Catch Me if You Can: Detecting Blackmarket-based Collusive Followers in Twitter (Supplementary)

Table 1: Important notations and denotations.

Notation	Denotation
m	Number of users
n	Number of topics
$M_{m\times n}$	User-topic matrix indicating the probability of a user following other users of different topics
τ_i	Topic i represented by 10 words
Fr^u	Friends of user u
Fol^u	Followers of user u
Rtw^u	Retweeters of the tweets posted by user u
Twt^u	Tweets posted by user u
U^t	Set of users at time t
W^t	Set of topics at time t
S_K	$\{s_u^t\}$, collusiveness score of top K users
A	User-user matrix based on aggressiveness of (un)following activity
B	User-user matrix based on retweeters who are also followers of users
C	User-user matrix based on topical similarity with their friends
$\mathbf{U}^t$	embedding space of users till time t
$\mathbf{W}^t$	embedding space of topics till time t
$\mathbf{CF^t}$	$\{\mathbf{U}^t, \mathbf{W}^t\}$, the model we want to learn
k	Size of the latent space
K	Number of collusive users returned by the model
M^t	Configuration of M till time t

1 PROOF OF THEOREMS

Proof of Theorem 1:

Proof. To prove that $\mathbf{L}$ is convex (in terms of one variable), we show its double derivative to be positive. We prove that $\mathbf{L}$ is convex w.r.t. u subject to w being constant.

$$\frac{\partial^2 \mathbf{L}}{\partial u^2} = \sum_{j=1}^{n} (\mathbf{w}_j^t)^2 + \alpha_p \sum_{i=1}^{m}\sum_{j>i}^{m} a^t{}_{i,j} + \alpha_q \sum_{i=1}^{m}\sum_{j>i}^{m} b^t{}_{i,j} + \alpha_r \sum_{i=1}^{m}\sum_{j>i}^{m} c^t{}_{i,j} + \lambda_u \mathbf{U}^t + \lambda_w \mathbf{W}^t \quad (1)$$

Here, all entries are positive, and hence complete summation is positive. Similarly, we can prove the same w.r.t. w, keeping u as constant. □

Proof of Theorem 2:

Proof. We prove that $\mathbf{L}$ is quadratic w.r.t u subject to w being constant. Let us replace w by k, a constant.

$$\mathbf{L} = \arg\min_{\mathbf{u}^t, \mathbf{w}^t} \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}(m_{i,j}^t - \mathbf{u}_i^{t\intercal} k)^2 + \frac{\alpha_p}{2}\sum_{i=1}^{m}\sum_{j>i}^{m} a^t{}_{i,j}\|\mathbf{u}_i^t - \mathbf{u}_j^t\|^2 + \frac{\alpha_q}{2}\sum_{i=1}^{m}\sum_{j>i}^{m} b^t{}_{i,j}\|\mathbf{u}_i^t - \mathbf{u}_j^t\|^2 + \frac{\alpha_r}{2}\sum_{i=1}^{m}\sum_{j>i}^{m} c^t{}_{i,j}\|\mathbf{u}_i^t - \mathbf{u}_j^t\|^2 + \frac{\lambda_u}{2}\|\mathbf{U}^t\|^2 + \frac{\lambda_w}{2}\|k\|^2 \quad (2)$$

Each of the components in the above equation is quadratic. Thus the complete function is quadratic.

Similarly, we can prove w.r.t. w, keeping u as constant. □

Proof of Proposition 1:

Proof. Since the convergence of updating rules of U and W can be guaranteed in Non-negative Matrix Factorization [3], we here only study the convergence of Equation 2 (in the main text) under the influence of lasso. Let us denote the solution of $\mathbf{L}$ at the iteration round t as $L(M, U, W)$. For the other variables being fixed with the value solved in the $(t-1)^{th}$ step, the minimization of Equation 2 w.r.t. lasso at the iteration round t turns into:

