Temporal Defences for Robust Recommendations

Neal Lathia, Stephen Hailes, and Licia Capra

Department of Computer Science,
University College London
London, UK WC1 E 6BT
{n.lathia | s.hailes | l.capra}@cs.ucl.ac.uk

Abstract. 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 Collaborative Filtering (CF) algorithms. To protect against these attacks, previous work focuses on designing sybil profile classification algorithms, whose aim is to find and isolate sybils. These methods, however, assume that the full sybil profiles have already been input to the system. Deployed recommender systems, on the other hand, operate *over time*, and recommendations may be damaged while sybils are still injecting their profiles, rather than only after all malicious ratings have been input. Furthermore, system administrators do not know when their system is under attack, and thus when to run these classification techniques, thus risking to leave their recommender system vulnerable to attacks. In this work, 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.

1 Introduction

Recommender systems based on Collaborative Filtering (CF) algorithms [1] have become important portals via which users access, browse, and interact with a plethora of web sites, ranging from e-commerce, to movie rentals, and music recommendation sites. These algorithms are built upon the assumption that users who have been like-minded in the past can provide insight into each other's future tastes. Like-mindedness is quantified by means of similarity metrics (e.g., Pearson correlation) computed over users' profiles: the more items two users have rated in common, and the more similar the ratings they have associated to these items, the more like-minded they are considered. Unfortunately, the ratings that CF algorithms manipulate may not be honest depictions of user preferences, as they may have been fabricated by malicious users to damage or modify the recommendations the system outputs. Abusing recommender systems this way is referred to as *shilling*, *profile injection* or *sybil* attacks; Mobasher *et al.* provide an in-depth review of the problem [2].

To protect a recommender system from these attacks, a variety of classification algorithms [4] have been proposed, whose goal is to examine users' profiles, determine whether they are honest or malicious, and isolate the latter. However, these approaches assume that the *full sybil profiles* have already been created (i.e., contained within the user-item matrix that CF algorithms operate on). In deployed recommender systems, such an assumption does not hold: sybil profiles 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. Furthermore, system administrators do not know when their system is under attack, so deciding when to run these computationally expensive classification algorithms becomes a fundamental issue.

In this work, we first demonstrate that attackers do have an incentive to spread their attack over time, as this makes it difficult for system administrators to know when to run classification algorithms and thus isolate them (Section 2). We provide a classification of *temporal sybil attacks* against which a deployed recommender system should defend itself (Section 3), and then propose and evaluate our *sybil detection technique*, which dynamically monitors the deployed recommender system to reveal anomalies in global, user, and item activity, with respect to normal user rating habits (Section 4). We then conclude the paper with a discussion of related work (Section 5) and future directions of research (Section 6).

2 From One-Shot to Temporal Attacks

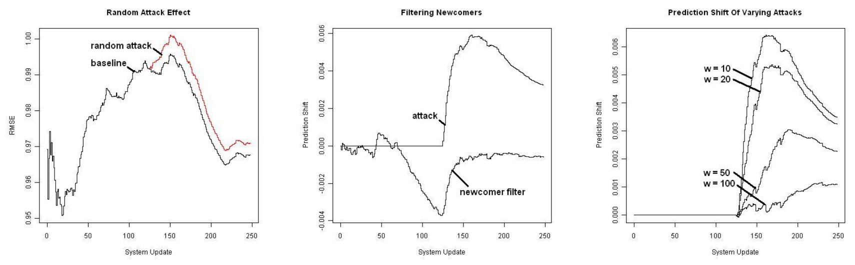
In this section, we show that sybils have an incentive to spread their attack over time, as in so doing they reduce the risk of being detected and thus isolated. To begin with, we model a deployed recommender system as done in [5]: given a dataset at time t, and a window size $|w|$ (reflecting how often the system will be updated), we train the CF algorithm with any data input prior to t and predict any ratings input between t and $(t+|w|)$. The entire process is repeated at each update, with what was previously tested on becoming incorporated into the training data. To quantify the performance of the recommender system over time, we re-define the root mean squared error (RMSE) as follow:

$$RMSE_t = \sqrt{\frac{\sum_{\hat{r}_{u,i} \in R_t}^{N} (\hat{r}_{u,i} - r_{u,i})^2}{|R_t|}} \tag{1}$$

Given a set R_t of predictions made up to time t, the current error is simply the average of all predictions made to date.

