Discovering Opinion Spammer Groups by Network Footprints

Junting Ye Leman Akoglu

Department of Computer Science, Stony Brook University
{juyye,leman}@cs.stonybrook.edu

Abstract. Online reviews are an important source for consumers to evaluate products/services on the Internet (e.g. Amazon, Yelp, etc.). However, more and more fraudulent reviewers write fake reviews to mislead users. To maximize their impact and share effort, many spam attacks are organized as campaigns, by a *group* of spammers. In this paper, we propose a new two-step method to discover spammer groups and their targeted products. First, we introduce NFS (Network Footprint Score), a new measure that quantifies the likelihood of products being spam campaign targets. Second, we carefully devise *GroupStrainer* to cluster spammers on a 2-hop subgraph induced by top ranking products. We demonstrate the efficiency and effectiveness of our approach on both synthetic and real-world datasets from two different domains with millions of products and reviewers. Moreover, we discover interesting strategies that spammers employ through case studies of our detected groups.

Keywords: opinion spam · spammer groups · spam detection · graph anomaly detection · efficient hierarchical clustering · network footprints

1 Introduction

Online reviews of products and services are an increasingly important source of information for consumers. They are valuable since, unlike advertisements, they reflect the testimonials of other, "real" consumers. While many positive reviews can increase the revenue of a business, negative reviews can cause substantial loss. As a result of such financial incentives, opinion spam has become a critical issue [17], where fraudulent reviewers fabricate spam reviews to unjustly promote or demote (e.g., under competition) certain products and businesses.

Opinion spam is surprisingly prevalent; one-third of consumer reviews on the Internet[1], and more than 20% of reviews on Yelp[2] are estimated to be fake. Despite being widespread, opinion spam remains a mostly open and challenging problem for at least two main reasons; (1) humans are incapable of distinguishing fake reviews based on text [25], which renders manual labeling extremely difficult and hence supervised methods inapplicable, and (2) fraudulent reviewers are

[1] http://www.nytimes.com/2012/08/26/business/book-reviewers-for-hire-meet-a-demand-for-online-raves.html

[2] http://www.businessinsider.com/20-percent-of-yelp-reviews-fake-2013-9

often professionals, paid by businesses to write detailed and genuine-looking reviews.

Since the seminal work by Jindal and Liu [17], opinion spam has been the focus of research for the last 7-8 years (Section 5). Most existing work aim to detect *individual* spam reviews [12, 17, 20, 21, 25, 26] or spammers [1, 11, 13, 18, 22, 26]. However, fraud/spam is often a *collective* act, where the involved individuals cooperate in groups to execute spam *campaigns*. This way, they can increase total impact (i.e., dominate the sentiments towards target products via flooding deceptive opinions), split total effort, and camouflage (i.e., hide their suspicious behaviors by balancing workload so that no single individual stands out). Surprisingly, however, only a few efforts aim to detect group-level opinion spam [23, 27, 28]. Moreover, most existing work employ supervised techniques [17, 12, 25, 20] and/or utilize side information, such as behavioral [17, 22, 23, 27, 28] or linguistic [12, 25] clues of spammers. The former is inadmissible, due to the difficulty in obtaining ground truth labels. The latter, on the other hand, is not adversarially robust; the spammers can fine-tune their language (e.g., usage of superlatives, self-references, etc.) and behavior (e.g., login times, IPs, etc.) to mimic genuine users as closely as possible and evade detection.

In this work, we propose a new unsupervised and scalable approach for detecting opinion spammer groups solely based on their network footprints. At its heart, our method consists of two key components:

- NFS **(Network Footprint Score).** We introduce a new graph-based measure that quantifies the statistical distortions caused by spamming activities in well-understood network characteristics. NFS is fast to compute and more robust to evasion than linguistic and behavioral measures, provided that spammers have only a partial view of the review network (Section 2).
- ***GroupStrainer.*** We devise a fast method to group spammers on a carefully induced subnetwork of highly suspicious products and reviewers. *GroupStrainer* employs a hierarchical clustering algorithm that leverages similarity-sensitive hashing to speed up the merging steps. The output is a set of spammer groups and their nested hierarchy, which facilitates sensemaking of their organizational structure, as well as verification by end analysts (Section 3).

In the experiments, we compare our method to various graph anomaly detection methods on synthetic datasets and study its performance on two large real-world datasets (`Amazon` and `iTunes`). Our results show that we effectively spot spammer groups with high accuracy, which is superior to existing methods.

2 Measuring Network Footprints

A perfect form of an opinion spam campaign would certainly reflect a near-replica of the characteristics that genuine reviewers exhibit on online review sites. Those characteristics include linguistic, behavioral, and relational (network-level) patterns. One can argue that language and behavior patterns are relatively easier to mimic by the spammers as compared to network-based patterns. For example,

spammers could adjust their usage of certain language constructs (e.g., superlatives, self-references, etc.) that have been found to be associated with deception [25], so as to evade classifiers [8].

On the other hand, spoofing network-level characteristics in general is adversarially harder for various reasons. First, spammers (adversaries) often do not have a complete view of the entire review network, due to its sheer scale and access properties. Moreover, the reviewers that belong to a spam campaign have to act as a group, which would create different dynamics in the review network than the independent actions of genuine reviewers. Finally, spammers do not replicate "unimportant" structures in the network due to limited budget.

Therefore, in this work we focus on the network-level characteristics of opinion spammers in the user–product bipartite review network. We develop a network-based method that is able to distinguish the network footprints of spammer groups from those of individual genuine users. In particular, we propose a new measure called the NFS (for Network Footprint Score), that quantifies the extent to which statistical network characteristics of reviewers are distorted by spamming activities.