$$\mathbf{L} = \arg\min_{\mathbf{u}^t, \mathbf{w}^t} \frac{\alpha_p}{2}\sum_{i=1}^{m}\sum_{j>i}^{m} a^t{}_{i,j}\|\mathbf{u}_i^t - \mathbf{u}_j^t\|^2 + \frac{\alpha_q}{2}\sum_{i=1}^{m}\sum_{j>i}^{m} b^t{}_{i,j}\|\mathbf{u}_i^t - \mathbf{u}_j^t\|^2 + \frac{\alpha_r}{2}\sum_{i=1}^{m}\sum_{j>i}^{m} c^t{}_{i,j}\|\mathbf{u}_i^t - \mathbf{u}_j^t\|^2 + \frac{\lambda_u}{2}\|\mathbf{U}^t\|^2 \quad (3)$$

which is a standard convex loss function. Therefore, we can deduce that, at t, the solution satisfies $L(M^t, U^t, W^t) \leq L(M^{t-1}, U^{t-1}, W^{t-1})$ with $W^t = W^{t-t}$. Therefore, $L(M^t, U^t, W^t)$ will monotonically decrease and the alternating iterations converge. □

Proof of Proposition 2:

Proof. Here, we use "proof by contradiction". We assume that given an ordinal relation, i.e., i^{th} row is more related to j^{th} row than to q^{th} row. NMF can ensure the corresponding representation u_i to be closer to u_j than to u_q. To prove this, we use the following counter example.

Given that,

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0.9 & 1 & 1 & 1 & 1 \\ 1 & 3 & 1 & 4 & 1 \end{bmatrix}$$

of which, the first three rows m_i, m_j or m_q are corresponding to i, j and q, respectively. According to $M = UW$, we obtain U and W:

$$U = \begin{bmatrix} 0.4413 & 0.5042 & 0.1377 \\ 0.0642 & 0.5211 & 0.3449 \\ 0.3550 & 0.4858 & 0.1934 \\ 1.4064 & 0.0233 & 1.220 \end{bmatrix}$$

$$W = \begin{bmatrix} 0.2661 & 0.9419 & 0.3476 & 1.0476 & 0.2785 \\ 1.5532 & 0.8432 & 1.6137 & 0.4657 & 1.6166 \\ 0.4821 & 1.3544 & 0.3875 & 2.0592 & 0.4675 \end{bmatrix}$$

Hence the distance between i and j in original space is smaller than the distance between i and q, i.e., $\|m_i - m_j\|^2 < \|m_i - m_q\|^2$. Whereas the distance between i and j, as well as i and q in representation space are $\|u_i - u_j\|^2 = 0.1854$ and $\|u_i - u_q\|^2 = 0.0109$, respectively. Therefore, it is easy to check that given i is more related to j than to q, $\|u_i - u_j\|^2 > \|u_i - u_q\|^2$, which means that the assumption does not hold. Therefore, Proposition 2 is proven. □

2 SPOTCU ALGORITHM

Input: A^t, B^t, C^t, $\mathbf{U}^{t-1}$, $\mathbf{W}^{t-1}$, S_K^{t-1}, $M^{t-1} \cup \{m_{i,j}\}$
Output: $\mathbf{U}^t$, $\mathbf{W}^t$, S_K^t
Initialization:
if u_i *is new* **then**
  Append new column to the right side of $\mathbf{U}^{t-1}$;
  Initialize $\mathbf{u}_i$, the u_i column in $\mathbf{U}^{t-1}$;
end
if w_j *is new* **then**
  Append a new column to the right side of $\mathbf{W}^{t-1}$;
  Initialize $\mathbf{w}_j$, the w_j column in $\mathbf{W}^{t-1}$;
end
Updation:
repeat
  for $u_i \in \mathbf{U}^t$ **do**
    Update $\mathbf{u}_i^t$ according to Eq. 3 (main text)
  end
  for $w_j \in \mathbf{W}^t$ **do**
    Update $\mathbf{w}_j^t$ according to Eq. 3 (main text)
  end
until *Convergence or maximum iterations reached.*;
Final Detection:
foreach (u_i, u_k) **do**
  $s_{i,k}^t = \frac{1}{1+\|\mathbf{u}_i^t - \mathbf{u}_k^t\|^2}$;
  $s_i^t = \max(s_{i,k}^t | u_k^t \in U^t)$;
  Insert $\{s_i^t, u_i\}$ into S_K^{t-1};
end

Algorithm 1: SpotCU: Spotting collusive users.

