Efficient and Secure Collaborative Filtering through Intelligent Neighbour Selection

Michael P. O'Mahony and **Neil J. Hurley** and **Guénolé C.M. Silvestre**[1]

Abstract. In this paper, we introduce novel neighbourhood formation and similarity weight transformation schemes for automated collaborative filtering systems. We define *profile utility*, which models the usefulness of user profiles for collaborative filtering as a function of the items they contain. We demonstrate that our approach leads to more efficient and scalable collaborative filtering when compared to a benchmark k–Nearest Neighbour approach, while providing system accuracy and coverage to the same standard. In particular, we show that our approach is completely secure against the malicious attacks outlined in the paper, whereas k–NN proves very vulnerable.

1 Introduction

Memory-based automated collaborative filtering (ACF) [8, 1, 9] is a recommendation algorithm which makes recommendations to users based on the ratings of similar users. A key element of the ACF algorithm is the user similarity measurement which is typically used for two separate purposes. Firstly, it is used in the selection of the set of neighbours whose ratings are combined to form the recommendation – the most similar users are selected. Secondly, it is used to form the weights in the computation of the recommendation as a weighted average of the neighbours' ratings. It is important to distinguish between these two stages of the algorithm and we argue that different approaches might be appropriate for each. In particular, the goal of neighbour selection is to select the most useful neighbours on which to base the prediction. While similarity is one measure of usefulness when considering the accuracy of the algorithm, the notion of neighbour *utility* can be extended to include issues such as efficiency and robustness. For example, the on-line efficiency of ACF is directly proportional to the number of similarity measurements that are made. To form a neighbourhood based on similarity, the similarity of *all* potential neighbours must be calculated. Basing neighbour selection on a less costly utility measure can lead to improved efficiency. In other work, clustering [10] and dimension-reduction [9, 2] are two approaches that have been proposed to improve efficiency.

In previous work [6, 5], we have identified *robustness* as another important performance measure of an information retrieval system. In the case of ACF, we have shown that it is possible for a malicious user to significantly manipulate the recommendations of the system by inserting small numbers of specially-tailored attack profiles into the system. These attack profiles are based on popular items (i.e. items that have been rated by many users in a system), on the basis that a profile containing such items is in the neighbourhood of a large number of users and that popular items tend to be positively rated. Therefore it is possible to predict the likely correlation of an attack profile with other users. In [5] we have proposed a neighbourhood filtering mechanism to defend against attack, but this mechanism is not completely robust against more sophisticated attacks.

It is well recognised that frequently occurring words are not a good basis for calculating the similarity between documents in information retrieval. Indeed, an inverse user frequency approach has been proposed for ACF [1], which seeks to penalise the ratings given to popular items. Here, we go a step further by proposing a profile utility measure based on certain characteristics of the items contained in a profile. Users with the highest utilities are selected as neighbours.

In this paper, we demonstrate that such filtering does not adversely affect accuracy or coverage (the percentage of predictions sought which a system is able to deliver), but dramatically improves the robustness of the system against the attacks that we have considered. Another important advantage of this neighbour selection policy is that profile utility can be calculated offline and is easily updated when new items are added. Hence neighbourhoods can be formed in a very efficient manner regardless of database size.

2 Memory-Based Collaborative Filtering

We base our analysis on the widely used GroupLens [8] algorithm, in which a prediction, $p_{a,j}$, for user a (the active user) and item j is calculated from the ratings of similar users (neighbours) as follows:

$$p_{a,j} = \bar{v}_a + \frac{\sum_{i=1}^{n} w(a,i)(v_{i,j} - \bar{v}_i)}{\sum_{i=1}^{n} |w(a,i)|} \tag{1}$$

where n is the (fixed) neighbourhood size, $v_{i,j}$ is the rating of user i for item j and $\bar{v}_a$ is the mean rating for user a. The weights, $w(a,i)$, are calculated using a similarity metric – we use *Pearson correlation* [8] which has been shown to give good performance [1].