To design our NFS measure, we leverage two key observations associated with real-world networks:

1. **Neighbor diversity:** The neighbors of a node in a real network are expected to consist of nodes with varying behavior and levels of activity. As such, the neighbors should not be overly dependent on one another; rather, they should be spread across sources of varying quality or importance. For example, in social networks a person has friends with varying levels of "popularity".
2. **Self-similarity:** Real-world networks are self-similar [4, 5]; that is, portions of the network often have properties similar to the entire network. In particular, the importances of the neighbors of a node follow a skewed, power-law-like distribution, just as the case for all of the nodes in the entire network.

In a nutshell, while the former observation implies (*i*) local diversity of node importances in the neighborhood of a node, the latter implies (*ii*) distributional similarity between node importances at local (i.e., neighborhood) and global level (i.e., at large). Importance of nodes in a network can be captured by their centrality. There exist a long list of work on developing various measures for node centralities [10]. In this work, we consider two different measures; degree and Pagerank centrality, which respectively use local and global information to quantify node importances.[3]

In the following, we describe how we utilize each of the above insights to measure the network footprints of spammer groups in review networks. More precisely, a review network is a bipartite graph G that consists of n reviewer nodes connected to m product nodes through review relations.

[3] We compute these measures based on the reviewer–product bipartite review network.

2.1 Neighbor diversity of nodes

We can translate Observation 1 above to the domain of review networks as follows. An honest set of reviewers for a product arises by independent actions of individuals with varying behavior and levels of activity. As a result, the centrality of the reviewers of a product is expected to vary to a large extent. In analogy to social networks where a person has friends with varying level of "popularity", a product would have reviewers with varying level of network centrality.

In other words, a set of reviewers all with centrality (degree or Pagerank) values falling into a narrow interval is suspicious. Such a large set of highly similar reviewers (i.e., "clones") raises the suspicion that they have emerged through certain means of a *cooperation*, e.g., under a spam campaign.

To quantify the diversity of neighbor centralities of a given product, we first split the centrality values of its reviewers into buckets to create a non-parametric estimation of their density through a histogram. We then compute the skewness of the histogram through entropy. More specifically, given the (degree of Pagerank) centralities $\{c_1^{(i)}, \ldots, c_{deg(i)}^{(i)}\}$ of the reviewers of a product i with degree $deg(i)$, we create a list of buckets $k = \{0, 1, \ldots\}$. We let the bucket boundary values grow exponentially as $a \cdot b^k$, as both degree and Pagerank values of nodes in real-world networks have been observed to follow skewed distributions [6, 9].

For degree centrality we use $a = 3$ and $b = 3$ such that the bucket boundaries are $\{1, 3, 9, 27, \ldots\}$ and place each reviewer j with degree $c_j^{(i)}$ to bucket k with $a \cdot b^{k-1} \leq c_j^{(i)} < a \cdot b^k$. On the other hand, we use $a = 0.3$ and $b = 0.3$ for Pagerank where bucket boundaries become $\{1, 0.3, 0.09, 0.027, \ldots\}$ as it takes values in $[0, 1]$, and place each reviewer j with Pagerank $c_j^{(i)}$ to bucket k with $a \cdot b^{k-1} \geq c_j^{(i)} > a \cdot b^k$. The choice of a and b has little effect on our results as long as we use a logarithmic binning [24] so as to capture the skewness of data.

Given the placement of reviewers into K buckets by their centrality, we next count the reviewers in each bucket and normalize the counts by the total count $deg(i)$ to obtain a discrete probability distribution $P^{(i)}$ with values $[p_1^{(i)}, \ldots, p_K^{(i)}]$. We then compute the Shannon entropy of $P^{(i)}$ as $H_c(i) = -\sum_{k=1}^{K} p_k^{(i)} \log p_k^{(i)}$ for centrality c. As such, a product i receives two neighbor-diversity scores, $H_{deg}(i)$ and $H_{pr}(i)$, for degree and Pagerank respectively. The lower these scores are, the more likely the product is the target of a spam campaign and hence the more suspicious are the reviewers of the product as they appear near-replicas of one another in the network—cooperating around the same goal, leaving similar network footprints.

2.2 Self-similarity in real-world graphs

Similarly, we can leverage Observation 2 to measure the distributional distortions caused by spam activities. Specifically, self-similarity implies that the centrality of reviewers for a particular product should follow a similar distribution as to the centrality of all reviewers in the network. Note that while Observation 1 enforces the neighbor centralities to be diverse, Observation 2 requires them to also closely

follow global distributional patterns. It is well-known that degree and Pagerank distributions of nodes in real-world graphs follow power-law-like distributions [6, 9]. As such, a diverse but e.g., Gaussian-distributed set of neighbor centralities would still raise a red flag in terms of self-similarity, while considered normal in terms of neighbor diversity.

Therefore, we define a second type of score for each product i as the KL-divergence between the histogram density of the centralities of its reviewers $P^{(i)} = [p_1^{(i)}, \ldots, p_K^{(i)}]$ and that of all reviewers in the network denoted by Q; $KL_c(P^{(i)} \| Q) = \sum_k p_k^{(i)} \log \frac{p_k^{(i)}}{q_k}$. We compute Q in the same way we computed P's as before, where this time we split into buckets the centrality values of all the reviewers in the network.[4] As a result, a product i receives two scores for divergence from self-similarity, $KL_{deg}(i)$ and $KL_{pr}(i)$, for degree and Pagerank respectively. The larger these scores are, the more likely the product is the target of a spam campaign.

2.3 NFS measure

To quantify the extent to which a product is under attack by a spam campaign, we combine the scores derived from the network footprints into a final score. Overall, we have four suspiciousness scores for a product, two based on neighbor-diversity; H_{deg} and H_{pr}, and two based on self-similarity; KL_{deg} and KL_{pr}. These capture different semantics; a product is likely a target the lower the H and the higher the KL scores. Moreover, they are not normalized within a standard range.