Algorithm 1 shows the pseudocode of SpotCu.

Time Complexity: As shown, for each of the above specified activities there would be changes in the M matrix, depending on the topic(s) being affected by the activities, and new embedding spaces are computed. The time complexity of SpotCU is $O(|U^t| \log K)$ for reporting K collusive users, while for updation it takes $O(|W^t + U^t|^2 \times K \times Iter)$, where $Iter$ is the number of iterations until convergence. The former factor comes from a min-heap which is used to report the K topmost collusive users, where each insertion takes $O(\log K)$. The min-heap of top K collusive users contains the user with least score among K at the top; we evict the topmost collusive user and insert current user while iterating only if his/her score is less as compared to current user. We perform the updation for each user, where each update takes $O(|W^t + U^t| \times K \times Iter)$ and hence comes the latter factor.

3 DISTRIBUTION OF FOLLOWERS/FRIENDS

Figure 1 shows the distribution of follower and friend counts for collusive and genuine users. We notice that the follower distribution is almost same for both types of users. This ensures that our dataset is unbiased on the follower count.

4 CATEGORIZING COLLUSIVE USERS

We further delve deeper into the ground-truth set of collusive users and categorize them based on their topic of interest to understand which classes of people mostly approach blackmarket agencies to increase their social reputation. We leverage the timeline information – we aggregate all the tweets posted/retweeted by a user to form a document for that user, run pre-trained Doc2Vec [2], and map users into latent space. Following this, we run k-means algorithm to cluster collusive users (we use the elbow method to determine the optimal number of clusters for k-means). Out of 7 clusters detected by k-means, we manually inspect four large clusters and present their corresponding word clouds in Figure 2. A careful inspection reveals the identify of the prominent clusters (size within parenthesis for Dataset I, Dataset II): promotion (24%, 21.6%), entertainment (20.6%, 17.4%), sports (19.6%, 9.3%), and politics (13.3%, 14.9%). Other clusters are difficult to interpret. This result indicates that users generally approach blackmarkets to kickstart their accounts by gaining followers which in turn helps them reach out to many people to promote their ideas or products.

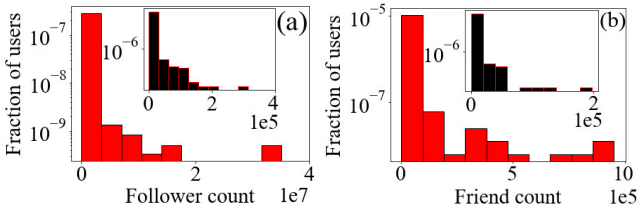

Figure 1: Distribution of (a) follower and (b) friend counts for collusive (black, inset) and genuine (red) users. Values in the y-axis are spaced logarithmically for better visualization.

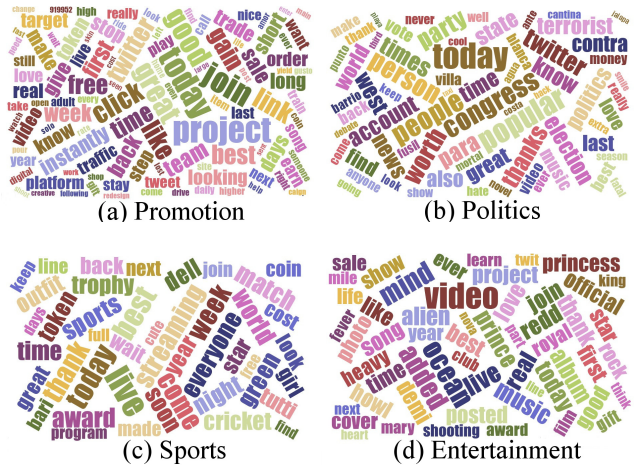

Figure 2: Word cloud of four large clusters of collusive users in Dataset I.