It has been noted in [3] that weights calculated on the basis of small numbers of co-rated items may not reflect the true similarity between users. Thus, if n is the number of co-rated items, we modify the weights calculated using Pearson correlation as follows: $w'_{a,i} = w_{a,i} \times \frac{n}{N}$ if $n < N$, where N is a constant (set to 50 in our experiments as per [3]). The weight remains unchanged if $n \geq N$.

We compare our scheme as described in Section 4 against the above k–NN algorithm. In [6], we compared the performance of k–NN against a neighbourhood thresholding approach using a variety of similarity measures. We found that k–NN using Pearson correlation provided the best performance in terms of predictive accuracy and coverage, although robustness against attack was poor. Nevertheless, we feel that this algorithm represents a sound benchmark since any new approach should achieve comparable accuracy and coverage, while, of course, offering greater efficiency and scalability and improved robustness against attack.

[1] University College Dublin, Belfield, Dublin 4, Ireland email: michael.p.omahony@ucd.ie

3 Robustness & Attacks

In previous work [6, 5], we have successfully implemented malicious attacks on ACF systems and have proposed robustness as an additional performance measure as a complement to existing measures – predictive accuracy, coverage, etc. We define robustness as follows:

DEFINITION 1. *Robustness is the ability of a system to provide accurate predictions given some degree of noise present in the data.*

In general, one must expect a certain level of noise to be present in all systems. When explicit rating entry is required by the user, for example, noise is likely to be introduced through carelessness, human error or when rating scales do not provide users with sufficient options. These are examples of *unbiased* noise. It is also possible that *biased* noise, entered with a specific motive in mind, may be present in a system. Such malicious attacks are the focus of this paper.

In our work to date, we have focused on three attack types: product *push*, product *nuke* and generalised *random* attacks. The goals of product push and nuke attacks are to promote or demote the predictions made for targeted items, respectively. For example, consider an author who wishes to increase his own sales by forcing a recommender system to output artificially high ratings for his work (push), while reducing predictions made for his competitors work (nuke). In random attacks, the objective is to reduce the overall performance of a system as a whole in an attempt to compromise the system's integrity. As recommendation quality deteriorates, users will begin to loose trust in the system and seek other solutions. As a potential real-life scenario, consider a recommender system owner who wishes to attract additional customers by attempting to undermining the output quality of rival systems. In this paper, due to limitations of space, we analyse product push attacks only, noting that our approach is also effective against these other attacks as outlined in our previous work.

3.1 Product Push Attack: Strategy

The attack is implemented by creating bogus or attack user profiles and inserting this data into a system through the normal user interface – no other access to a system database is assumed for attackers.

From the attacker's perspective, there are several criteria that need to be satisfied if attacks are to be successful. Firstly, if attack profiles are to influence predictions, they need to be present in the neighbourhoods of targeted users. Since it is not tractable to create separate attack profiles to target every genuine user in a system, attack profiles need to a strong similarity with as many genuine users as possible. Our approach is to build attack profiles using popular items since, by definition, many genuine profiles contain these items and in addition, the ratings for these items are likely to be consistent and high.

Secondly, for the Pearson similarity metric, attack profiles need to correlate in the same direction (i.e. either positively or negatively) with targeted users if predictions made for items are to be pushed. (If this is not the case, the opposite effect will be achieved, and items will instead be nuked). Thus, along with the item to be pushed, the attack profiles are constructed from two "groups" of items. The first group consists of items that are generally rated higher than average in the database (i.e. *liked* items), and the second group consists of items that are generally rated lower than average (*disliked* items). By assigning a higher rating to the liked items, an attacker can be confident that attack profiles will correlate in the same direction with the majority of genuine users in the database.

Referring to (1), the contribution of any potential neighbour to a prediction depends on the magnitude or the term $(v_{i,j} - \bar{v}_i)$. For attack profiles, the obvious strategy is to choose ratings for the item groups that will maximise this term. Thus, the minimum rating, R_{min}, is assigned to each of the disliked items and ratings of $R_{min} + 1$ to the liked items. The item being pushed is set to the maximum rating, R_{max}. While this strategy requires a certain knowledge of the data contained in databases, it is not unreasonable to assume that such knowledge is possible to estimate (e.g. in movie domains) or to mine (e.g. on Amazon.com, using feedback provided by users).

