybil profiles. In this chapter, we do not assume that the profiles are populated in the same order (i.e., all sybils rate movie m_1 first, m_2 second, etc), or that they even all contain the same items; instead, we assume that the *rate* at which they are populated is roughly similar (in Section 6.5 we discuss the consequences of breaking the latter assumption).

There are a number of factors that attackers control when they implement a temporal attack: *how many* sybils should rate content, the *rate* at which sybils should inject ratings, and *how long* they should continue rating for. Attacks can thus be classified according to how attackers calibrate these factors, and whether they hit the system with (*many, few*) sybils rating (*many, few*) items per window, for a predefined sequence of windows. Figure 6.2(a) summarises this view. Each quadrant represents a combination of these two variables; a third dimension (not pictured here) would represent the rate of attack. This is important because these variables reflect the rationale of the ongoing attack. For example, many sybils rating many items translates to inputting a high volume of malicious data, and may reflect an ongoing random attack.

The relation we have outlined above is important since it explores how an intelligent attacker would go about achieving particular goals when attacking a recommender system. However, there are a number



(a) Attack Classification (b) Attack Impact With No Defences

Figure 6.2: Attack Types and Impact With No Defences

of factors that attackers cannot control, related to how the non-sybil users behave: how many *non-sybil* users there are, the number of *ratings* that they input per window, *what* they rate, and *how* they rate. We will leverage this information in order to construct a defence to temporal attacks, which we introduce in the next section.

6.3.1 Measuring Attacks

There are a number of ways of measuring the effect of attacks, ranging from prediction shift, to hit ratio, and average rank [MBW07]; each aims to measure changes introduced by malicious ratings. In this chapter, we are interested in measuring how well our defences detect ongoing attacks (rather than how the attacks change recommendations); we thus focus on the detection *precision*, *recall* and the potential attack *impact*. Given a window t, and a set of sybils S_t who rate X_t items each during the window, the impact of the attack is simply the number of sybil ratings input at the current window t, or $S_t \times X_t$, divided by the total number of ratings R_t input in that window:

$$impact_t = \begin{cases} \frac{S_t \times X_t}{R_t}, & \text{if attack is undetected} \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

In measuring attacks this way, we assume that, if a system administrator can be told that the system is under attack, then one of the many sybil-identifying classifiers that are described in the literature can be used to prune the actual sybil ratings. If no attack is flagged, then we measure the relative size of the attack. This metric gives higher weight to attacks that inject more ratings; Figure 6.2(b) plots the attack impact for varying sizes of sybil groups and rating rates when no defences are in place. While it is certainly possible to envisage attacks that, by carefully tuning what the sybils rate, cause more damage with fewer ratings than higher volume equivalents, in this chapter we are not concerned with comparing attacks to each other. Instead, we use the metric to see how many ratings attackers can slip into the system without causing behavioural anomalies. We also measure the number of true positives (TP, attacks that were flagged), false positives (FP, non-attacks that are flagged), and false negatives (FN, attacks that were not flagged). Precision and recall are then computed as:

$$precision = \frac{TP}{TP + FP}; \quad recall = \frac{TP}{TP + FN} \tag{6.4}$$

All these metrics, however, are related: the precision and recall relate the proportions of false positives and negatives to the true positives, while the impact, by being non-zero when an attack slips through, displays the false negatives in a manner that takes into account the size of the attack that failed to be detected. In effect, we have two metrics that explore facets of false negatives. The emphasis we place on false negatives throughout this chapter is motivated as follows: we cannot know (and only assume) that the data we experiment with is the fruit of honest, well-intentioned users; similarly, we can only know that an attack is taking place when we manually insert it. We therefore place a higher importance on reducing false negatives (i.e., finding all the attacks that we insert) within the data that we have: false positives in the real data may very well be attacks that produce anomalous behaviour, and are likely to deserve further inspection. However, we note here that the defences described below produced no false positives when run on the temporal rating data with no attacks manually injected.

6.4 A Temporal Defence

In the above section we outlined the factors that attackers determine: the *time* (number of windows), *size* (number of sybils), *rate* (number of ratings per sybil per window), and *strategy* (which items need to be rated: random/targetted) when implementing an attack. In this section, we describe a method of detecting different forms of attacks, based on monitoring the global behaviour (Section 6.4.1), user behaviour (Section 6.4.2), and item behaviour (Section 6.4.3) for anomalies. The key to our proposal is that attacks may be identifiable by finding consistent anomalies caused by the sybil group's behaviour.