The one-shot attack considered in the literature operates as follow: if we indicate with S the set of sybils in the system, and with X the set of ratings they inject, then all X would be input in the system within a single time window, that is, between time t and $(t+|w|)$. We visualise the effect of a one-shot attack with the following example: we consider five sub-samples of $10,000$ Netflix profiles[1], and weekly system updates ($|w| = 7$ days); for each of these sub-samples, during

[1] http://www.netflixprize.com



(a) Random Attack (b) Filtering Newcomers (c) Different Attack Lengths

Fig. 1. Time-Averaged RMSE Of One-Shot Attack; Prediction Shift When Pruning Newcomer's Ratings; Injecting Attacks Over Varying Time Windows

the 125*th* week-long window we inserted 100 sybils who each rate approximately 10,000 items (in this example, we limit ourselves to exploring the temporal effect of a *random* attack, where each sybil randomly picks one of the available items, and then rates it with a random value drawn uniformly from the rating scale). Figure 1(a) plots the impact that these ratings have on the time-averaged RMSE. As shown, the one-shot attack has a pronounced effect on the time-averaged RMSE: performance is consistently degraded over the rest of the updates.

However, this attack is very simple to detect: sybils appear all at once, rate high volumes of items within a single window length, and disappear. CF systems can easily defend against these attacks by *distrusting newcomers*: ratings coming from new users are considered *suspect* and excluded by the CF algorithm; if these users re-appear in future time windows, their ratings will be subsequently considered, otherwise they will never influence the recommender systems. We thus repeated the previous experiment, but excluded suspect ratings from the kNN CF algorithm. Note that, by excluding suspect ratings, we maintain our ability to formulate recommendations for all users (including sybil and new honest users), while removing the influence that suspect ratings exert. Figure 1(b) plots the difference in prediction (*prediction shift*) when exercising the same one-shot attack described above, with and without new ratings being suspected, against a baseline scenario of predictions computed when no sybils are inserted. Note that 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 (the prediction shift is negative). Removing the ratings of users who rate and never return thus curbs these attacks and takes small steps towards de-noising the data [6].

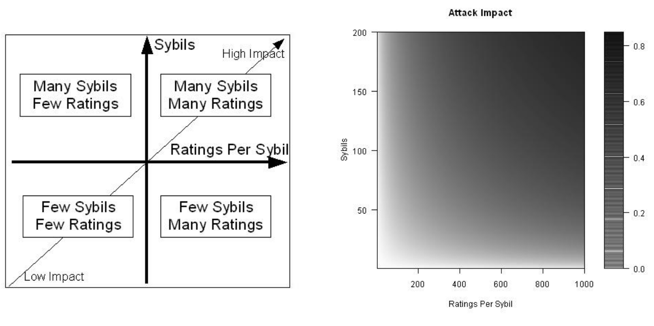
Attackers may respond to this simple detection technique, by widening the number of windows taken to inject ratings. Sybils under the attacker's control would therefore appear in multiple windows and, after the first appearance, no longer be suspect. In order to explore the incentives that attackers have to rate quickly, 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 $W \in [10, 20, 50, 100]$. 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 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.

Based on the above experiments, real sybils' behaviour is unlikely to follow the one shot pattern. Rather, we have observed that: (a) there is an incentive for attackers to inject ratings over more than one window, in order to not have their ratings be suspect and filtered out by simple detection schemes; and (b) attackers may attempt to displace the balance between sybil and non-sybil ratings, since higher volumes of sybil ratings per window have more pronounced effects. With this in mind, we provide next a classification of the types of temporal attacks that malicious users may undertake, before proposing a defence mechanisms capable of protecting a recommender system against them.

3 Temporal Attack Model

When enacting a sybil attack, there are two main factors that attackers can control: the *number of sybils* perpetrating the attack, and the *rate* at which they operate (i.e., number of ratings per sybil per window), for a predefined sequence of windows. We thus classify attacks according to how malicious users calibrate these two factors, and determine the four different attack types depicted in Figure 2(a). Protection mechanisms developed so far in the literature ignore these two factors and rather focus on another one: the *strategy* that attackers deploy in determining what items to rate (i.e, whether at random or targeted), as if the ratings were input all at once. We take a different stance and look at the attack while it unfolds: the strategy adopted is then treated as an orthogonal dimension, with respect to the temporal factors.