To unify the scores into a single score with a standard scale, we leverage the cumulative distribution function (CDF). In particular, let us denote by $\mathcal{H} = \{H(1), H(2), \ldots\}$ the list of entropy scores (based on degree or Pagerank) we computed for a set of products. To quantify the extremity of a particular $H(i)$, we use the empirical CDF over $\mathcal{H}$ and estimate the probability that the set contains a value as *low* as $H(i)$ as

$$f(H(i)) = P(H \leq H(i)) \ ,$$

which is equal to the fraction of scores in $\mathcal{H}$ that are less than or equal to $H(i)$. On the other hand, for KL scores, we estimate the probability that the set $\mathcal{KL} = \{KL(1), KL(2), \ldots\}$ contains a value as *high* as $KL(i)$ by

$$f(KL(i)) = 1 - P(KL \leq KL(i)) \ .$$

As such, $f(\cdot)$ takes low values for low $H(i)$ values and high $KL(i)$ values. Finally, we combine the f values to compute the NFS of a product i as given in Equ. (1), such that $NFS(i) \in [0, 1]$ where high values are suspicious.

$$NFS(i) = 1 - \sqrt{\frac{f(H_{deg}(i))^2 + f(H_{pr}(i))^2 + f(KL_{deg}(i))^2 + f(KL_{pr}(i))^2}{4}} \quad (1)$$

[4] We use Laplace smoothing for empty buckets.

Algorithm 1: *Compute*NFS

Input: Reviewer–Product graph $G = (V, E)$, degree threshold η
Output: Network Footprint Score (NFS) of products with degree $\geq \eta$
Compute centrality c of each reviewer in G, $c = \{\text{degree, Pagerank}\}$
Create a list of buckets $k = \{0, 1, \ldots\}$
foreach reviewer j in G, $1 \leq j \leq n$ **do** *//Compute global histogram Q*
  if $c =$degree **then**
    place j to bucket$_k$ with $a \cdot b^{k-1} \leq c_j^{(i)} < a \cdot b^k$, ($a = 3$, $b = 3$)
  else place j to bucket$_k$ with $a \cdot b^{k-1} \geq c_j^{(i)} > a \cdot b^k$, ($a = .3$, $b = .3$)
  forall the non-empty buckets $k = \{0, 1, \ldots\}$ **do**
    $q_k = |bucket_k|/n$
foreach product i with $deg(i) \geq \eta$ **do**
  Create a list of buckets $k = \{0, 1, \ldots\}$
  foreach neighbor (reviewer) j of product i **do**
    if $c =$degree **then**
      place j to bucket$_k$ with $a \cdot b^{k-1} \leq c_j^{(i)} < a \cdot b^k$, ($a = 3$, $b = 3$)
    else place j to bucket$_k$ with $a \cdot b^{k-1} \geq c_j^{(i)} > a \cdot b^k$, ($a = .3$, $b = .3$)
  forall the non-empty buckets $k = \{0, 1, \ldots\}$ **do** *//local histogram* $P^{(i)}$
    $p_k^{(i)} = |bucket_k|/deg(i)$
  Compute entropy $H_c(i)$ based on $p_k^{(i)}$'s
  K' = number of uncommon buckets where $q_k \neq 0$ and $p_k^{(i)} = 0$
  forall the buckets k where $q_k \neq 0$ **do** *//local smoothed histogram* $P^{(i)}$
    if $p_k^{(i)} = 0$ **then** $p_k^{(i)} = 1/(deg(i) + K')$ *//Laplace smoothing*
    else $p_k^{(i)} = (p_k^{(i)} \cdot deg(i))/(deg(i) + K')$ *//Re-normalize*
  Compute divergence $KL_c(P^{(i)} \| Q)$ based on $p_k^{(i)}$'s and q_k's
  Compute NFS of i by Equ. (1) based on $H_c(i)$ and $KL_c(P^{(i)} \| Q)$

The complete list of steps to compute NFS is given in Algorithm 1. Note that we utilize centrality density distributions P_{deg} and P_{pr} over neighbors to compute the NFS of a product. These distributions are meaningful only when a product has a large number of review(er)s, since only a few data points cannot constitute a reliable distribution (in this work products with less than 20 reviews are ignored, i.e., we set $\eta = 20$). This, however, is not a severe limitation of our approach. The reason is that spam campaigns often involve reasonably large number of reviewers in order to (1) increase the total impact on a target product and (2) share the overall effort. Small spam campaigns are of little spamming power and can be overlooked without much risk.

3 Detecting Spammer Groups

We compute the NFS for the products, as a measure of their abnormality of being targeted by suspiciously similar reviewers. Such groups of highly similar reviewers potentially work together under the same spam campaigns. Our end goal is to identify all such spammer groups.

To achieve this goal, we construct a subnetwork consisting of the top products with the highest NFS values[5] denoted as P_1, all the reviewers of these products R, and all the products that these reviewers reviewed $P_2 \supseteq P_1$. In other words, the subnetwork is the induced subgraph of our original graph G on the nodes within 2-hops away from the top products in P_1, i.e. $G[P_2 \cup R]$. We represent this subgraph with a $p \times u$ adjacency matrix A, where $|P_2| = p$ and $|R| = u$.

An example of A can be seen in Figure 3 (top). This matrix contains highly similar users, i.e., columns, since the subgraph is biased toward products with high NFS values. However, it is clear that the reviewer groups are not immediately evident from the figure. To fully automate the group identification process, we propose a fast algorithm called *GroupStrainer* that finds clusters of highly similar columns of A, which carefully re-organizes/shuffles the columns to better reveal the reviewer groups. The output of *GroupStrainer* on the example matrix can be seen in Figure 3 (bottom), and will be discussed further in Section 4.