6.4.1 Global Thresholding

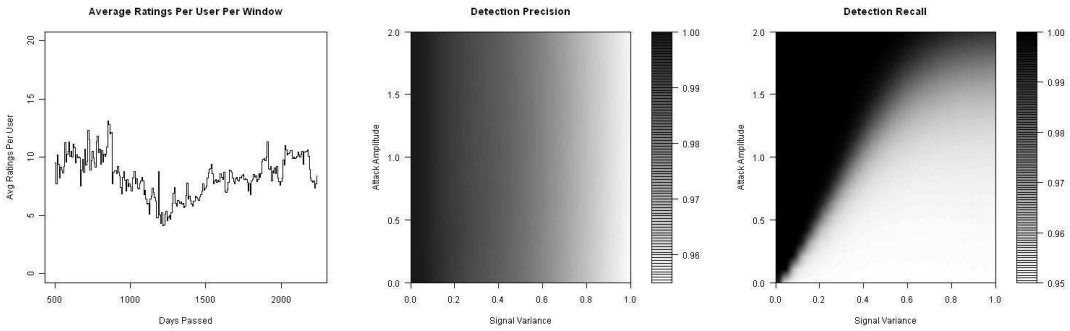
The first perspective of system behaviour that we consider is at the *global*, or aggregate, level. While the number of ratings that users input varies over time, the average ratings per user per window (in the Netflix data) remains relatively flat: Figure 6.3(a) plots this value over time. From this, we see that the average user will rate between $5 - 15$ movies per week. Since the mean is derived from a long-tailed distribution, it is a skewed representation of the "average" user. However, an attacker, by deploying a group of sybils who inject ratings at a pre-defined rate, will modify this aggregate value; the first dimension of our defence thus aims at monitoring changes to the average ratings per user MU_t over time. Given a window t, the current mean ratings per user MU_t, standard deviation σ_t, the R_t ratings input by U_t users an alarm is raised if the volume of incoming ratings departs from the mean measured to date by an amount determined with a global threshold $\alpha_g \geq 1$:

$$\frac{R_t}{U_t} \geq (MU_t + (\alpha_g \times \sigma_t)) \tag{6.5}$$

Otherwise, we update the current MU_t value as an exponentially weighted moving average (with a weighting factor β_t):

$$MU_t = (\beta_t \times MU_{t-\mu}) + ((1 - \beta_t) \times \frac{R_t}{U_t}) \tag{6.6}$$

MU_t is updated *conservatively*: if an attack is flagged, then it is not updated. We also update both the α_t and β_t variables. The β_t variable determines the weight that is given to historical data: relying too heavily on historical data will not capture current fluctuations, while weighting current values too highly



(a) Avg Ratings Per User (b) Simulation Precision (c) Simulation Recall

Figure 6.3: Netflix Ratings Per User Per Week; Global Thresholding Precision and Recall

will disperse temporal trends. We therefore determine the next value $\beta_{t+\mu}$ with the standard deviation measured to date:

$$\beta_{t+\mu} = \min(|\sigma_{t-\mu} - \sigma_t|, 1) \tag{6.7}$$

The value is capped at 1, thus ensuring that when there is high variability in the data, β_t gives higher preference to current values, while smaller standard deviation shifts β_t to give higher weight to historical values. The α_t variable determines the extent to which the current $\frac{R_t}{U_t}$ value can deviate from MU_t before an attack is flagged. When an attack is flagged, we reduce α_t, in effect, making it more difficult for attackers to learn the appropriate threshold. We set α_t to jump between pre-specified values (0.5 and 1.5):

$$\alpha_{t+\mu} \begin{cases} 1.5, & \text{if no attack detected} \\ 0.5, & \text{otherwise} \end{cases} \tag{6.8}$$

Monitoring incoming ratings at the aggregate level is sensitive to two factors: how naturally variable the incoming ratings are, and the amount of variance that attacks introduce. In other words, a mechanism like this may not work if there is already high variance in the average ratings per user and sybils do not displace the mean value. We therefore evaluated this technique with two methods: in the first, we *simulate* a stream of incoming ratings (in order to control both the variance and size of attack); we then turned to *real data* where we could explore the effects of varying attacks in a more realistic setting.