Our goal is to provide a monitoring and detection mechanism capable of alerting the system administrator when any of these attacks is in place. In order to assess the quality of our monitor, we will be measuring: *precision*, defined as the number of attacks that were flagged (true positives) over all flagged situations (true positives plus false positives); *recall*, defined as the number of attacks that were flagged, over all the attacks enacted (true positives plus false negatives); and *impact*, defined as the number of sybil ratings input at the current window, divided by the total number of ratings input in that window. Intuitively, the higher the precision, the lower the false alarms being raised; furthermore, the higher the recall, the lower the number of attacks that slip through. Finally, the impact gives an indication of the damage that an undetected attack (false negative) is causing; intuitively, the higher the number of malicious ratings that slip through, the higher the impact. Figure 2(b) illustrates the attack impact



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

Fig. 2. Attack Types and Impact With No Defences

for varying numbers of sybils and rating rates, when no defences are in place (and thus all attacks pass undetected). In this work, we place higher emphasis on reducing false negatives (finding all the attacks that we manually insert), rather than false positives. This is because we cannot know (and only assume) that the data we experiment with is the fruit of honest, well-intentioned users: false positives in the real data may very well be attacks that are likely to deserve further inspection; however, we note that the defences described below produced no false positives when run on the rating data with no attacks injected.

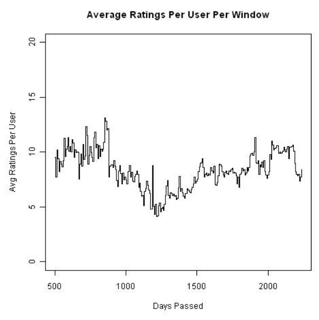
In the next section, we construct step by step a defence mechanism capable of defending a recommender system against the temporal attacks identified above. We do so by reasoning about factors that attackers cannot control, related to how the non-sybil users behave: how many non-sybil users there are, the number of ratings they input per window, what they rate, and how they rate.

4 A Temporal Defence

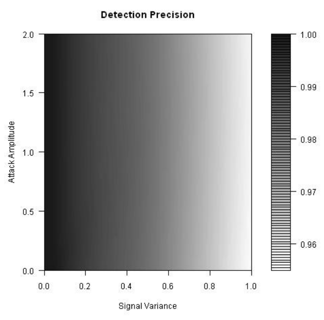
In this section, we focus on each type of attack in turn, and construct a comprehensive mechanism capable of detecting them all. The mechanism monitors different types of information to detect anomalies: global behaviour (Section 4.1), user behaviour (Section 4.2), and item behaviour (Section 4.3). Key to our proposal is the capturing of stable features in the rating data, so that anomalies introduced by attacks can be flagged.

4.1 Global Monitoring - Many Sybils/Many Ratings Scenario

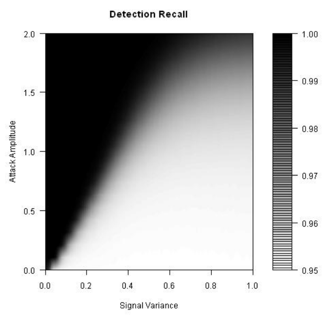
The first perspective of system behaviour that we consider is at the *global*, or aggregate, level. While the number of ratings that users input varies over time, the average ratings per user per window (in the Netflix data) remains relatively flat: Figure 3(a) plots this value over time. From this, we see that the average user rates between $5 - 15$ movies per week. Since the mean is derived from



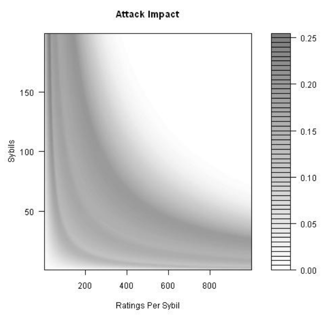
(a) Avg Ratings Per User Per Week in Netflix