5 ISPOTCU: AN INCREMENTAL VERSION OF SPOTCU

The optimization process in Algorithm 1 requires several iterations through the dataset to train the model. The model produces the optimal solution when the entire dataset is provided beforehand. However, in real-world scenario, data arrive in streaming fashion, and the number of observations (Twitter users in this case) increases over time. It would be computationally expensive and

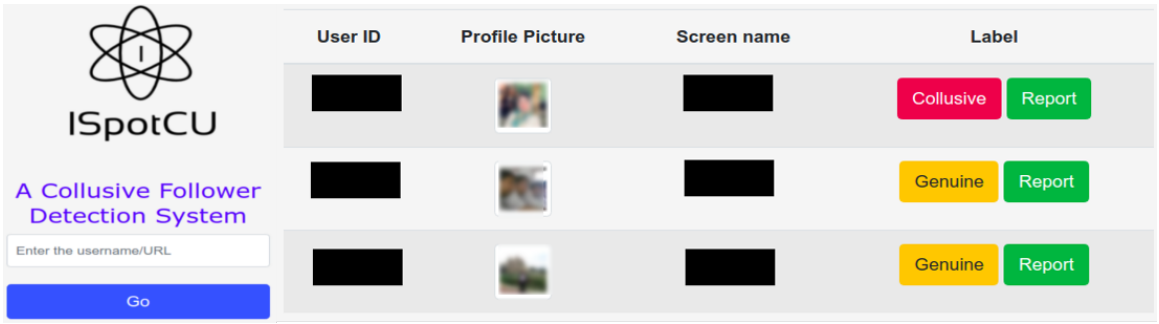

Figure 3: Home page of our online tool.

memory intensive to run the model every time from the scratch upon receiving each user information.

In order overcome this problem for designing a real-world system, we propose ISpotCU, an incremental version of SpotCU. ISpotCU optimizes parameters at the element level — optimizing each coordinate of the latent vector, keeping the other element constant [1]. By setting the derivative of Equation 2 (in the main text) to 0, we obtain the solution of $u^t_{i,f}$ and $w^t_{j,f}$:

$$\mathbf{u}^t_{i,f} = \frac{(m_{i,j} \cdot w^t_{j,f}) + \hat{Q_u}}{Q_u}, \quad \mathbf{w}^t_{j,f} = \frac{(m_{i,j} \cdot u^t_{i,f})}{Q_w} \tag{4}$$

$$Q_u = \lambda_u + \sum_{j=1}^{n} w^{2t}_{j,f} + (\alpha_p \sum_{j=1}^{m} a^t_{i,j} + \alpha_q \sum_{j=1}^{m} b^t_{i,j} + \alpha_r \sum_{j=1}^{m} c^t_{i,j})$$

$$\hat{Q_u} = (\alpha_p \sum_{j=1}^{m} a^t_{i,j}\mathbf{u}^t_{j,f} + \alpha_q \sum_{j=1}^{m} b^t_{i,j}\mathbf{u}^t_{j,f} + \alpha_r \sum_{j=1}^{m} c^t_{i,j}\mathbf{u}^t_{j,f}) \tag{5}$$

$$Q_w = \lambda_w + \sum_{i=1}^{m} u^{2t}_{i,f}$$

Here, $u_{i,f}$ represents f^{th} element of i^{th} latent user representation learned, and $w_{j,f}$ represents f^{th} element of j^{th} latent topic representation learned.

For memory efficiency, we utilise two caches which store Q_u, $\hat{Q_u}$ and Q_w, respectively. The pseudocode of ISpotCU is shown in Algorithm 2. It performs the updation of embedding space of only the number of users involved in the activity being considered. The underlying assumption is that the new interaction should not change $\mathbf{U^t}$ and $\mathbf{W^t}$ too much from a global perspective, while it should change the local features for $\mathbf{U^t}$ and $\mathbf{W^t}$ significantly. For the stopping criteria, our empirical study shows that one iteration is usually sufficient to obtain good results. Moreover, it is important to note that after updating a latent vector, we need to update the cache C accordingly.