Note that the goal here is *not* to cluster all the columns of A (notice the last several in Figure 3 (bottom) that do not belong to any group), but to chip off groups where columns within each group are strongly similar. Moreover, we ideally do not want to pre-specify the number of groups a priori, which is a challenging parameter to set. To achieve both of these goals, *GroupStrainer* adopts an agglomerative hierarchical clustering scheme, where columns are iteratively merged to form larger groups. Such a scheme also reveals the nested, hierarchical structure of the groups that provides further insights to the end analyst.

A naive agglomerative clustering has $O(u^3)$ complexity, where in each step similarities of all-pairs are compared. Moreover, clusters are merged two at a time in each step. In our approach, we leverage Locality-Sensitive Hashing (LSH) [15] to speed up the process of finding similar set of clusters. Provided a set of similar clusters, we can then merge two or more clusters at a time which speeds up the hierarchy construction.

In a nutshell, LSH is a randomized algorithm for similarity search, which ensures similar points are hashed to the same hash bucket with high probability, while non-similar points are rarely hashed to the same bucket. As a result, it quickens the similarity search by narrowing down the search to points that are hashed only to the same buckets. In order to systematically reduce the probability of highly similar points hashing to different buckets, it employs multiple hash functions. At its heart, LSH generates signatures for each data point, where it ensures that the similarity between the original points is proportional to the probability of their signatures to be equal. As a consequence, the more similar two points are, the more likely their signatures match, and the more probable that they hash to the same bucket (hence locality-, or similarity-sensitive hashing). LSH uses different, suitable signature generation functions with respect to different similarity functions. In this paper, we use two: min-hashing for Jaccard similarity and random-projection for cosine similarity.

[5] Rather than an arbitrary top k, we adopt the mixture-modeling approach in [14], where NFS values are modeled as drawn from two distributions; Gaussian for normal values and exponential for outliers. Model parameters and assignment of points to distributions are learned by the EM algorithm. Top products are then the ones whose NFS values belong to the exponential distribution.

Algorithm 2: *GroupStrainer*

1 **Input:** $p \times u$ adjacency matrix A of 2-hop network of top products with highest NFS values, similarity lower bound s_L (default 0.5)
2 **Output:** User groups U^{out} and their hierarchical structure
3 Create a list of similarity thresholds $S = \{0.95, 0.90, \ldots, s_L\}$
4 Set $k^1 = u$, $U^1 := \{1, 2, \ldots, u\} \rightarrow \{1, 2, \ldots, u\}$ *// init reviewer groups*
5 **for** $T = 1$ *to* $|S|$ **do**
6 Estimate LSH parameters r and b for threshold $S(T)$
7 ***//Step 1. Generate signatures***
8 Init signature matrix $M[i][j] \in \mathbb{R}^{rb \times k^T}$
9 **if** $T = 1$ **then** *// use Jaccard similarity, generate min-hash signatures*
10 **for** $i = 1$ *to* rb **do**
11 $\pi_i \leftarrow$ generate random permutation $(1 \ldots p)$
12 **for** $j = 1$ *to* k^T **do** $M[i][j] \leftarrow min_{v \in N_j} \pi_i(v)$
13 **else** *// use cosine similarity, generate random-projection signatures*
14 **for** $i = 1$ *to* rb **do**
15 $r_i \leftarrow$ pick a random hyperplane $\in \mathbb{R}^{p \times 1}$
16 **for** $j = 1$ *to* k^T **do** $M[i][j] \leftarrow sign(rsum(A(:, U^T{=}j)/|U^T{=}j|) \cdot r_i)$
17 ***//Step 2. Generate hash tables***
18 **for** $h = 1$ *to* b **do**
19 **for** $j = 1$ *to* k^T **do** $hash(M[(h-1) \cdot r + 1{:}h \cdot r][j])$
20 ***//Step 3. Merge clusters from hash tables***
21 Build candidate groups g's: union of clusters that hash to *at least one* same bucket in all hash tables, i.e. $\{c_i, c_j\} \in g$ if $hash_h(c_i) = hash_h(c_j)$ for $\exists h$
22 **foreach** candidate group g **do**
23 **foreach** $c_i, c_j \in g$, $i \neq j$ **do**
24 **if** $sim(\mathbf{v_i}, \mathbf{v_j}) \geq S(T)$ **then** *//merge*
25 $g = g \backslash \{c_i, c_j\} \cup \{c_i \cup c_j\}$
26 $k^T = k^T - 1$, $U^T(c_i) = U^T(c_j) = min(U^T(c_i), U^T(c_j))$
27 $k^{T+1} = k^T$, $U^{T+1} = U^T$
28 **return** $U^{out} = U^{|S|+1}$, evolving groups $\{U^1, \ldots, U^{|S|}\}$ to build nested hierarchy

The details of our *GroupStrainer* is given in Algorithm 2, which consists of three main steps: (1) generate LSH signatures (Lines 7-16), (2) generate hash tables (17-19), and (3) merge clusters using hash buckets (20-24).

In the first iteration of step (1), i.e. $T = 1$, the clusters consist of individual, binary columns of A, represented by $\mathbf{v_j}$'s, $1 \leq j \leq u$. As the similarity measure, we use Jaccard similarity which is high for those columns (i.e., reviewers) with many exclusive common neighbors (i.e., products). Min-hashing is designed to capture the Jaccard similarity between binary vectors; that is, it can be shown that the probability that the min-hash values (signatures) of two binary vectors agree is equal to their Jaccard similarity (Lines 9-12). For $T > 1$, the clusters consist of multiple columns. This time, we represent each cluster j by a length-p real-valued vector $\mathbf{v_j}$ in which the entries denote the density of each row, i.e., $\mathbf{v_j} = rsum(A(:, U^T = j)/|U^T = j|$, where U^T is the mapping of columns to clusters at iteration T, $U^T = j$ depicts the indices of columns that belong to cluster j, $rsum$ is the row-sum of entries in the induced adjacency matrix, and

$|U^T = j|$ is the size of cluster j. Then, we use the cosine similarity between $\mathbf{v_i}$ and $\mathbf{v_j}$ as the similarity measure of two clusters i and j. In LSH, random-projection based signature generation captures cosine similarity; that is, it can be shown that the probability that the random-projection values of two real-valued vectors agree is proportional to their cosine similarity [19] (Lines 13-16).