3.2 Metrics

We adopt two metrics to evaluate system robustness which are deliberately independent from system accuracy. We feel that this is an important requirement as the following example illustrates. The metric commonly used to evaluate predictive accuracy is *Mean Absolute Error* or MAE, which measures the absolute difference between true and predicted ratings. Suppose that the true rating for a particular item given by a certain user is 3 and that the pre- and post-attack predictions for this item are 2 and 4 respectively. In this scenario, MAE remains unchanged following the attack even though a significant prediction shift has occurred. As this simple example shows, MAE is not an adequate metric to measure system robustness.

3.2.1 Mean Absolute Prediction Error (MAPE)

We define our first metric, MAPE, as the absolute difference between pre- and post-attack predictions. Let A_i be the set of users over which we evaluate an attack on item i. We calculate MAPE for item i as

$$\text{MAPE}(i) = \frac{1}{|A_i|} \sum_{a \in A_i} |p'_{a,i} - p_{a,i}| \tag{2}$$

where $p'_{a,i}$ is the post-attack prediction. We then calculate the overall robustness of a system by taking the average MAPE over all items attacked. A low value of MAPE indicates that a system is robust to a particular attack. With this metric, the prediction shift evident in our example above is captured, giving the desired result of 2.

3.2.2 Percentage of Good Predictions

Another approach to measure attack success is to calculate the percentage of *good predictions* that are made for targeted items at various attack strengths. We define a good prediction as either the best or second-best rating on a particular scale. For a (successful) product push attack, we would expect to observe an increase in the number of good predictions that are made for targeted items. This represents a useful measure of attack success since users are more likely to act on recommendations made for items with high predicted ratings.

4 Neighbour Selection Strategies

In ACF, neighbours are typically selected on the basis of similarity to the active user. Cosine similarity, Spearman Rank and in particular, Pearson correlation, have been widely used to calculate similarity based on the co-rated items between users.

In previous work [1, 7], it has been recognised that certain items may be of greater importance when calculating similarities. For example, unpopular items (i.e. items that have not been rated by many users in a system) and items that receive diverse ratings by the user population may be of more use in distinguishing between users and in identifying users' tastes. In the next section, we expand on these observations to formulate a novel neighbour selection strategy.

4.1 Profile Utility

In our approach, we assign to each user profile a global weight that models a profile's usefulness as a potential neighbour, based on certain characteristics of the items that are contained in the profile. In general, we define the profile utility of user a as follows:

$$U(a) = f(j_1, j_2, ..., j_n) \tag{3}$$

where $j_i \in I_a$ is the set of items user a has rated. Firstly, we consider profile utility in terms of item popularity, and posit that users who have rated predominantly popular items are unlikely to serve as good predictors. Specifically, we rate utility as the summation of the inverse popularity of items contained in a profile as follows:

$$U(a) = \frac{1}{|I_a|} \sum_{j \in I_a} \text{InvPop}(j) \tag{4}$$

We define the inverse popularity of an item as:

$$\text{InvPop}(j) = \begin{cases} 1 - \frac{n(j)}{m} & \text{if } n(j) < m \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where $n(j)$ is the number of users who have rated item j and m is a threshold popularity. Profiles containing unpopular items are thus assigned a higher weight, and the parameter m can be used to control the set of possible neighbours by assigning a zero contribution from items that exceed the specified popularity threshold. Neighbourhoods are formed by simply selecting k users with the highest utility that contain the item for which a prediction is being sought. Once neighbours have been selected, predictions are calculated as in (1).