In order to simulate a stream of incoming ratings, we draw a sequence of $\frac{R_t}{U_t}$ values from a normal distribution with (fixed) mean MU and standard deviation $\sigma \in [0, MU]$. Then, at random moments, we simulate an injected attack where a group of sybils shifts the incoming value by the attack *amplitude* $\gamma \in [0, (2 \times MU)]$; in other words, at an attack time t, the window's value is $(\frac{R_t}{U_t} + \gamma)$. We then note whether an attack was flagged, and can compute the detection precision and recall with the results.

When running the simulation, we assumed that, after a brief training phase, the system could be attacked at any time during a period of $1,000$ windows, for a pre-determined number (50) of sequential attack windows. We re-ran each simulation parameter setting $10,000$ times and present averaged results. Figure 6.3(b) shows the resulting precision, which fades as σ increases, but is otherwise dependent on σ (the variability in the ratings per user per window) rather than the attack amplitude γ. In other words,

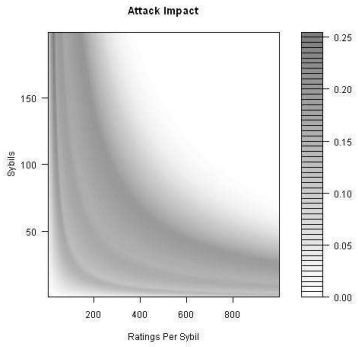


Figure 6.4: Global Thresholding Impact

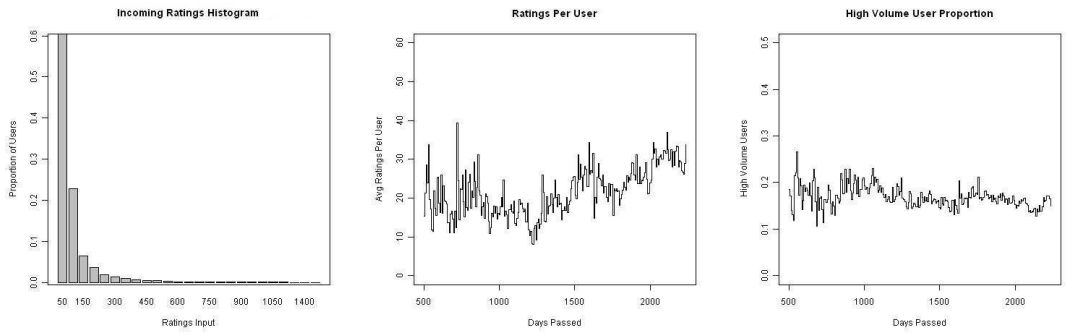
the number of false positives depends on how naturally variable the data is, and, given that the real data displays low spread, the number of false positives is likely to be low. Figure 6.3(c), instead, displays the detection recall. This plot highlights the trade-off between σ and γ: the best recall is when a small σ is modified with a large γ, while the worst values are found when a large σ is deviated by a small γ. However, we note that the minimum precision is slightly below 0.90, while the minimum recall remains above approximately 0.95.

We returned to the Netflix subsets in order to test this method with real data. To do so, we trained our monitor with all ratings per window until the attack time, and then measure the attack impact after injecting the attack. Since the attacker may unleash the sybils at any time, we repeated our experiments, starting attacks at each possible window, and plot average results across all windows. As Figure 6.4 shows, this method catches attacks where large groups of sybils inject their profiles at a very high rate; the top right corner of the plot is flattened to zero impact. However, two sensitive areas remain: first, where *many* sybils inject *few* ratings, and when *few* sybils inject *many* ratings. Attackers can thus respond by either reducing the size of the sybil group, or the the sybil's rate. However, this plays into our hands: in Section 6.4.2 we address the former, while Section 6.4.3 describes how the latter attacks can also be monitored.

6.4.2 User Monitoring

One of the shortcomings of the Global Thresholding detection mechanism is when *few* sybils rate *many* items each. We address this pitfall by designing a user monitor, which aims to detect this particular scenario. Figure 6.5(a) plots an example distribution of ratings input in a single window; we find that majority of the users input a low number of ratings per week, while a minority of outliers rate a high volume of movies. An attack in this context would thus entail setting a group of sybils to rate a high volume of content over a number of windows; detecting this behaviour focuses on examining *how many* high volume raters there are and *how much* these outliers are rating.