(b) Simulation Precision



(c) Simulation Recall



(d) Global Monitoring - Impact in Netflix

Fig. 3. Global Monitoring

a long-tailed distribution, it is a skewed representation of the "average" user. However, when an attacker deploys a large group of sybils, this aggregate value changes: 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, and a weighting factor β_t, 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 $\alpha_t \geq 1$:

$$\frac{R_t}{U_t} \geq (MU_t + (\alpha_t \times \sigma_t)) \tag{2}$$

If an alarm is not raised, we update the current MU_t value as an exponentially weighted moving average:

$$MU_t = (\beta_t \times MU_{t-|w|}) + ((1-\beta_t) \times \frac{R_t}{U_t}) \tag{3}$$

MU_t is updated *conservatively*: if an attack is flagged, then it is not updated. We also conservatively 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 will disperse temporal trends. We therefore update β_t with the standard deviation measured to date:

$$\beta_{t+|w|} = \max(|\sigma_{t-|w|} - \sigma_t|, 1) \tag{4}$$

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 a 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, making it more difficult for attackers to learn the appropriate threshold. In this work, α_t jumps between pre-specified values (0.5 and 1.5). This parameter is sensitive to the context; we selected these values after examining our scenario (users rating movies). We leave a more in-depth investigation of scaling the threshold as a matter of future work.

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: 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.

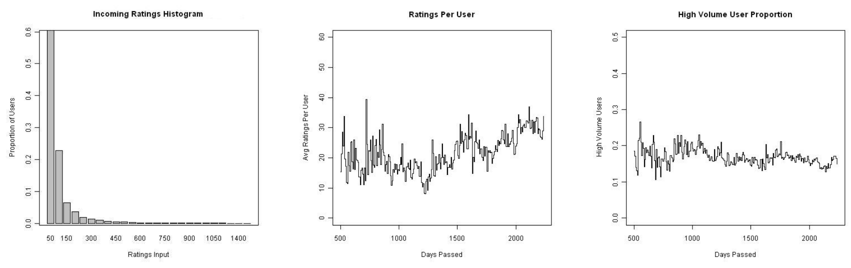
In order to simulate a stream of incoming ratings, we draw a sequence of $\frac{R_t}{U_t}$ values from a normal distribution with mean μ and standard deviation $\sigma \in [0, \mu]$. Then, at random moments, we simulate an injected attack where a group of sybils shifts the incoming value by the attack *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 of sequential attack windows (we used a value of 50, as this gives the attack high impact while being difficult to detect - see Figure 1(c)). We re-ran each simulation parameter setting $10,000$ times and present averaged results. Figure 3(b) shows the resulting precision, which fades as σ increases. The main point to note is that precision is dependent on σ (the variability in the ratings per user per window) rather than the attack amplitude γ. In other words, the number of false positives depends on how naturally variable the data is. Figure 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. Note that, in this simulated setting, the minimum precision is slightly below 0.90, and the minimum recall remains above approximately 0.95: we thus consistently get high precision and recall.

We returned to the Netflix dataset in order to test this method when injecting attacks on 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 3(d) 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 sybil's rate. In Section 4.2 we address the former, while Section 4.3 describes how to detect the latter.

4.2 User Monitoring - Few Sybils/Many Ratings Scenario

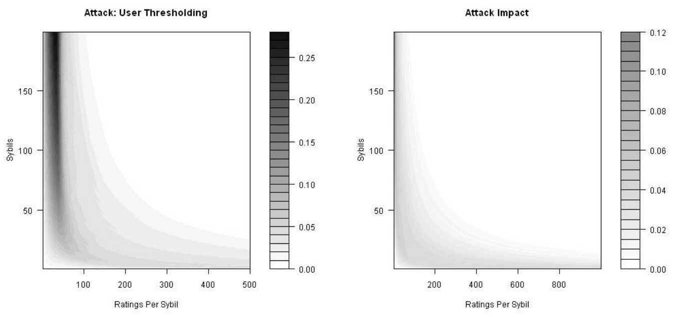
The previous monitor cannot detect when *few* sybils rate *many* items each. We address this by designing a user monitor aimed at detecting these specific attacks. Figure 4(a) plots the distribution of ratings input per user in a sample window of the Netflix data; we find that the majority of users input a rather 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 small 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 that they have input in the current window and MU_t. Figure 4(b) plots the ratings per high volume user over time. The mean ratings per high volume user HM_t can then be monitored, in a similar way as we monitored the entire distribution in the previous section: an



(a) Example Ratings Per User (1 Week) (b) Ratings Per High Volume Raters Over Time (c) Proportion of High Volume Raters Over Time

Fig. 4. Analysis of Users' Behaviour in Netflix



(a) Single Thresholding (b) Combined Threshold

Fig. 5. User Monitoring

exponentially weighted moving average is regularly updated, and large deviations from the expected value flag an ongoing attack.