We observe in Table 2 that in most cases, the performance gain is achieved after the first iteration. As the number of iterations increases, there is not much improvement in the performance. The intuition behind this observation is that the local features are given importance upon observing them, instead of considering the entire history. Thus, one iteration is enough for ISpotCU to learn from a new interaction incrementally, making ISpotCU very efficient for learning online.

Figure 3 shows the screenshot of our online tool. The end user needs to submit the Twitter screen name or URL of the user under

Input: A^t, B^t, C^t, $\mathbf{U}^{t-1}$, $\mathbf{W}^{t-1}$, S_K^{t-1}, $M^{t-1} \cup \{m_{i,j}\}$
Output: $\mathbf{U}^t$, $\mathbf{W}^t$, S_K^t
Initialization:
if u_i *is new* **then**
  Append new column to the right side of $\mathbf{U}^{t-1}$;
  Initialize $\mathbf{u}_i$, the u_i column in $\mathbf{U}^{t-1}$;
end
if w_j *is new* **then**
  Append a new column to the right side of $\mathbf{W}^{t-1}$;
  Initialize $\mathbf{w}_j$, the w_j column in $\mathbf{W}^{t-1}$;
end
Updation:
Update $\mathbf{u}_i^t$ according to Eq. 4
Update $\mathbf{w}_j^t$ according to Eq. 4
Update cache according to Eq. 5
Final Detection:
foreach (u_i, u_k) **do**
  $s^t_{i,k} = \frac{1}{1+\|\mathbf{u}^t_i - \mathbf{u}^t_k\|^2}$;
  $s^t_i = \max(s^t_{i,k} | u^t_k \in U^t)$;
  Insert $\{s^t_i, u_i\}$ into S_K^{t-1};
end

Algorithm 2: ISpotCU: An incremental algorithm to spot collusive users in real time.

# of iterations	Dataset I	Dataset II
1	0.778	0.796
2	0.780	0.799
3	0.781	0.804
4	0.783	0.806
5	0.783	0.806

Table 2: Change in F_1 score of ISpotCU with the number of iterations on two datasets. We keep 90% data for developing the model. We notice that the change in accuracy is insignificant across consecutive iterations.

inspection. The data is sent as a HTTP POST request to our backend server. The backend server receives the details of the user entered and collects its followers using the Twitter API. It then labels each follower as collusive or genuine using ISpotCU. The labels are shown to the end users in a tabular format. A 'report' button is also associated with each label, upon clicking of which the feedback is sent to our backend server. Currently, the backend of the tool is developed using Python-Flask[1]. The response of the server is in

[1]http://flask.pocoo.org/

JSON format which also serves as a REST API. One can request our API to also check multiple accounts, up to a rate limit. The online tool is deployed in a server with 32 GB RAM, 12 cores, 2.7GHz CPU running Ubuntu 16.04 operating system.

User Study: Along with the quantitative evaluation, we also conducted a user study with the help of 50 volunteers. First, we randomly assigned 50 Twitter users to each volunteer. Volunteers visited the Twitter profile of each user, clicked on the follower list (each follower was labeled as 'Genuine' or 'Collusive' by the tool) and marked the wrong label. We recorded the responses of the volunteers and measured the average accuracy. The results shows that our online tool is highly accurate, achieving an average accuracy of 83.5% (with standard deviation of 0.03).

REFERENCES

[1] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast Matrix Factorization for Online Recommendation with Implicit Feedback.. In *SIGIR*, Raffaele Perego, Fabrizio Sebastiani, Javed A. Aslam, Ian Ruthven, and Justin Zobel (Eds.). ACM, 549–558.

[2] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International Conference on Machine Learning*. 1188–1196.

[3] Daniel D. Lee and H. Sebastian Seung. 2001. Algorithms for Non-negative Matrix Factorization. In *Advances in Neural Information Processing Systems 13*, T. K. Leen, T. G. Dietterich, and V. Tresp (Eds.). MIT Press, 556–562. http://papers.nips.cc/paper/1861-algorithms-for-non-negative-matrix-factorization.pdf