In step (2), LSH performs multiple hash operations on different subset of signatures to increase the probability that two highly similar items hash to the same bucket in at least one hash table (Lines 18-19).

Step (3) involves the main merging operations. First we construct the groups of candidate clusters to be merged, where all the clusters that hash to the *same* hash bucket in *at least one* hash table are put into the same group g (Line 21). These are called candidate clusters as LSH is a probabilistic algorithm, and can yield false positives. Therefore, rather than merging all the clusters in each group directly, we verify whether their similarity is above the desired threshold (Line 24), before committing the merge (Lines 25-26). Note that at this merge step, more than two clusters can be merged. With respect to complexity *GroupStrainer* is only sub-quadratic; it performs pairwise similarity computation among clusters only within groups (Line 23), rather than among all the clusters. The number of clusters within each group at a given iteration is often much smaller than the total. This contributes to a significant reduction in running time, while enabling us to focus on merging highly similar clusters.

Finally, we describe the process of constructing a nested hierarchy of clusters. A specific iteration finds groups of clusters with similarity above a desired threshold. At the beginning ($T = 1$), we set a large/conservative threshold of 0.95 such that only extremely similar columns are grouped. As T increases, we gradually lower the similarity threshold so as to allow the hierarchy to grow. Here, the user can specify a lower bound s_L for the threshold (default 0.5), so as to prevent clusters with similarity below the bound to be merged. Depending on s_L, our hierarchy may contain a single or multiple tree(s), as well as singleton columns that do not belong to any cluster. We treat all non-singleton trees as candidates of spammer groups, and inspect them in size order. Other ranking measures can also be used to prioritize inspection.

4 Evaluation

We first evaluate the performance of our method on synthetic datasets, as compared to several existing methods. Our method consists of two steps: NFS computation and *GroupStrainer*. The former tries to capture the targeted products, and the latter focuses on detecting spammer groups through their collusion on the target products. As such, we design separate data generators for each step to best simulate these scenarios. In addition, we apply our method on two real-world review datasets and show through detailed case analyses that it detects many suspicious user groups. A summary of the datasets is given in Table 1.

Table 1. Summary of synthetic and real-world datasets used in this work.

	Synthetic Data				Real-world Data	
	Chung-Lu1	Chung-Lu2	RTG1	RTG2	`iTunes`	`Amazon`
# of users	532,742	2,133,399	604,520	876,627	966,808	2,146,074
# of products	157,768	665,381	604,805	876,950	15,093	1,230,916
# of edges	1,299,059	5,191,053	3,097,342	4,644,572	1,132,329	5,838,061

4.1 Performance of NFS on synthetic data

Synthetic data generation. We use two models to create synthetic graphs: Chung-Lu [7] and the RTG [2]. Chung-Lu creates random graphs with a given degree sequence. We draw the degrees of reviewers and products from a power-law distribution with exponent 2.9 and 2.1, respectively, as observed in the real world [1, 6, 9]. RTG model also creates realistic bipartite graphs that not only follow power-law degree distribution but also contain communities, which are common in real-world graphs. We create two graphs with different sizes using each model (Table 1). Next, we follow the injection process in [16] to simulate and inject spammer groups into our graphs. Specifically, we add 3 spammer groups with 1000, 2000 and 4000 users respectively. Each spammer group targets a set of designated products (100, 200 and 400 in size). Each injected spammer writes 20 reviews to their target products, with σ percent camouflage written to untargeted ones. There exist two strategies to camouflage: writing reviews (1) to top 100 most popular (highest degree) products; and (2) to random untargeted products. This way, we create four injection configurations; σ=10% or 30% camouflage on popular products, and σ=10% or 30% on random ones, where larger σ and random camouflage are relatively harder to detect.

Compared methods. NFS measures the suspiciousness of products. In order to rank the users, we utilize the FraudEagle method [1]. FraudEagle computes scores for users and products by propagating *unbiased* beliefs in the review network. We assign NFS values of products as their initial beliefs (i.e., priors). Thus users who targeted many products with large NFS values gain high score at convergence. In our setting, review ratings (often from 1 to 5) are not utilized. Thus we ignore the edge signs in FraudEagle to make these methods comparable.[6]

In addition to (1) FraudEagle [1], we also compare to (2) CatchSync [16], designed to spot synchronized behavior among users and (3) Oddball [3], for detecting users whose neighbors are in near-clique or star shapes. Oddball requires unipartite graphs, thus we use the projected review network on users, where two users with at least 5 common neighbors (products) are connected.

Performance results. In Section 2, we introduced two key observations that we use to design NFS: neighbor diversity and self-similarity. In Fig. 1, we show the entropy H_{deg} vs. KL-divergence KL_{deg} of products on the Chung-Lu1 graph as an example. We can see that the products targeted by a group of spammers reside in the bottom-right part of the figure.

[6] Accordinly, we use a single edge compatibility table (i.e., [0.9 0.1; 0.1 0.9]) for FraudEagle [1].

Table 2. AUC of Precision-Recall curve on synthetic datasets. Two values in each entry: (former) performance on high degree users (threshold at 20) and performance on all users. (HDP: number of high degree products (≥ 20); FE: FraudEagle.)