(a) How Much High Volume Raters Rate. Given the current mean value of ratings per user per window MU_t, we differentiate *high* from *low* volume raters based on the difference between the ratings



(a) Static Ratings Per User (b) Temporal Ratings Per User (c) Proportion High Volume Raters

Figure 6.5: Example Ratings Per User (1 Week), Proportion of Ratings Per High Volume Raters and High Volume Raters Over Time

that they have input in the current window and MU_t:

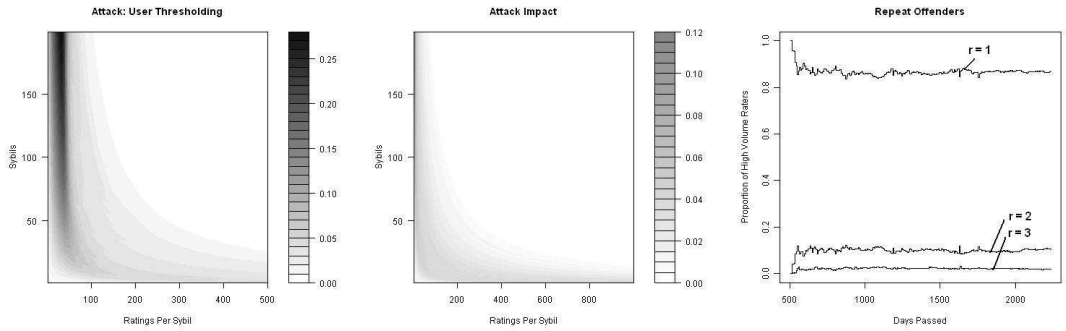
$$high(U_t, MU_t) = \begin{cases} \text{true}, & \text{if } U_t - MU_t > 0 \\ \text{false}, & \text{otherwise} \end{cases} \tag{6.9}$$

The mean ratings per high volume user, HM_t can then be monitored, in a similar way that we monitored the entire distribution in the previous section: an exponentially weighted moving average is regularly updated, and large deviations from the expected value flags an ongoing attack. In Figure 6.5(b) we plot the ratings per high volume user over time.

(b) How Many High Volume Raters. Given the high volume raters found with Equation 6.9, we also keep track of how many users HU_t there are relative to all the users who have rated in the current window. In other words, a user is suspect if they are at the highest end of the user-rating distribution, and both the *size* of this group and *volume* of ratings they input may indicate an ongoing attack. As we plot in Figure 6.5(c), the size of this group of users, divided by the total number of high volume raters per window, tends to be relatively stable; injecting different forms of attacks upsets both this and the mean ratings per high volume user values.

We take advantage of both pieces of information in order to amplify our detection mechanism: we create a *combined score* per window by multiplying the HM_t value by the proportion of suspect users HU_t. This way, we aim to capture fluctuations in both the *group size* and *rate* that a potential group of sybils will inflict when performing their attack.

We evaluated the user monitor with the Netflix subsets for cross-validated results with real data. We did so in two steps. First, Figure 6.6(a) shows the resulting impact if only part (a) of the above is used to defend the system: this defence can overcome similar scenarios that we addressed in the previous section or while lessening the threat of smaller groups of high-volume rating sybils. This threat is not fully eliminated: the top-left of the plot shows a remaining non-zero impact section. This is the effect of the false negatives of our monitor: sybils who rate at high volume but are not flagged. In Figure 6.6(b), we plot the impact of the *combined* defences. Overall, it reduces the impact of random attacks: Figure 6.6(a) reports attack impacts between approximately $[0, 0.25]$, while the combined defences range



(a) User Thresholding (b) Combined Threshold (c) Size of Repeat Offenders

Figure 6.6: User Monitor/Combined Impact Results, and Proportion of High Volume Raters Who Have Been In The Group for Varying Lengths of Time

between approximately $[0, 0.12]$. Nevertheless, the attacks that have the highest impact are now those where *many* sybils rate *few* items. In the next section, we consider the scenario where this type of attack now dominates.

Future Developments. One aspect that may aid the user monitor, but we leave as future work, is the *consistency* of membership to the high-volume raters group. In Figure 6.6(c) we plot the proportion of high-volume raters who have been in this group for $r \in \{1, 2, 3\}$ number of consecutive windows, after pruning the newcomers' first ratings. We find that over 80% of the high-volume raters are appearing in this group for the first time; as r is incremented, the relative group size falls sharply: at most 12% of the group members are making their second appearance, 3% are making their third. Sybils who are injecting a lot of noise for an extended period of time would become familiar faces in the high-rating group. Furthermore, the extent that honest users who rate large volume of movies per week input valuable data is questionable.

