A Survey of Attack-Resistant Collaborative Filtering Algorithms

Bhaskar Mehta Thomas Hofmann
Google Inc.
Brandschenkestrasse 110
Zurich, Switzerland - 8004
{bmehta,thofmann}@google.com

Abstract

With the increasing popularity of recommender systems in commercial services, the quality of recommendations has increasingly become an important to study, much like the quality of search results from search engines. While some users faithfully express their true opinion, many provide noisy or incorrect ratings which can be detrimental to the quality of the generated recommendations. The presence of noise can violate modeling assumptions and may thus result in unstable estimates or predictions. Even worse, malicious users can deliberately insert attack profiles in an attempt to bias the recommender system to their benefit. This is a particularly important issue, and it is necessary for systems to provide guarantees on the robustness of recommendations to ensure continued user trust. While previous research has attempted to study the robustness of various existing Collaborative Filtering (CF) approaches, the explicit design of robust recommender systems remains a challenging problem. Approaches such as Intelligent Neighborhood Selection, Association Rules and Robust Matrix Factorization are generally known to produce unsatisfactory results. In this paper, we review previous approaches to robust collaborative filtering; we also describe promising recent approaches that exploit a Singular Value Decomposition (SVD) and are both accurate as well as highly stable to shilling.

1 Introduction

Collaborative filtering technology is being widely used on the web as an approach to information filtering and recommendation by commercial service providers like *Amazon* and *Yahoo!*. For multimedia data like music and video, where pure content-based recommendations perform poorly, collaborative filtering is the most viable and effective solution, and is heavily used by providers like *YouTube* and *Yahoo!* Launchcast. For malicious attackers, or a group interested in popularizing their product, there is an incentive in biasing the collaborative filtering technology to their advantage. Such attacks have been refered to as *shilling* attacks, and attackers as *shillers*. Since user profiles of shillers looks very similar to an authetic user, it is a difficult task to correctly identify shilling attacks. Early algorithms exploited signatures of attack profiles and were moderately accurate. In particular, by looking at individual users and mostly ignoring the combined effect of such malicious users, these detection algorithms suffred from low accuracy in detecting shilling profiles. Recent approaches based on SVD (cf. [8]) have proved to be much more accurate, exploiting the *group effect*, i.e. eliminating groups

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

of attackers which seem to work together. However, employing such detection approaches as a preprocessing step are computationally expensive, and essentially not an online process. The next logical step is to build in detection into the recommendation algorithm itself; this work provides a survey of such robust collaborative filtering algorithms proposed in the past and until recently.

1.1 Shilling attacks on Collaborative Filtering

Collaborative Filtering systems are essentially social systems which base their recommendation on the judgment of a large number of people. Like other social systems, they are also vulnerable to manipulation by malicious social elements. As an example, a loosely organized group managed to trick the Amazon recommender into correlate the book *Six Steps to a Spiritual Life* (written by the evangelist Pat Robertson) with a book for gay men[1].

A lot of web-enabled systems provide free access to users via a simple registration process. This can be exploited by attackers to create multiple identities for the *same* system and insert ratings in a manner that affect the robustness of a system or algorithm, as has been studied in recent work [6]. *Shilling attacks* add a few user profiles which need to be identified and protected against. Shilling attacks can be classified into two basic categories: inserting malicious profiles which rate a particular item highly are called *push* attacks, while inserting malicious profiles aimed at downgrading the popularity of an item are called *nuke* attacks. Various attack strategies were then invented; these include [3]:

1. *Random attacks*, where a subset of items is rated randomly around the overall mean vote.
2. *Average attacks*, where a subset of items is rated randomly around the mean vote of every item
3. *Bandwagon attacks*, where a subset of items is rated randomly around the overall mean, and some popular items are rated with the maximum vote.

Random and Bandwagon attacks are low-knowledge attacks requiring information only about some popular items and overall vote statistics. Average attacks require more information and have been shown to be near optimal [7] in impact. They have also been observedly difficult to detect [15].

The strength of shilling attacks is specified using two metrics: *filler size* and *attack size*. Filler size is the set of items which are voted for in the attacker profile, usually measured in %. Attack size refers to the number of shilling profiles inserted into user data. The impact of the attacks is measured by the increase in the number of users to whom an attacked item is recommended. Generally, average attacks are strogner than random or bandwagon attacks.