Dataset	Camouf.	HDP	Oddball[3]	CatchSync[16]	FE[1]	NFS+FE
Chung-Lu1	10% Pop.	6170	0.990/0.937	1.000/0.009	0.570/0.569	1.000/1.000
	30% Pop.	6172	0.997/0.973	1.000/0.008	0.570/0.570	1.000/1.000
	10% Rand.	6205	0.982/0.886	1.000/0.007	0.552/0.552	1.000/1.000
	30% Rand.	6266	0.881/0.386	0.957/0.007	0.532/0.526	1.000/1.000
Chung-Lu2	10% Pop.	25306	0.977/0.943	1.000/0.002	0.294/0.294	1.000/1.000
	30% Pop.	25302	0.995/0.988	1.000/0.002	0.294/0.294	1.000/1.000
	10% Rand.	25330	0.955/0.887	1.000/0.002	0.280/0.279	1.000/1.000
	30% Rand.	25392	0.711/0.374	0.982/0.002	0.261/0.256	1.000/0.977
RTG1	10% Pop.	17771	0.945/0.852	1.000/0.008	0.176/0.176	1.000/1.000
	30% Pop.	17766	0.929/0.842	0.997/0.007	0.176/0.176	1.000/1.000
	10% Rand.	17780	0.918/0.803	0.995/0.007	0.168/0.168	1.000/1.000
	30% Rand.	17843	0.637/0.367	0.878/0.007	0.163/0.158	0.952/0.950
RTG2	10% Pop.	25658	0.906/0.778	1.000/0.005	0.129/0.129	1.000/1.000
	30% Pop.	25658	0.879/0.746	1.000/0.005	0.129/0.129	1.000/1.000
	10% Rand.	25678	0.877/0.741	0.987/0.005	0.123/0.123	1.000/1.000
	30% Rand.	25716	0.577/0.331	0.778/0.005	0.119/0.115	0.952/0.951

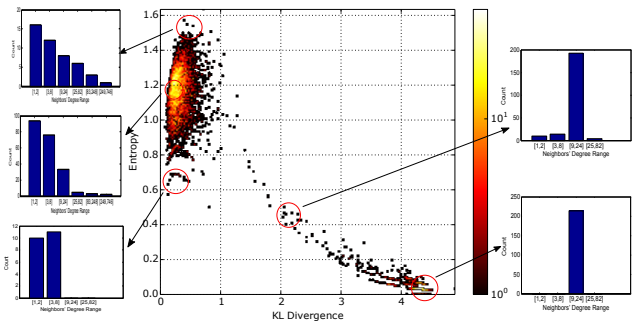

Fig. 1. Degree entropy H_{deg} vs. KL-divergence KL_{deg} of products (dots) in Chung-Lu1 (σ=10% pop. camouflage). Bottom-right depicts suspicious products with neighbors having abnormally similar degrees, whereas upper-left are clustered with normal products whose neighbors have diverse degree distributions that obeys the overall trend.

Table 2 presents the performance results (AUC of precision recall curve) of different methods. Each method is tested on high-degree users as well as all the users. Note that for all methods, users with high degrees are easier to rank as they exhibit more information for analysis. However, when all users are considered, the rankings become contaminated with low degree users bubbling up high in the ranking due to errors in scoring, and the performance of the methods drop relatively. This problem is especially evident for CatchSync. In contrast, our approach outperforms others and achieves near-perfect accuracy on all settings (ranking the spammers on top). These results demonstrate the effectiveness and robustness of NFS.

Table 3. Group discovery performance of *GroupStrainer* for varying ϵ (camouflage) and SCI (collusion) for spammers. We report NMI/hierarchy similarity threshold.

Dataset	SCI = 1.0	SCI = 0.9	SCI = 0.8	SCI = 0.7	SCI = 0.6	SCI = 0.5	SCI = 0.4
$\epsilon = 0$	1.000/0.95	1.000/0.70	1.000/0.65	1.000/0.65	0.997/0.65	1.000/0.60	0.948/0.55
$\epsilon = 0.2$	0.994/0.65	0.997/0.55	1.000/0.60	0.995/0.60	0.998/0.55	0.990/0.60	0.980/0.50
$\epsilon = 0.4$	0.993/0.50	1.000/0.55	0.993/0.55	0.998/0.55	0.994/0.55	0.988/0.55	0.980/0.50
$\epsilon = 0.6$	0.989/0.55	0.998/0.50	0.991/0.50	0.991/0.55	0.996/0.50	0.995/0.50	0.984/0.45
$\epsilon = 0.8$	0.984/0.50	0.987/0.50	0.989/0.50	0.993/0.50	0.977/0.45	0.991/0.50	0.976/0.45

4.2 Performance of *GroupStrainer* on synthetic data

Spammer group generation. As *GroupStrainer* operates on a carefully induced subgraph, we inject 20 spammer groups into a subgraph with 800 users and 200 products. Spammers are assigned to groups (of sizes between 10 to 40) at random without replacement (each spammer belongs to only one group), while the targeted products (of sizes between 2 and 12) are randomly chosen with replacement (products can be attacked several times).

From real-world datasets, we observed varying degree of collusion among spammers; in some groups they write reviews to *all* the targeted products, while in other groups they are organized into sub-groups to target different subsets of products. To the best of our knowledge, the underlying motivation is to alleviate their suspiciousness and reduce their workload at the same time. To capture this behavior, we use a *Spammer Collusion Index* (SCI) for each spammer group g defined as $SCI(g) = \sum_{g_i,g_j \subset g, i \neq j} \frac{|t(g_i) \cap t(g_j)|}{|t(g_i) \cup t(g_j)|} / \binom{n}{2}$, where g_i, g_j are subgroups in g, $t(g_i)$ denotes the set of products g_i targets, and n is the number of subgroups in g. As such, SCI is the average Jaccard similarity of subgroups' target sets. We divide groups with more than 5 targets randomly into two subgroups to simulate collusion behavior. In addition, all spammers have ϵ probability to randomly write reviews to untargeted products (i.e., camouflage).