In a similar manner, utility can also be defined in terms of item rating distribution. The greater the variation of ratings across items, the more likely it is that such items are useful to the prediction process. Thus, we can define utility in terms of the entropy of items in a user's profile, where the entropy of item j is defined as:

$$H(j) = -\sum_i p_{i,j} \log_2(p_{i,j}) \tag{6}$$

where $p_{i,j}$ is the probability of ratings on item j valued i. Profile utility can now be defined as:

$$U(a) = \frac{1}{|I_a|} \sum_{j \in I_a} \frac{H(j)}{H(j)_{max}} \tag{7}$$

where $H(j)_{max}$ is the maximum entropy of item j which assumes that the distributions over all classes of ratings are identical. In addition, other measures of profile utility may be considered – for example, thresholds may be applied to item entropy, item popularity may be combined with entropy data, etc.

4.2 Similarity Weight Transformation

In addition, it is necessary to reduce similarity weights for attack profiles to zero to protect predictions made for new or less popular items – i.e. where the number of genuine users who have rated such items is less than the neighbourhood size. In these circumstances, at least some attack profiles created to target such items would influence predictions if they were not filtered from neighbourhoods. This observation motivates the following similarity transformation scheme:

$$w'_{a,i} = w_{a,i} \times \frac{1}{|I_a \cap I_i|} \sum_{j \in I_a \cap I_i} \text{InvPop}(j) \tag{8}$$

where the inverse popularity of items is calculated as before. By choosing a low value for the popularity threshold m, the transformed weights for attack profiles (that consist primarily of popular items) will approach zero. By combining this scheme with that of Section 4.1, a system is much more likely to be robust against the type of attacks considered in this paper. As before, weights can also be transformed using item entropy, etc.

5 Evaluation

We used three datasets in our evaluation. The MovieLens dataset [4] consists of 943 users, 1,682 movies and contains 100,000 transactions in total. Movies are rated on a scale of 1 to 5.

The EachMovie dataset, provided by Compaq Equipment Corporation, has some 72,916 users who entered a total of 2,811,983 numeric ratings for 1,628 different movies. From this, we selected a random sample of 1,399 users, containing 91,982 transactions on a rating scale of 1 to 6 (modified from the original 6-point scale of 0 to 1).

Finally, the Smart Radio dataset, provided by the Department of Computer Science, Trinity College Dublin, contains 4,075 ratings by 63 users on 1,055 songs. The rating scale is from 1 to 5. This dataset contains significantly less users than those above.

The experimental procedure adopted in all cases was to remove a single user-item pair from the dataset, and a prediction for this pair was then made using all remaining data. In all cases, we present average results over all items contained in the respective datasets. For each dataset, the optimal neighbourhood size was chosen by experiment, with k set to 60 for both MovieLens and EachMovie and 18 for Smart Radio. The number of items in each attack profile was approximately 50 for MovieLens and EachMovie and 20 for Smart Radio. In this paper, we present results for profile utility and similarity weight transformation in terms of inverse popularity only – see Section 7 for comment on the the other approaches discussed earlier. Due to limitations of space, we do not present results for the EachMovie dataset – note, however, that the trends observed for this dataset closely matched those of the MovieLens dataset.

5.1 Performance of Benchmark k–NN Algorithm

To begin, we evaluate the effect of the product push attack using the benchmark k–NN algorithm. In Table 1 we present MAPE against attack strength for the MovieLens and Smart Radio datasets. For both datasets, significant prediction shifts were achieved by the attack. For example, an MAPE of 2.10 resulted for MovieLens for an attack strength of 50 attack profiles inserted. Note that greater than 98% of these prediction shifts were positive, as required for a product push attack. Given that MovieLens operates on a rating scale of 1 to 5, a shift of 2 is sufficient to change a prediction from "dislike" to "like". A similar result obtained for Smart Radio when 8 attack profiles were inserted. For both datasets, MAPE remained constant after particular attack strengths – these points coincide with the neighbourhood sizes used by the algorithm (k was set to 60 for MovieLens and 18 for Smart Radio). In our experiments, we used the same attack profile repeatedly and therefore no additional effect could be achieved by inserting more than k attack profiles into the system.