6.4.3 Item Monitoring

The last scenario that we address is the attacks that see *many* sybils rate *few* items each. This form of attack overcomes the previously outline defences: the sybils do not rate enough items each to be detected by the user monitor, and there are enough of them to not shift the rating per user temporal mean and flag their presence. To attempt to detect this kind of attack, we first reason on what items the group of sybils may be rating, and then design and evaluate an *item*-monitor to identify ongoing anomalous behaviour.

CF algorithms, that will be affected by injected profiles, operate on vectors of ratings. It thus seems intuitive that, in order to have the greatest impact possible, groups of sybils who inject very sparse profiles (by rating few items each) will tend to be rating a similar subgroup of items, rather than dispersing the ratings over a broad range of items, which would have a smaller effect. This strategy recalls the structure of *targetted* attacks [MBW07], where injected profiles contain *filler*, *selected*, and *target* item ratings. These profile regions correspond to the ratings that sybils must enter in order to construct an attack; for example, if an attack aims to promote a fantasy movie, the sybils may rate famous *selected* fantasy movies, along with a number of *filler* items to disguise each profile as a "normal" user profile. The

difference between a random and targetted attack is thus determined by the *strategy* of how to populate the profiles: what the *selected, filler,* and *target* items are (in the case of a random attack, there is no target item) and how they are rated; furthermore, the common subgroup of items that all sybils rate is the *selected* and *target* item. On a temporal scale, this form of attack would entail a large group of sybils rating items amongst this subgroup within a number of windows (proportional to the attack length). We therefore turn to monitoring the items in a system to detect these kinds of attacks. We further assume that it is very unlikely for an item that is *already* popular to be subject to an attack that aims to promote it; similarly, it is unlikely that unpopular items be demoted. In other words, we assume that the purpose of attackers is to maliciously reverse an ongoing trend (rather than reinforce a pre-existing one). Given this, we design an item monitor to identify the target of attacks by focusing on three factors: the *amount* that each item is being rated, the distance the *mean* of the incoming ratings for each item has from an "average" item mean, and a temporal mean *change detector*.

(a) The Item Is Rated By Many Users. At each time t, with R_t ratings input for I_t items, the average ratings per item MI_t (with standard deviation $\sigma_{i,t}$) can be computed. We can then select, from the available items, those that have been rated the most in the current window by selecting all those that received I_t ratings greater than the mean number of ratings per item MI_t:

$$high(I_t, MI_t) = \begin{cases} \text{true}, & \text{if } I_t > MI_t + (\alpha_t \times \sigma_{i,t}) \\ \text{false}, & \text{otherwise} \end{cases} \tag{6.10}$$

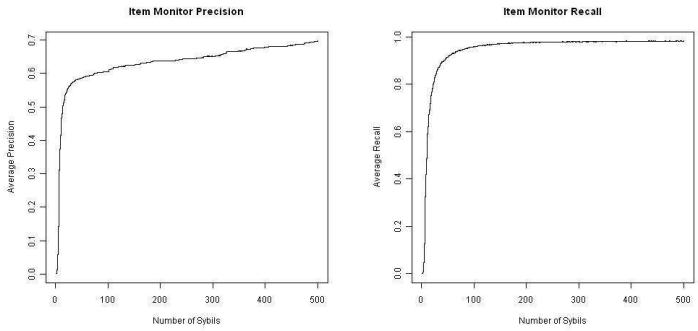
(b) The Item is Rated With Extreme Ratings. Using only the ratings input in the current window w, we determine the *mean* score $\bar{r}_i$ for each item i, and then average these to produce the expected mean score v per item:

$$v = \frac{1}{I_t} \sum_{i \in I_t} \bar{r}_i \tag{6.11}$$

If an item has been targetted for attack (and either demoted or promoted by a group of sybils simultaneously), then the corresponding $\bar{r}_i$ will reflect this by being an outlier of the global average item mean v.

(c) The Item Mean Rating Shifts. We compare the item mean computed with historical ratings and the $\bar{r}_i$ value determined from the ratings in the current window. A successful attack will shift this value by some distance δ: in this work, since we are operating on the Netflix 5-star ratings scale, we set δ to slightly below 2.