Metrics for Collaborative Filtering and Shilling The task of evaluating predictions in collaborative filtering is easily described as the measurement of the *deviation* from observed values; accuracy of a CF algorithm is measured over some held-out data from the training dataset. The effect of an attack is measured by the deviation in predited ratings before and after attack profiles have been added. *Prediction Shift* measures the average change in prediction of the attacked item (before and after attack) of a CF algorithm. This metric is also sensitive to the strength of an attack, with stronger attacks causing a larger prediction shift.

Hit Ratio measures the effect of attack profiles on top-k recommendations. Since the end effect of a recommender system is a list of items recommended to a particular user, this metric captures the fraction of users affected by shilling attacks. Let $H_{u,i} = 1$ if an item i is a top-k recommendation to user u, and $H_{u,i} = 0$ otherwise. Hit ratio is a fraction between 0–1; when expressed as a percentage (%), it is defined as follows:

$$\mathcal{H} = \frac{100}{N} \times \sum_u \Delta H_{u,i}, \quad \text{s.t. } N = \text{\# users} \tag{1}$$

[1]Story at `http://news.com.com/2100-1023-976435.html`.

Finally, *Precision and recall* are standard IR measures which are also applicable widely; various detection aaproaches report precision and recall for various siyes of attacks. Clearly, high precision is an indication of an accurate method.

2 Robust Collaborative Filtering using Intelligent Neighbor Selection

The earliest algorithms for collaborative filtering relied on neighborhood formation based on user similarity; these are known as k-nearest neighbor (kNN) CF algorithms. These algorithms remain extremely popular due to their simplicity and intuitiveness. This family of algorithms use a weighted reconstruction of the votes of users similar to the current user as a prediction for the rating for a previously unrated item. Various improvements have been made to the basic mechanism of predicting votes using Pearson's correlation, but they mostly comply to the following scheme: assume the user database consists of a set of votes $v_{i,j}$ corresponding to the vote for user i on item j. The predicted vote for an active user for item j, $p_{a,j}$ is a weighted sum of the votes of other users:

$$p_{a,j} = \overline{v}_a + \kappa \sum_{i=1}^{n} w(a,i)(v_{i,j} - \overline{v}_i), \quad \text{where } w(a,i) \text{ is a similarity function} \tag{2}$$

As explained in Section 1.1, attacks in recommender systems can be achieved by the addition of malicious profiles. By special construction, these profiles can be made to look extremely similar to influential users. The impact is that such malicious users can be wrongly identified as *neighbors* for normal users, thus influencing the results of the recommendation algorithm. O'Mahony et al.[12] first discussed *neighbourhood filtering*, where the key idea is to prevent suspicious users from influencing other users. The strategy for selecting useful neighbors takes into account the *reputation* of all users participating in the recommendation process. The reputation measurement strategy is adapted from literature and requires the computation of a reputation score for a particular user who is providing ratings on an item y_i. The first step is calculating the reputation of users who form a neighborhood for other users that have rated y_i; the second step is to filter the neighborhood and remove any possible malicious ratings. The observation here is that if there is indeed an attack, there would be a marked difference between real users and malicious ones, thus leading to two clusters; thus clustering is performed to detect if such a pattern is observed for a particular item. Given that an attack may be trying to promote an item, or demote it, its essential to know the *direction* of the shift for this filtering strategy to work. The approach employed by [12] overcomes this by thresholding the difference in the mean values of the two clusters; if this is above a threshold (evaluated empirically), an attacked is supposed to have occured. To detect which cluster contains the attack users, the one with the lower standard deviation is chosen. The authors provide experimental evidence that this strategy has successful outcome.

The approach suffers from some drawbacks: the detection of the wrong cluster of users can result in filtering-out of genuine users, and thus predicting baised estimates. Further, real life attacker might employ strategies which ensure more deviation in their votes, thus fooling the filtering process. Also, there might be both push and nuke attackers for the same item, which the algorithm is not designed to handle. Thirdly, the running time of the algorithm will be much higher than what is required for large-scale systems. Fourthly, given that neighborhood selection methods are thresholded, it is possible that the number of attackers is so high that the selected neighbours of a user may all be malicious; thus the outlined approach might fail in the face of large and continuous attacks.