Table 3 shows the group detection performance of *GroupStrainer* on datasets simulated with varying levels of camouflage (i.e., ϵ) and collusion (i.e., SCI) among the spammers. We report NMI $\in [0, 1]$ (Normalized Mutual Information) that measures the clustering quality of *GroupStrainer* w.r.t. ground truth, as well as the corresponding level (i.e., similarity threshold) of the hierarchy that maps to the original groups. We find that *GroupStrainer* recovers the (sub)group hierarchy among spammers effectively, with high accuracy even for large ϵ.

4.3 Results on real-world data

Having validated the effectiveness of NFS and *GroupStrainer* through synthetic results, we next employ our method on two real-world review datasets; `iTunes` app-store reviews, and `Amazon` product reviews.

Fig. 2 illustrates the scatter plot of H_{deg} vs. KL_{deg} for products in `iTunes`, where outliers with large NFS values are circled (H_{pr} vs. KL_{pr} looks similar).

Table 4. Summary statistics of detected groups in `iTunes` and `Amazon`. #P (#U): number of products (users), t: time stamps, ⋆: ratings, s: scattered (distribution), c: concentrated (distribution), *Dup*: number of duplicate reviews/total count.

	`iTunes`					`Amazon`				
ID	#P	#U	t, ⋆	*Dup*	Developer	#P	#U	t, ⋆	*Dup*	Category, Author
1	5	31	s, c	51/154	all same	10	20	c, c	90/138	Books, all same
2	8	38	c, s	29/202	2 same	4	12	s, c	32/47	Books, all same
3	4	61	s, c	34/144	all inaccessible	7	9	c, c	44/60	Books, all same
4	4	17	c, s	0/68	1 inaccessible	7	19	s, c	5/88	Books, all same
5	5	102	c, s	8/326	different	23	42	c, c	2/468	Music, all same
6	6	50	s, c	4/173	all same	8	17	s, c	9/73	Books, 4/8 same
7	2	56	c, c	12/112	different	6	18	s, c	4/94	Movies&TV, all same
8	4	42	c, c	8/112	2 same					
9	6	67	s, c	0/137	all same					

The adjacency matrix A of the 2-hop induced subnetwork on the outlier products is shown in Fig. 3 (top). While the groups are not directly evident here, the *GroupStrainer* output clearly reveals various colluding user groups as shown in Fig. 3 (bottom). Statistics and properties of the groups are listed in Table 4.

We note that group#1 is the same 31 users spamming 5 products of the same developer with all-5⋆ reviews, as was previously found in [1]. Our method finds other suspicious user groups; e.g., group#2 consists of 8 products, each receiving all their reviews on the same day from a subset of 38 users. Interestingly, while the time-stamp is the same (concentrated), the ratings are a bit diversified (not all 5⋆s but 3&4⋆s as well)—potentially for camouflage. This behavior is observed among other groups; while some groups concentrate on both time and ratings (c,c), e.g., all 5⋆ in one day, most groups aim to diversify in one of the aspect. We also note the duplicate reviews across reviewers in the same group.

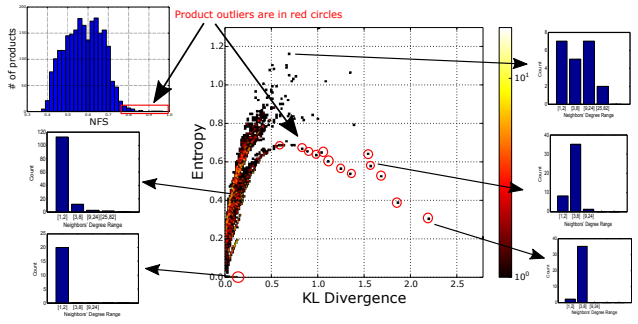

Fig. 2. Degree entropy H_{deg} vs. KL-divergence KL_{deg} of products (dots) in `iTunes`. Top-left: distribution of NFS scores; circled points: outlier products by NFS.

Similar results are obtained on `Amazon`, as shown in Fig. 4. We also provide the summary statistics and characteristics of the 7 colluding groups in Table 4.

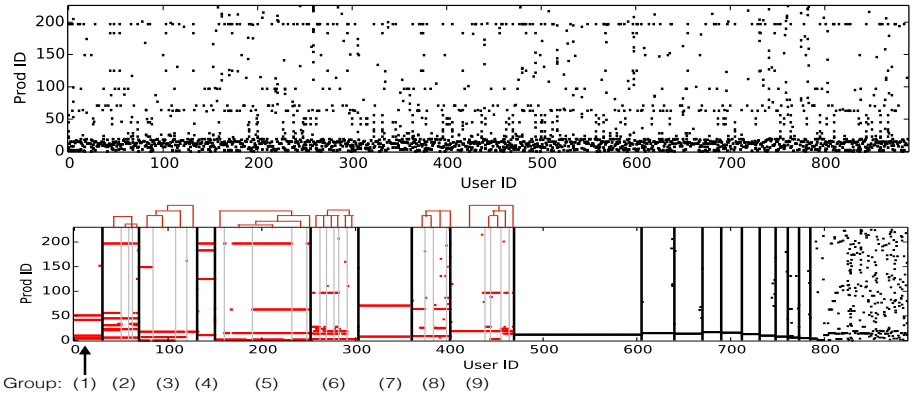

Fig. 3. (top) 2-hop induced network of top-ranking products by NFS in `iTunes`, (bottom) output of *GroupStrainer* with 9 discovered colluding user groups.

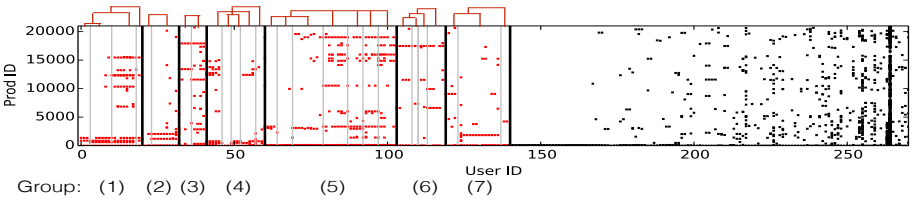

Fig. 4. Output of *GroupStrainer* on `Amazon`, with 7 discovered colluding user groups.