(b) How Many High Volume Raters. Beside the high volume ratings HM_t, we also keep track of how many users HU_t there are relative to all the users who have rated in the current window. A user becomes 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 4(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 5(a) shows the resulting impact if only part (a) of above is used to defend the system: this defence can overcome both scenarios similar to that addressed in the previous section, and also lessen the threat of smaller groups of high-volume rating sybils. This threat is not fully eliminated though: the top-left corner of the plot shows a remaining non-zero impact section (more precisely, impact is approximately $[0, 0.25]$). In Figure 5(b), we plot the impact of the *combined* defences: this time, the impact decreases to $[0, 0.12]$. There is now only one type of attack left, where *many* sybils rate *few* items. We tackle this scenario next.

4.3 Item Monitoring - Many Sybils/Few Ratings Scenario

The last scenario that we address is that of *many* sybils rating *few* items each. This form of attack would be undetected by the previously outlined 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 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.

In order to have the greatest impact possible, 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 *targeted* attacks [2], where injected profiles contain *filler*, *selected*, and *target* item ratings: for example, if an attack aims to promote a fantasy movie (target), the sybils will rate it, alongside famous fantasy movies (selected) that are likely to appear in the profiles of many honest users, together with a number of items (filler) to disguise each profile as a "normal" user profile. The difference between a random attack and a targeted one is thus determined by how profiles are populated: 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. 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 nuked. 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: (a) the *amount* that each item is being rated, (b) the distance the *mean* of the incoming ratings for each item has from an "average" item mean, and (c) a temporal mean *change detector*.

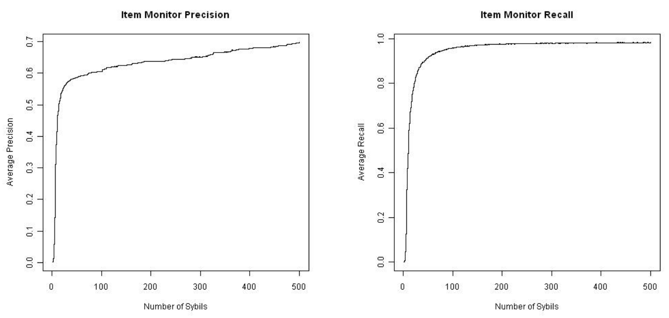
(a) The Item Is Rated By Many Users. In each window w, with R_t ratings input for I_t items, the average ratings per item MI_t (with standard deviation $\sigma_{i,t}$) can be computed. We can then select, from the available items, those that have been rated the most in the current window by selecting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

(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:

$$v = \frac{1}{I_t} \sum_{i \in I_t} \bar{r}_i \tag{5}$$

If an item has been targeted for attack (and either nuked 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.



(a) Avg Precision (b) Avg Recall

Fig. 6. Item Monitoring

An attack is flagged for an item if the above three conditions are met: it is rated more than average, the mean of the incoming ratings shows that it is 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 nuke 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 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 [7], 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(a) and 6(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 as to how to deal with items that are being rated anomalously by users who are not the sybils that we explicitly control in our experiments; 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.

5 Related Work

Anomaly detection algorithms have been used when securing a wide range of systems: they defend against financial fraud [8] and protect web servers from denial of service attacks [9]. These techniques are applicable to recommender systems too, the main problem being how to define what an anomaly is, and how to monitor the large volume of users and items. In this work, 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 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 thus act as an incentive for attackers to draw out the length of their attack, thus reducing its overall effect (as seen in Section 2).

Anomaly detection has also been seen before in recommender system research. Bhaumik *et al.* [10] 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* [11] infer user trust values based on modelling 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 targeted to affect particular users' recommendations. We also explore a wide variety of attacks, ranging from the *random* to *targeted* scenarios, where a key aspect of the attacks is the fact that sybil groups of varying size are rating items.

The idea of temporality in attacks has also been explored from the point of view of user reputation; Resnick and Sami [12] prove a variety of properties of their reputation system, which takes into account the order that 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 [6] 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. In this work, 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. Alternatively, if the system also offers social networking functionality, defences that identify sybils according to their location on the social graph can be applied [3]; in this work we assumed that no such graph was present.