An attack is flagged for an item if the above three conditions are met: it is rated more than average, and the mean of the incoming ratings shows that it is both not being rated in the same way as other items are, and a change from the historical value is being introduced. Our monitor therefore focuses on identifying the moments when groups (or subgroups) of sybils rate the *target* item. We therefore modified our evaluation mechanism to test how well we find items when they are attacked, depending on how many sybils push in the target rating at the same time. We evaluated the monitor as follows: at time t, a group of sybils rates a randomly chosen target item. The sybils demote the item if it is popular (it has mean greater than 3), and promote it otherwise. We do not discriminate on the number of ratings



(a) Avg Precision (b) Avg Recall

Figure 6.7: Item Monitor: Average Precision & Recall

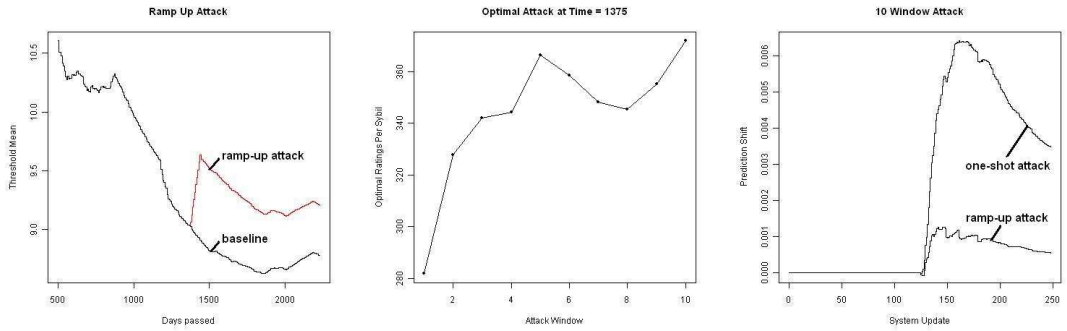
that movies currently have when determining whether to nuke or promote it; however, previous work shows that it is harder to protect sparsely rated items from attack [LR04], and our item selection process is biased toward selecting these items. We then check to see if the monitor flags any suspicious items, and measure the number of true/false positives and false negatives. We repeat the same run (i.e., group size and attack window) for 50 different items, and measure the resulting precision and recall. However, since an attack may begin in any of the available windows, we then repeat this process for each possible window, and average the results across time. Finally, we repeat this entire process with each Netflix subset to produce cross-validated results. The results therefore take into account the differences between sybil group size, target item, attack time, and honest user behaviour.

The average precision and recall values are plotted in Figures 6.7(a) and 6.7(b). They highlight that these methods work best when *many* sybils are rating the same item, with recall near 99% and precision near 70%. The fact that the precision is not performing as well as the recall implies that there are a higher proportion of false positives rather than false negatives: when an item is under attack, it is likely to be flagged as such, but few items that are not attacked may be flagged as well. As with the user monitor, it remains unclear how to deal with items that are being rated anomalously by users who are not the sybils that we explicitly control in our experiment. In fact, we can only be certain that users are malicious if we explicitly injected them: otherwise, we have assumed that the users in the dataset are honest and well-intentioned, which may not be the case. It is therefore preferable, in this case, to have a monitor with higher recall than precision, since we are sure that the sybils we inject are being found.

6.5 Adaptive Attack Models

The previous sections describe methods to detect different forms of automated sybil attacks by spotting anomalies in user behaviour. One of the natural limits of these techniques is when honest users' behaviour deviates from what was previously learned to be normal; a recent example is the vast numbers of web users who searched for news relating to the death of Michael Jackson [1] (a news article recommender system may thus see these articles being read in an anomalously high volume). On the other hand, attackers may modify their methods in order to overcome the defences. In this section, we switch to the

[1] http://tech.slashdot.org/story/09/06/29/003214/Google-Mistook-Jackson-Searches-For-Net-Attack



(a) Global Monitor (b) Optimal Ratings Per Sybil (c) Prediction Shift

Figure 6.8: Example Ramp-Up Attack: How it Affects the Monitor's Values, the Optimal Ratings Per Sybil and Prediction Shift

point of view of the attackers: we describe how attackers may overcome these defences, and evaluate the effect that such attacks have on the Netflix data.

6.5.1 The Ramp-Up Attack

In Section 6.4.1, we described a method of curbing random attacks by means of monitoring an exponentially weighted moving average of the ratings per user per window; the user- and item- monitors had a similar flavour. An attack was flagged if the current ratings per user value exceeded a threshold above the moving mean, determined by the historical values' variance and a weight α_g. The key insight of this detection mechanism is that an attack will *suddenly* and *noticeably* shift the moving mean, and can thus be detected. Similarly, our evaluation assumed that the *rate* at which sybils inject ratings remained constant.