3 Robust Collaborative filtering using Existing Approachces: Association Rules and PLSA

4 Robustness of Association Rules

A popular approach for frequent-pattern mining is the use of association rules. This technique has been used for market basket analysis, finding interesting patterns of what users buy together, and have been found useful as prediction models too [1]. Applying this technique to user ratings, [13] suggest how a robust method for collaborative filtering could be devised. The application of the above approach has been demonstrated to provide significant robustness to user recommendations. As compared to kNN, k-means clustering and PLSA, the outlined algorithm has a very low *hit-ratio*, meaning that a small fraction of users are recommended an attacked item. For attack sizes below 15%, the hit-ratio of Association rule-based CF is below 0.1, while for k-NN it ranges from 0.8 to 1.0 (meaning all users are recommended an attacked item). For more details, we refer the interested reader to reported numbers from [13](Fig 2).

Clearly, the hit-ratio for association rules is very low, thus implying high robustness. This however comes at the cost of accuracy; Sandvig et al. report that the coverage of this algorithm is below 0.5, which means that the algorithm cannot make any predictions for over half the items in the recommender system. These are typically items which are not very frequently rated. This makes the above approach suitable only in cases where top-n recommendations are required, rather than a complete set of predictions.

4.1 Robust Collaborative filtering using PLSA

Probabilistic Latent Semantic Analysis (PLSA) is a well known approach for text analysis and indexing used to discover hidden relationships between data; it has also been extended to collaborative filtering [5]. PLSA enables the learning of a compact probabilistic model which captures the hidden dependencies amongst users and items. While accuracy has been a well known advantage of PLSA, recent studies have also concluded that PLSA is a very robust CF algorithm, and is highly stable in the face of shilling attacks. [11] indicates that the prediction shift for PLSA is much lower than similarity based approaches; [7] investigated the reasons for PLSA's robustness over many experiments and observed the model to understand the mechanisms. The intuition is that PLSA leads to clusters of users (and items) which are used to compute predictions, rather than directly computing neighbors. However this intuition is challenged by experimental results using a k-means clustering algorithm in the same work. Clearly, shilling profiles deceive clustering algorithms due to their high similarity with normal users.

[7] also outlines a detection approach for shilling attacks exploiting the high stabilty of PLSA. The main idea is that PLSA is a mixture model where membership to a distribution is not constrained; a data point can belong (probabilistically) to many distributions. However some clusters maybe *tighter* thn others: [7] shows that using the average Mahalanobis distance to indetify tight clustes leads o the detection of attack profiles with reasonably high accuracy.

Robust PLSA using shilling detection We suggest using the following strategy to further robustify PLSA: we eliminate the tightest clusters, as identified by the above tightness measure. We now renormalize the probability distribution of the remaining clusters ($p(z|u)$) so that they sum up to 1. One can even attempt to eliminate the suspicious users, randomly perturb the parameters and rerun the last few steps of training. Initial results of this version have maintained the prediction accuracy, while reducing the prediction shift in a statistically significant manner. A more thorough investigation of this idea is under progress.

5 Robust Collaborative Filtering using SVD

SVD stands for Singular Value Decomposition; it is a method of factorizing a matrix into two orthonormal matrices and a diagonal matrix. SVD has become an important linear algebra procedure over the last 2 decades due to its extensive application in Information Retrieval and Data mining. It has been used for Latent Semantic Analysis and Collaborative Filtering with much success. Since SVD is fundamental to the algorithms discussed in this paper, we explore SVD in detail. Further, we briefly explain the Robust Matrix Factorization algorithm described in [9] which is also based on SVD and is robust variant of SVD. Finally, we explain our proposed VarSelect SVD variant as a robust CF solution.