6 Conclusion and Future Work

In this work, we have confronted the problem of sybil attacks to deployed recommender systems, 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 *targeted* attack). We introduced a windowed-view of temporal behaviour, defined a classification of temporal attacks, and then designed and evaluated a *global, user,* and *item* monitoring mechanism that flags when different forms of attack are taking place. Our work centred on the Netflix dataset: we captured a variety of features of this data that remain stable over time and are noticeably affected by a sybil attack. There are a number of other strategies that attackers may adopt, such as the bandwagon or average attacks strategies [2] when unleashing a set of sybils that we have not explored above. Our detection mechanism, in focusing on complimentary dimensions of attacks (the *group size* and *rate* of sybils as they attack), aims to detect attacks regardless of the adopted strategy.

Our ongoing and future work spans many directions: we have started broadening the range of datasets that we apply these defences to, in order to see how varying contexts (i.e., rating movies, music, places) change the stable factors that we take advantage of. We are also conducting experiments in less homogeneous settings, where different types of attacks are taking place simultaneously, to assess the precision, recall and impact of our monitors when combined. In this work, we assumed that the rate at which profiles are populated is roughly similar across sybils and constant in time; our future work aims to remove this assumption, thus addressing the case of attackers that *incrementally* change the rate of attack, to avoid exceeding the current thresholds and thus pass undetected. Note though that it is extremely difficult for attackers to know the values of current thresholds, as they vary with the updating of the exponentially weighted moving averages; experimenting, in order to discover the thresholds, would be difficult since avoiding detection in one window does not guarantee that the same rate will avoid detection in the next.

References

1. G. Adomavicius and A. Tuzhilin. Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE TKDE*, 17(6), June 2005.
2. B. Mobasher, R. Burke, R. Bhaumik, and C. Williams. Toward Trustworthy Recommender Systems: An Analysis of Attack Models and Algorithm Robustness. In *ACM TOIT*, 2007.
3. H. Yu, M. Kaminsky, P. Gibbons, and A. Flaxman. SybilGuard: Defending Against Sybil Attacks Via Social Networks. In *ACM SIGCOMM*, volume 4, pages 267–278, Pisa, Italy, 2006.
4. C. Williams, B. Mobasher, and R. Burke. Defending Recommender Systems: Detection of Profile Injection Attacks. *Journal of Service Oriented Computing and Applications.*, August 2009.
5. N. Lathia, S. Hailes, and L. Capra. Temporal Collaborative Filtering With Adaptive Neighbourhoods. In *ACM SIGIR*, Boston, USA, 2009.
6. X. Amatriain, J.M. Pujol, N. Tintarev, and N. Oliver. Rate it Again: Increasing Recommendation Accuracy by User Re-Rating. In *ACM RecSys*, New York, USA, 2009.
7. S. K. Lam and J. Riedl. Shilling Recommender Systems for Fun and Profit. In *Proceedings the 13th International Conference on World Wide Web*, New York, USA, 2004.
8. S. X. Wu and W. Banzhaf. Combatting Financial Fraud: A Coevolutionary Anomaly Detection Approach. In *10th Annual Conference on Genetic and Evolutionary Computation*, pages 1673–1680, Atlanta, GA, USA, 2008.
9. V. A. Siris and F. Papagalou. Application of Anomaly Detection Algorithms for Detecting SYN Flooding Attacks. *Computer Communications*, 29:1433–1442, May 2006.
10. R. Bhaumik, C. Williams, B. Mobasher, and R. Burke. Securing Collaborative Filtering Against Malicious Attacks Through Anomaly Detection. In *Proceedings of the 4th Workshop on Intelligent Techniques for Web Personalization (ITWP'06)*, Boston, July 2006.
11. Y. Yang, Y. Sun, S. Kay, and Q. Yang. Defending Online Reputation Systems against Collaborative Unfair Raters through Signal Modeling and Trust. In *Proceedings of ACM SAC TRECK*, 2009.
12. P. Resnick and R. Sami. The Influence Limiter: Provably Manipulation Resistant Recommender Systems. In *Proceedings of Recommender Systems (RecSys '07)*, Minneapolis, USA, 2007.