Attackers may attempt to overcome this defence by *incrementally* changing the sybils' rate of attack. In the best case, the attacker would know what the current threshold is, and could set a group of sybils to inject at a rate that would push the mean up to (but not beyond) this value. Doing so would allow the attacker to then increase the rate in the following window; as the moving mean increases (along with the threshold), the attacker may be free to unleash evermore higher rates of profile injection. We call this a *ramp-up attack*.

The ramp-up attack can be used to defeat all three of the above defences; we experiment with this by considering the scenario of a system that only has the global monitor in place, and an attacker who would like to inject as much noise as possible into it during a period of 10 consecutive windows, starting roughly halfway through the dataset's timespan. Furthermore, we only consider the optimal case, where attackers know *a priori* what the incoming ratings per user per window values are, and can thus deduce what the threshold will be and tune sybils' rates accordingly. In doing so, we give attackers an advantage that they are unlikely to have; we discuss this further below.

In Figure 6.8(a) we plot the exponentially weighted moving average of the global threshold over time, both with and without a ramp-up attack. The effect of the ramp-up attack is to shift the mean, which then remains parallel to the original. Longer ramp-up attacks would shift the mean even further. Based

on this mean (and corresponding threshold), we computed the maximum ratings per sybil per window that avoids detection throughout the course of the ramp-up attack; we plot this sequence in Figure 6.8(b). An interesting point to note is that, since the mean relies on how both the honest and sybil users are behaving, the optimal values do not increase monotonically: there are windows where the honest users collectively rate less, which would thus highlight the sybils' misbehaviour unless their rate is reduced. Lastly, we measured the prediction shift when sybils inject ratings in this fashion, and compare it to the original attack (with the same duration) that we examined in Section 6.2. The results, in Figure 6.8(c), highlight the importance of our defences: forcing attackers to ramp-up rather than simply dump ratings into the system affects the prediction shift much less.

The results we show here are the best-case scenario for the attacker, since they knew the current threshold and ratings per user value. In practice, it is unlikely that the attackers know the current threshold, for a number of reasons:

1. The incoming ratings per user value (that we assumed that attackers knew) is computed at end of the current window. Attackers who may monitor all users to learn the precise value would then have no time to inject ratings.

2. The current threshold varies as the exponentially weighted moving average is updated; even if attackers knew the previous window's threshold there is no guarantee that the attacker can inject the maximum number of undetectable ratings in the current window.

3. Experimenting, in order to discover the threshold, would be difficult since, as we saw in Figure 6.8(b), avoiding detection in one window does not guarantee that the same rate will avoid detection in the next.

4. Furthermore, attempting to discover the threshold will (a) impact the threshold itself, since α is updated, and (b) reveal a set of sybils once the threshold has been surpassed, requiring attackers to then restart their efforts from scratch.

Similar ramp-up attacks may be performed to overcome the defences in Sections 6.4.2 and 6.4.3. However, the main insight from the above experiments is that ramp-up attacks are more difficult and require more time to execute than attacks on an unmonitored system. This therefore highlights that, while the temporal monitors that we describe above are not infallible, they provide a significantly difficult obstacle that attackers now need to overcome.

6.6 Discussion & Related Work

Anomaly detection algorithms have been used when securing a wide range of systems, in order to, for example, defend against financial fraud [WB08] or defend web servers from denial of service attacks [SP06]. These techniques are readily applicable to recommender systems; the only problem being how to define what an anomaly is, and how to monitor the large volume of users and items. In this chapter, we have introduced novel methods that detect anomalies in various aspects of rating behaviour while learning what normal user behaviour is, thus liberating system administrators from these challenges.

To do so, we leveraged the effect that honest users have on the temporal dynamics of the system. For example, we used the fact that a majority of users rate very few items in order to identify the sybils who are rating a lot. The only way that sybils may dodge pushing the monitored variables over the detection thresholds is by *not rating*: our defences acts as an incentive for attackers to draw out the length of their attack, thus reducing its overall effect (as seen in Section 6.2).

The monitors that we described above each address different forms of attack: the global monitor detects large groups with high rates of profile injection, the user monitor detects the few users who are injecting large volumes of ratings, and the item monitor detects when many users are rating a target item in an anomalous fashion. We thus evaluated each one separately in order to highlight each monitor's strengths and weaknesses. However, we already saw that more than one monitor may flag the same attack; for example, the user monitor detected many of the same attacks as did the global monitor. In the future, we plan to evaluate how multiple defences operate when combined, and the overlap between user, item, and global behaviour as different attacks are taking place.