5.1 Singular Value Decomposition (SVD)

SVD is a more general form of Eigen value decomposition (EVD), applicable to rectangular matrices. SVD factorizes a rectangular $n \times m$ matrix $\mathbf{D}$ as $\mathbf{D} = \mathbf{U\Sigma V}^{\mathrm{T}}$ where $\mathbf{U}, \mathbf{V}$ are unitary normal matrices and $\mathbf{\Sigma}$ is a diagonal matrix of size $\mathrm{rank}(\mathbf{D}) \leq \min(m, n)$, where $\mathrm{rank}(\mathbf{D})$ is the rank of the matrix $\mathbf{D}$. Moreover, the entries on the diagonal of $\mathbf{\Sigma}$ are in non-increasing order such that $\sigma_i \geq \sigma_j$ for all $i < j$. Note that we may chose to set all singular values $\sigma_i = 0$, $i > k$ for some $k \leq \mathrm{rank}(D)$ (say $k = 10$), leading to a low-rank approximation $\mathbf{D}_k$ of the matrix $\mathbf{D}$ ($\mathbf{D}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^{\mathrm{T}}$).

SVD for Collaborative Filtering: Applications of SVD to Collaborative Filtering assume the representation of user-item ratings by such a $n \times m$ matrix $\mathbf{D}$. Typically, user–item matrices are very sparse ($\leq 5\%$ non-zero entries). Initial applications of SVD to CF (c.f. [14]) compensated for sparsity by replacing the missing values by overall mean. This approach, though more successful than previous CF approaches, is highly biased towards the used means. In addition, the lack of sparsity implies a larger computational problem to solve. In the last decade, there has been significant research on SVD for large and sparse matrices e.g. *PROPACK* and *SVDPACK*. However, these approaches do not treat missing values in a principled fashion, either treating them as zeros, or doing mean imputation. A recent algorithm by Gorrell [4] proposed a new approach to computing SVD for virtually unbounded matrices. This method is based on the Generalized Hebbian Algorithm and calculates SVD by iterating through only observed values. The method has been found to be highly accurate for CF and scales easily to the NetFlix dataset with 100 million votes.

5.2 Robust Matrix Factorization

Robust regression problems have been studied in a linear setting where observables Y and inputs X are known and Y is assumed to be noisy. Robust Matrix Factorization (RMF) is algorithm which performs a robust SVD for CF using an alternating fitting scheme [9]. The core idea is the use of bounded cost-functions, which limit the effect of outliers. There is an entire body of work on such bounded functions which are effective against noise; these functions are called Maximum Likelihood estimators or *M-estimators*.

Armed with a robust estimator, we would like the perform the following Matrix factorization: assume we want to find the rank–1 factors $\mathbf{G}, \mathbf{H}$ as for data $\mathbf{D}$. such that

$$\underset{\mathbf{G},\mathbf{H}}{\mathrm{argmin}} \sum_{D_{ij} \neq 0} \rho(D_{ij} - G_i \cdot H_j) \text{ s.t. } \rho(r) = \begin{cases} |r| \leq \gamma & \frac{1}{2\gamma} r^2\,, \\ |r| > \gamma & |r| - \frac{\gamma}{2} \end{cases}$$

where ρ is an M-estimator called the Huber M-estimator (see [9]. The above optimization can be solved using *Iteratively Re-weighted Least Squares* and is described by [9]. Experiments show that Robust Matrix factorization algorithm performs well in the face of moderate attacks. Clearly, the effect of shilling is low at small attack sizes, as the majority opinion is given more importance. However, once the number of votes by shillers are more than actual users, RMF starts treating the shillers' view as the majority opinion. Mehta et al. also show that RMF is more tolerant to shilling and model deviations than SVD and pLSA: importantly, the prediction accuracy of RMF is higher than any other method; this trend continues even in the face of attacks.However for larger attacks, RMF is clearly inadequate at a robust CF algorithm. In the next section, we show how the RMF and SVD frameworks can be further robustified to yield our desired robust CF algorithm.

5.3 VarSelect SVD for Collaborative Filtering

VarSelect [8] is a variable selection algorithm based on PCA for detecting attack profiles. Shilling profiles tend to be highly correlated, which is a result of the colluded nature of shilling attacks. It is known that for multivariate data, highly correlated variables add very little information, and thus are eliminated by dimensionality reduction methods. VarSelect uses Principal Component Analysis to find which users add least information, and produces a ranking of users in order of utility. Experiments have shown that shillers are found with high precision at the top of these rankings.