The change in the number of good predictions versus attack strength for k–NN is also presented in Table 1. For MovieLens, when **only** 1 attack profile inserted into the database, the average percentage of good predictions achieved across all dataset items was 28%, compared to 18% pre-attack – a percentage increase of 57%. For

Table 1. The robustness provided by the benchmark k–NN algorithm when subjected to a product push attack.

MovieLens			Smart Radio		
# Attack Profiles	MAPE	% Good Preds.	# Attack Profiles	MAPE	% Good Preds.
0	0	18	0	0	41
1	0.35	28	1	0.92	73
5	0.88	49	2	1.31	83
10	1.20	62	4	1.73	90
30	1.79	77	8	2.12	94
50	2.10	79	12	2.32	95
70	2.20	79	18	2.55	95
100	2.20	79	24	2.55	95

Smart Radio, the attack was even more successful when, again, just 1 attack profile was inserted. In this case, the percentage of good predictions increased by 77% – from 41% to 73%. These results indicate that there is a substantial benefit for potential attackers. It is reasonable to assume that users are more likely to act on recommendations that receive high predicted ratings. Thus, in the context of a product push attack, the insertion of only a few attack profiles has the potential to convert a significant number of "browsers" into "buyers".

The trends observed using both metrics were similar. Further, it is apparent that the *cost/benefit ratio* (from the attacker's perspective) began to fall as attack strength increased. If we model attack cost by simply the number of attack profiles inserted, we can see that there is little additional gain for the attacker by inserting more than 30 attack profiles into the database (MovieLens). At this attack strength, the percentage of good predictions achieved across all items was 77% – only a further increase of at most 2% was achieved at greater attack strengths. Since attack profiles were designed to cause maximal prediction shifts, the presence of even low numbers of attack profiles in neighbourhoods resulted in significant prediction shifts.

While these results are encouraging for would-be attackers, they clearly present a real cause for concern for recommender system operators. Thus having established that certain malicious attacks are indeed feasible and the need for a robustness performance measure for ACF, we now evaluate our approach, designed to secure systems against attack, as outlined in Section 4.

5.2 Performance of Profile Utility/Similarity Weight Transformation Algorithm

In this section, we compare our approach against the benchmark k–NN algorithm using several performance criteria and discuss the selection of a suitable value for the popularity threshold value m.

5.2.1 Efficiency and Scalability

In our approach, we simply choose as neighbours the k users with the highest profile utility who have rated the item for which a prediction is being sought. For k–NN, similarities must be computed between the active user and all other users who have rated the item in question before neighbours can be selected. Since profile utility can be computed offline and is easily updated when new items are rated, it represents a considerable increase in efficiency when compared to k–NN. Formally, the profile utility algorithm is $O(j)$ compared to $O(r.j)$ for k–NN, where j is the average number of items co-rated by the active user and neighbour profiles and r is the number of users who have rated the item for which a prediction is sought. In contrast to k–NN, the efficiency of our approach is independent of r and thus does not suffer from scalability problems as a system grows in size.

5.2.2 Accuracy and Coverage

In Tables 2 and 3, we compare the performance of our approach to that of the benchmark k–NN algorithm. The data shows how the predictive accuracy and coverage provided by our approach varied with the threshold value m. Accuracy is measured according to Mean Absolute Error (MAE). Note that the performance of the benchmark k–NN algorithm is not effected by m and thus accuracy and coverage values presented for this algorithm remain constant in the tables.

The data presented for k–NN relates to an attack strength of 0. In contrast, the data presented for our approach is independent of attack strength – i.e. the same results were obtained for all attack strengths shown in Table 1, including the case where no attack was present.

The trends observed for both datasets were similar. We can see that the accuracy and coverage provided by our approach matched that of k–NN for $m \geq 150$ for MovieLens and $m \geq 24$ for Smart Radio – thereby validating our neighbour selection strategy in choosing neighbours that contain less popular items. At lower values of m, many test users were unable to form neighbourhoods (or only small-sized neighbourhoods) and thus the accuracy and coverage provided our by approach was diminished.