Anomaly detection has also been seen before in recommender system research. Bhaumik *et al*. [BWMB06] propose a method to monitor *items* as they are under attack, by looking at changes to an item's mean rating over time. Similarly, Yang *at al* [YSKY09] infer user trust values based on modeling the signal of incoming ratings. They use these techniques to monitor when *real users*, who each control 50 sybils, are attacking a system. To that extent, their system is under a variety of potentially conflicting attacks. Our work differs on two main points: first, we evaluate a system that iteratively updates and computes personalised recommendations for each user. We also propose methods that assume a large set of users and items, and flag attacks while monitoring all users and items (rather than simply monitoring users/items individually). We evaluate attacks that may not demonstrate anomalies within a single time window, but appear between system updates, and may be targetted to affect particular users' recommendations. We also explore a wide variety of attacks, ranging from the *random* to *targetted* scenarios, where a key aspect of the attacks is the fact that sybil *groups* of varying size are rating items. There are a number of other particular strategies that attackers may adopt (such as the bandwagon or average attacks strategies [MBW07]) when unleashing a set of sybils that we have not explored above. Our detection mechanisms, in focusing on complementary dimensions of attacks (the *group size* and *rate* of sybils as they attack) hope to detect attacks regardless of the adopted strategy.

The idea of temporality in attacks has also been explored from the point of view of user reputation; Resnick and Sami [RS07] prove a variety of properties of their reputation system, which takes into account the order in which ratings are input. It remains unclear how these systems would work in practice: many reputation or trust-based systems assume that the ratings input by users are the ground truth, without taking into account that users are both naturally inconsistent when they rate [APTO09] and what they rate will be influenced by what they are recommended. Furthermore, one of the most troubling problems that both monitoring techniques and reputation systems suffer from is *bootstrapping*; systems can be easily abused when the variables that monitor or reflect user behaviour have had little to no data. We use all ratings input prior to a pre-specified time ϵ to bootstrap each monitor. System administrators

may opt to ask a controlled, closed group of trusted users to rate for varying lengths of time in order to bootstrap [ALP+09]. Alternatively, if the system also offers social networking functionality, defences that identify sybils according to their location on the social graph can be applied [DC08]; in this work we assumed that no such graph was present.

While it is often the case that designing security mechanisms is context-sensitive, there are lessons that we learn here that may be applicable to other scenarios. For example, Yu *et al*. [YSK+09] design a method for recommender systems with binary-voting like Digg, and demonstrate the ability to fend off attacks where the number of sybils greatly exceeds the number of honest users. While our work focuses on the ordinal-scale rating-based Netflix data, the similarity between the two contexts is the need for sybils to outweigh honest user behaviour in order to achieve malicious goals, and doing so in tandem is a key insight into detecting their misbehaviour.

6.7 Summary

In this chapter, we have confronted the problem of sybil attacks to recommender systems. The focal point of the contributions we make here is that sybils are detectable not only via *what* they rate (as state of the art sybil classifiers learn from) but also by *how* they insert these malicious ratings. In fact, the mere act of casting the problem of recommender system robustness onto a temporal scale already makes it harder for attackers to meet their goals: they can no longer simply dump ratings into a system and expect these to have any effect. Furthermore, the actions of the system's honest users can be leveraged in order to identify the automated attacks. We introduced a windowed-view of temporal behaviour, defined the notion of temporal attacks, and then designed and evaluated a *global, user,* and *item* monitor that flags when different forms of attack are taking place: where sybil groups (of varying size) inject item ratings (at varying rates) over time in order to either disrupt the system's recommendations (via a *random* attack) or modify the recommendations of a particular item (with a *targetted* attack).

Lastly, we paved the way for future research by describing how attackers may overcome these defences; namely, by performing a *ramp-up* attack that can fool the defences into believing that no attack is taking place. We compared the effects of a ramp-up attack, when the system is defended by our algorithms, and a one-shot attack on a system with no defences, and concluded that the ramp-up attack is not as immediately effective as an attack on an undefended system: our methods thus increase the time and effort that attackers require to accomplish their goals. Future work should thus investigate how recommender systems can identify an ongoing ramp-up attack and adapt the system's defences accordingly.