VarSelect SVD: We first describe the broad framework for our proposed algorithm. SVD and PCA are closely related since PCA can be achieved via SVD. In essence, PCA seeks to reduce the dimensionality of the data by finding a few orthogonal linear combinations (called the *Principal Components*) of the original variables with the largest variance. A principal component is a linear combination of the variables and there are as many PCs as the number of the original variables. Principally, PCA is equivalent to performing an eigen decomposition of the *covariance* matrix of the original data. Since we want to combine VarSelect with Collaborative Filtering, SVD provides the required framework. The algorithm supports two phases: detection (followed by removal of profiles/votes), and recommendation/model building. For efficiency, it is required that these two phases can share computational steps. Since the detection may not be perfect, no user profiles should be completely deleted and even suspected attackers should be able to receive recommendations. Further, the entire procedure should be unsupervised, i.e. no further input should be required after the detection phase has been performed (e.g. thresholding how many shillers are there in the system).

An important observation we make here is that calculating the covariance matrix is unnecessary; we can compute the SVD of $\mathbf{X}$ to get the loading matrix $\mathbf{U}$ (which contains the Principal components). This saves a significant computational effort as Eigen-decomposition of large covariance matrices is very expensive. Note that PCA requires $\mathbf{X}$ to be zero-mean. This can be explioted by the VarSelect procedure which has been shown to require only the first 3–5 Principal components suffice to detect attack profiles reliably. Thus a complete SVD is not required: instead, partial eigen-decomposition can be performed. Such routines are available as *svds* and *eigs* in MATLAB and Octave, and use the Arnoldi method.

Finding suspected Attack profiles: PCA can find a set of variables which are highly correlated, a fact exploited in the design of Varselect. Varselect essentially performs Variable Selection using a selection criteria called *Normalized Loading Combination*. There are several other selection procedures discussed in literature ([2, 10] provides a good overview of these criteria). [10] reports that the simplest strategy of averaging loading coefficients (*LC*) performs the best. We choose the following heuristic: normalize the scores so that they sum to 1, and then choose all user with scores below $1/n$ for n users. We observe also that 50% recall is the lowest observed; thus we suggest that for attacks of upto 10%, flagging top-20% should suffice. These selected users are known as *flagged* users.

Computing Recommendations The recommendation model is finally based on SVD as well. In essense, we perform SVD on the data matrix treating flagged users in a special manner. To simplify the prediction model, we absorb the eigenvalues into the left and right factors, as in the GHA-based SVD method. As previously, the data matrix is factorized into a factors $\mathbf{G}$ and $\mathbf{H}$, such that the Frobenius norm of the remainder is minimized:

$$\underset{\mathbf{G},\mathbf{H}}{\operatorname{argmin}} \, ||\mathbf{D} - \mathbf{GH}||_F, \tag{3}$$

In the context of Collaborative Filtering, note that the left matrix $\mathbf{G}$ is user specific, i.e. each user has a corresponding row encoding their hidden preferences. Similarly, the right matrix $\mathbf{H}$ contains a column for each item. The solution to the above optimization requires iterating through all user votes and performing Hebbian updates. Every vote potential influences both $\mathbf{G}$ and $\mathbf{H}$ during the training phase.

Our modification in presence of *flagged* users is to only update the left vectors and not the right vectors. In other words, the contributions of suspicious users towards the prediction model is zero, while the model can still predict the votes for flagged users. For normal users, we update both left and right vectors as with SVD-GHA. This elegant s solution comes with a very small computational cost of checking if a given user is flagged as suspicious. Note also that the model can be initialized with values learnt from the partial SVD performed for PCA. We note that this results in faster convergence for the already computed dimensions. Additionally, we use a regularization parameter κ (set to 0.01); this step has been found to provide better model fitting and faster convergence.

One issue with the above algorithm is that coverage is low for high number of suspected users r. It is possible that some items are voted on mostly by flagged users, hence enough information may not be known. Therefore, even interested users may not be recommended that item. To improve coverage, we ignore only the extreme votes of the flagged users (i.e. maximum vote 5/5 and minimum 1/5); middle votes can be still used to train the right vectors. This removal significantly weakens potential bandwagon attacks as well as average attacks, since the largest deviations in prediction occur due to extreme votes.