5.2.3 Robustness

The strategy underlying our product push attack was to choose popular items to build the attack profiles to (a) ensure a high degree of similarity between genuine and attack profiles and (b) minimise the number of different attack profiles required to target as many genuine users as possible. However, neighbour selection using profile utility favours profiles that contain less popular items. Even if attack profiles contained some unpopular items, the effect is not likely to be significant since, by definition, unpopular items have proportionally less probability of being co-rated by users. In our simulations, all attack profiles were filtered from neighbourhoods for the range of threshold values m presented in Tables 2 and 3. Thus, our approach was completely robust against attack with MAPE $= 0$. For $m > 400$ for MovieLens and $m > 50$ for Smart Radio, the systems were no longer robust because attack profiles began to appear in neighbourhoods, underlining the importance of including the threshold value m in the computations.

Choosing a threshold value. The value we select needs to ensure robustness against attack and provide accuracy and coverage comparable to k–NN. Our results indicate a trade-off exists between these criteria: low values for m offer increased robustness while higher values achieve the desired accuracy and coverage. For MovieLens, we see that a range of values for m exists that satisfies these criteria: i.e. $150 \leq m \leq 400$. If we express this range of values as a ratio of the number of users present in the dataset, we obtain the range $(0.16, 0.42)$. Similarly, for EachMovie and Smart Radio, we obtain $(0.18, 0.43)$ and $(0.38, 0.79)$ respectively. For MovieLens and EachMovie, which are of comparable size, the window is similar. For Smart Radio, the window is shifted upward, which we suspect is due to its much smaller size. For these particular datasets, the results indicate that significant windows exist in which to choose suitable values for m. However, further analysis is required using other datasets to confirm these findings.

Table 2. MovieLens. Comparison in performance between the benchmark k–NN algorithm and the profile utility with similarity weight transformation algorithm for various values of the threshold (m). Note that the performance of k–NN is independent of m.

	k–NN (Attack Strength=0)		Profile Utility (Attack Strength $\geq$ 0)		
Threshold (m)	Cov (%)	Acc (MAE)	Cov (%)	Acc (MAE)	Robust. (MAPE)
5	91.6	0.788	32.8	0.981	0
10	91.6	0.788	66.0	0.896	0
20	91.6	0.788	82.9	0.842	0
40	91.6	0.788	89.4	0.809	0
60	91.6	0.788	91.0	0.797	0
80	91.6	0.788	91.4	0.793	0
100	91.6	0.788	91.4	0.790	0
150	91.6	0.788	91.6	0.788	0
250	91.6	0.788	91.6	0.788	0
400	91.6	0.788	91.6	0.788	0

Table 3. Smart Radio. Comparison in performance between the benchmark k–NN algorithm and the profile utility with similarity weight transformation algorithm for various values of the threshold (m). Note that the performance of k–NN is independent of m.

	k–NN (Attack Strength=0)		Profile Utility (Attack Strength $\geq$ 0)		
Threshold (m)	Cov (%)	Acc (MAE)	Cov (%)	Acc (MAE)	Robust. (MAPE)
3	71.8	0.986	63.7	1.041	0
6	71.8	0.986	70.9	0.996	0
9	71.8	0.986	71.7	0.992	0
12	71.8	0.986	71.8	0.988	0
15	71.8	0.986	71.8	0.987	0
18	71.8	0.986	71.8	0.987	0
21	71.8	0.986	71.8	0.987	0
24	71.8	0.986	71.8	0.986	0
27	71.8	0.986	71.8	0.986	0
30	71.8	0.986	71.8	0.986	0
40	71.8	0.986	71.8	0.986	0
50	71.8	0.986	71.8	0.986	0

6 Discussion

Various other approaches to neighbourhood formation have been proposed. For example, [10] proposes clustering as an efficient neighbourhood formation strategy. The set of users is partitioned into clusters in an off-line process. To make a prediction for a given user, the entire cluster containing the user becomes the neighbourhood. Hence, the performance of the algorithm depends on the (fixed) size of the clusters. However, this algorithm is prone to attack unless the clustering algorithm succeeds in clustering all attack profiles into one cluster. By varying the items which make up the attack profile, a careful attacker can ensure that this is unlikely to occur.