6 Discussion

In the previous sections, we have described several state-of-the-art algorithms for robust collaborative filtering. Clearly, there has been more work in the detection of shilling attackers, rather than modelling shillers as a type of noise. In our opinion, a robust recommender algorithm should have the following characteristics:

1. Highly stable to low number of attack profiles inserted on the attacked item (i.e. low prediction shift)
2. Moderately–Very stable against medium sized attacks ($< 5\%$ attackers)
3. Low average effect on prediction accuracy (mean average error) on non-attacked items.
4. Very high stability to the addition of random noise.
5. No loss of accuracy if no attackers are present in a dataset.
6. Ability to partially trust users, i.e. to be able to generate recommendations for suspicious users.
7. Scalability to handle hundreds of thousands of users with a few thousand possible attackers.
8. Ability to handle multiple simultaneous attacks on different items.
9. Ability to generalize to attack models not encountered in training.
10. Non-requirement for processing the entire dataset again when new profiles are added.
11. Being as parameter-free as possible: the algorithm should be able to figure out various thresholds based on the data without requiring human intervention.

Experimental results published previously show that most algorithms surveyed by us do not conform to a majority of the items in the above checklist. Almost all algorithms we studied have a high degree of stability against medium sized attacks. Some of the algorithms are able to handle low attacks, and detect them reliably. Surprisingly, most of the algorithms meet point 3 and 4 as well: random noise has low effect on both neighbor selection based methods and model-based methods. The performance of these algorithms also does not suffer on non attacked items; the only exception might be in methods where there is an explicit step for detecting an item under attack, and this step gives false positives.

On the issue of running these algorithms on untainted data without any attack profiles, most researchers have not reported the performance of their approaches. However, all algorithms that do some user filtering (e.g. removing suspicious users) do suffer from loss of accuracy. An interesting result in this context has been reported in [9]: in this work, some part of the user ratings was removed. The removed data was a random fraction of the extreme votes (say the lowest, and the highest numerical votes), usually to the tune of 20% of a user's votes; for users less than 10 votes. Various algorithms (e.g. kNN, PLSA, SVD and RMF) were run on the remaining 80% data; the results were that all algorithms gained significantly more stability to shi;lling attacks, to the tune of 20% reduction in prediction shift. The RMF and SVD algorithms had even higher stability; interestingly all algorithms under test did not suffer from a large decrease in predictive accuracy. However, the overall performance of SVD and PLSA based algorithms was much better than kNN.

With respect to partial trust, all algorithms which explicitly remove suspicious user profiles will degrade user experience for suspected process. Since the detection of malicious users profiles is not perfect, false positives will arise from time to time. An ideal algorithm should thus be able to accept that attackers maybe in the dataset, and model this explicitly. The main idea is that all users should be able to receive recommendations, and possibly the user interface should not change at all. The algorithm may internally discard some user data for training, or some ratings and making this transparent to users. Algorithms which do this provide high stability as well: VarSelect SVD and Association Rule mining are examples of this. Intelligent neighbor filtering can do this as well; it has been suggested to weight the contribution of a user's neighbors by the degree of trust (which can be easily expressed as a fraction between 0–1) with good success.

Scalability is a general issue for good algorithm design; efficient algorithms are available for several methods used for recommendation and also for detection. With increasingly cheap hardware and cloud computing becoming common, computational challenges are slowly fading away. However, responsiveness of systems to sudden attacks is crucial. Several approaches seem to run in batches, making it difficult to detect attacks online. To a certain extent, the training part of the recommendation models is also offline, thus restricting the extent of the impact. However, the overall scalability of detection is an important aspect: several approaches have expensive detection algorithms which cannot reuse the previously trained models, or incrementally train their models. Approaches like Varselect SVD which are online in nature are highly scalable as a result.

In the real world, several interest groups would try to boost their products at the same time. Thus one can expect more than one type of attack going on at the same time. Almost all published work on detection however consider only one type of attack at a time for their experimental. It is clearly possible that several of these might actually be effective against multiple attacks; for example the detection approaches proposed by [3] use supervised classification trained on generated examples of several types of attacks. These classifiers are then run on each profile, thus possibly detecting more than one types of attack at a time. Similarly, VarSelect detection and VarSelect SVD were demonstrated to be effective against uncoordinated attacks. While no such results have been discussed by [13], it is likely that this approach will be stable to multiple attacks as there wont be much co-occurrence data for the attacked items to make attack votes significant for the association rule learner (recall that less than 50% of items are actually considered by this approach when creating recommendations.)

The continued menace of email spam shows that given enough incentive, spammers can innovate and create new types of attack campaigns, While various attack models for shilling have been identified, it is ovbious that there are several strategies for new attacks that spammers can come up with. Thus it is imperative for robust CF algorothms to be able to generalize to new types of attacks. Supervised learning methods can clearly fail in this regards; for unsupervised methods to suceed, it is important to understand the intent of the attackers. [7] showed that if the intent is to maximize the prediction shift for a large number of users, the resulting attack model is the average attack. [8] also discusses how the *group effect* is a result of spammers trying to maximize their impact. When obfuscation strategies are employed to add stealth to attack profiles, the strength of attacks goes down. Exploiting the group effect is one mechanism for general attack detection.

One practical aspect we noticed is that several algorithms have too many parameters that need to be manually set. In a real world setting, exploring these parameters manually may not be possible or efficient. It would be better if the algorithm can search its own parameters, either using heuristics, or some principled mechanism (like cross-validation). VarSelect SVD is an example of such an algorithm; while the parameter search is not optimal, there is a broad range of stable values for the parameter r which leads to good stabilty and high maintainability.

References

[1] R. Agrawal, T. Imieliński, and A. Swami, *Mining association rules between sets of items in large databases*, ACM SIGMOD Record **22** (1993), no. 2, 207–216.

[2] Noriah M. Al-Kandari and Ian T. Jolliffe, *Variable selection and interpretation in correlation principal components*, Environmetrics **16** (2005), no. 6, 659–672.

[3] R. Burke, B. Mobasher, C. Williams, and R. Bhaumik, *Classification features for attack detection in collaborative recommender systems*, ACM Press New York, NY, USA, 2006, pp. 542–547.

[4] Genevieve Gorrell, *Generalized hebbian algorithm for incremental singular value decomposition in natural language processing.*, EACL, 2006.

[5] Thomas Hofmann, *Latent semantic models for collaborative filtering.*, ACM Trans. Inf. Syst. **22** (2004), no. 1, 89–115.

[6] Shyong K. Lam and John Riedl, *Shilling recommender systems for fun and profit*, WWW '04: Proceedings of the 13th international conference on World Wide Web (New York, NY, USA), ACM Press, 2004, pp. 393–402.

[7] Bhaskar Mehta, *Unsupervised shilling detection for collaborative filtering*, AAAI, 2007, pp. 1402–1407.

[8] Bhaskar Mehta, Thomas Hofmann, and Peter Fankhauser, *Lies and propaganda: detecting spam users in collaborative filtering*, IUI '07: Proceedings of the 12th international conference on Intelligent user interfaces (New York, NY, USA), ACM Press, 2007, pp. 14–21.

[9] Bhaskar Mehta, Thomas Hofmann, and Wolfgang Nejdl, *Robust Collaborative Filtering*, In Proceedings of the 1st ACM Conference on Recommender Systems (Joseph A. Konstan, John Riedl, and Barry Smyth, eds.), ACM Press, October 2007, pp. 49–56.

[10] Bhaskar Mehta and Wolfgang Nejdl, *Attack-resistant Collaborative Filtering*, To appear: In Proceedings of the 31st ACM SIGIR Conference, ACM Press, 2008.

[11] Bamshad Mobasher, Robin D. Burke, and Jeff J. Sandvig, *Model-based collaborative filtering as a defense against profile injection attacks.*, AAAI, 2006.

[12] Michael P. O'Mahony, Neil J. Hurley, and Guenole C. M. Silvestre, *An evaluation of neighbourhood formation on the performance of collaborative filtering*, Artificial Intelligence Review - Special Issue **21** (2004), no. 3-4, 215–228.

[13] J. J. Sandvig, Bamshad Mobasher, and Robin Burke, *Robustness of collaborative recommendation based on association rule mining*, RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems (New York, NY, USA), ACM, 2007, pp. 105–112.

[14] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, *Application of dimensionality reduction in recommender systems–a case study*, 2000.

[15] Sheng Zhang, Yi Ouyang, James Ford, and Fillia Makedon, *Analysis of a low-dimensional linear model under recommendation attacks*, SIGIR, 2006, pp. 517–524.