The usefulness of inverse item popularity has also been recognised by other work. In [7], a modified similarity measurement is proposed which calculates the correlation between users using only a fixed number of the least popular co-rated items. However, this algorithm does not apply a popularity threshold and therefore attack profiles are not necessarily excluded from neighbourhoods. In addition, the efficiency of this algorithm depends on the number of users in a system and is thus susceptible to scalability problems.

7 Conclusions and Future Work

In this paper, we have proposed novel profile utility and similarity weight transformation schemes for ACF. We have shown that our approach provides performance similar to the benchmark k–NN algorithm in terms of accuracy and coverage, while significantly improving efficiency and scalability. In addition, our approach is robust against the attack considered, whereas k–NN is vulnerable.

In future work, we will investigate securing ACF against other attack types – for example, by probing a system and using the recommendations obtained from a system to build attack profiles. In addition, we will extend our analysis by considering other techniques to model profile utility and to transform similarity weight measurements. For example, different approaches to quantify item popularity and rating distributions are outlined in [1, 7]. However, from our preliminary work carried out to date, profile utility defined in terms of item entropy did not perform as well as when profile utility was modeled using item popularity. Further analysis is required to understand these findings, but we believe that the differences in item entropy across the items in the datasets we evaluated were insufficient to ensure robustness against attack.

A limitation of our approach is that sophisticated attackers, familiar with our algorithm, might attempt to defeat the system by inserting large numbers of unpopular items into attack profiles. Additional work needs to be carried out to quantify this threat. However, given our algorithm, such a strategy does not guarantee success and, importantly, imposes a substantially increased cost on the attacker.

REFERENCES

[1] John S. Breese, David Heckerman, and Carl Kadie, 'Empirical analysis of predictive algorithms for collaborative filtering', in *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI '98), San Francisco, CA, USA*, pp. 43–52, (July 1998).

[2] John Canny, 'Collaborative filtering with privacy via factor analysis', in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, Tampere, Finland*, pp. 238 – 245, (2002).

[3] Jonathan Herlocker, Joseph Konstan, Al Borchers, and John Riedl, 'An algorithmic framework for performing collaborative filtering', in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, Berkeley, CA, USA*, pp. 230–237, (August 1999).

[4] http://movielens.umn.edu/.

[5] Michael P. O'Mahony, Neil J. Hurley, and Guenole C. M. Silvestre, 'Collaborative filtering – safe and sound?', in *Proceedings of the 14th International Symposium on Methodologies for Intelligent Systems (ISMIS), 2003, Maebashi City, Japan*, (October 2003).

[6] Michael P. O'Mahony, Neil J. Hurley, and Guenole C. M. Silvestre, 'An evaluation of the performance of collaborative filtering', in *Proceedings of the 14th Irish International Conference on Artificial Intelligence and Cognitice Science (AICS), 2003, Dublin, Ireland*, pp. 164–168, (September 2003).

[7] Rachael Rafter and Barry Smyth, 'Item selection strategies for collaborative filtering', in *Proceedings of the the 18th International Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, Mexico*, (2003).

[8] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J.Riedl, 'Grouplens: An open architecture for collaborative filtering of netnews', in *Proceedings of the ACM Conference on Computer Supported Cooperative Work, Chapel Hill, NC, USA*, pp. 175–186, (October 1994).

[9] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Riedl, 'Analysis of recommendation algorithms for e-commerce', in *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC-00), Minneapolis, MN, USA*, pp. 158–167, (October 2000).

[10] B.M. Sarwar, G. Karypis, J. Konstan, and J. Riedl, 'Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering', in *Proceedings of the Fifth International Conference on Computer and Information Technology (ICCIT 2002)*, (2002).