We find that the majority of the targeted products in our groups belong to the Books category. This is not a freak occurrence, as the media has reported that the authors of books get involved in opinion spam to gain popularity (see URL in footnote[1] on pg. 1). For example, group#1 consists of 20 users spamming 10 books. 19/20 users write their reviews on the exact same day. 15/20 has duplicate reviews across products, in total 90/138 reviews have at least one copy. Our dataset listed the same author for 9/10 books. Manual inspection of the authors revealed that the 10th book also belonged to the same author but was mis-indexed by Amazon.

5 Related Work

Opinion spam is one of the new forms of Web-based spam, emerging as a result of review-based sites (TripAdvisor, Yelp, etc.) gaining popularity. We organize related work into two: (*i*) spotting *individual* spam reviewers or fake reviews, and (*ii*) detecting *group* spammers.

Detecting individual spam reviews and reviewers. To identify individual fake *reviews*, supervised models have been trained based on text and behavioral features [17] or linguistic patterns [12, 25]. Another approach [20] employs the semi-supervised co-training method by using review text and reviewer features as two

separate views. Relational models have also been explored [21], which correlate reviews written by the same users and from the same IPs. Similarly, behavioral methods have been developed to identify individual spam *reviewers* [13, 22]. Other approaches use association based rule mining of rating patterns [18], or temporal analysis [11]. There also exist network-based methods that spot *both* suspicious reviewers and reviews [1, 26]. Those infer a suspiciousness score for each node/edge in the user-product or user-review-product network.

Detecting group spam. There exist only a few efforts that focus on group-level opinion spam detection [23, 27, 28]. This is counter-intuitive, since spam/fraud is often an organized act in which the involved individuals cooperate to reduce effort and response time, increase total impact, and camouflage so that no single individual stands out. All related work in this category define group-level or pairwise spam indicators to score reviewer groups by suspiciousness. Their indicators are several behavioral and linguistic features. In contrast, our work solely uses network footprints without relying on side information (e.g., language, time-stamps, etc.) that can be fine-tuned by spammers.

6 Conclusion

We proposed an unsupervised and scalable approach for spotting spammer groups in online review sites solely based on their network footprints. Our method consists of two main components: (1) NFS; a new measure that quantifies the statistical distortions of well-studied properties in the review network, and (2) *GroupStrainer*; a hierarchical clustering method that chips off colluding groups from a subnetwork induced on target products with high NFS values. We validated the effectiveness of our method on both synthetic and real-world datasets, where we detected various groups of users with suspicious colluding behavior.

Acknowledgments. The authors thank the anonymous reviewers for their useful comments. This material is based upon work supported by the ARO Young Investigator Program under Contract No. W911NF-14-1-0029, NSF CAREER 1452425, IIS 1408287 and IIP1069147, a Facebook Faculty Gift, an R&D grant from Northrop Grumman Aerospace Systems, and Stony Brook University Office of Vice President for Research. Any conclusions expressed in this material are of the authors' and do not necessarily reflect the views, either expressed or implied, of the funding parties.

References

1. L. Akoglu, R. Chandy, and C. Faloutsos. Opinion fraud detection in online reviews by network effects. In *ICWSM*, 2013.
2. L. Akoglu and C. Faloutsos. RTG: A recursive realistic graph generator using random typing. In *ECML/PKDD*, pages 13–28, 2009.

3. L. Akoglu, M. McGlohon, and C. Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *PAKDD*, pages 410–421, 2010.
4. A.-L. Barabási, R. Albert, and H. Jeong. Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A: Statistical Mechanics and its Applications*, 281(1-4):69–77, June 2000.
5. A. A. Benczr, K. Csalogny, T. Sarls, and M. Uher. Spamrank – fully automatic link spam detection. In *AIRWeb*, pages 25–38, 2005.
6. A. Broder. Graph structure in the web. *Computer Networks*, 33(1-6):309–320, 2000.
7. F. R. K. Chung and L. Lu. The average distance in a random graph with given expected degrees. *Internet Mathematics*, 1(1):91–113, 2003.
8. N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial classification. In *KDD*, pages 99–108, 2004.
9. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.
10. K. Faust. Centrality in affiliation networks. *Social Networks*, 19(2):157–191, 1997.
11. G. Fei, A. Mukherjee, B. Liu, M. Hsu, M. Castellanos, and R. Ghosh. Exploiting burstiness in reviews for review spammer detection. In *ICWSM*, 2013.
12. S. Feng, R. Banerjee, and Y. Choi. Syntactic stylometry for deception detection. In *ACL*, 2012.
13. S. Feng, L. Xing, A. Gogar, and Y. Choi. Distributional footprints of deceptive product reviews. In *ICWSM*, 2012.
14. J. Gao and P.-N. Tan. Converting output scores from outlier detection algorithms to probability estimates. In *ICDM*, 2006.
15. A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.
16. M. Jiang, P. Cui, A. Beutel, C. Faloutsos, and S. Yang. Catchsync: catching synchronized behavior in large directed graphs. In *KDD*, pages 941–950, 2014.
17. N. Jindal and B. Liu. Opinion spam and analysis. In *WSDM*, pages 219–230, 2008.
18. N. Jindal, B. Liu, and E.-P. Lim. Finding unusual review patterns using unexpected rules. In *CIKM*, pages 1549–1552. ACM, 2010.
19. W. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. volume 26 of *Contemporary Mathematics*, pages 189–206. 1984.
20. F. Li, M. Huang, Y. Yang, and X. Zhu. Learning to identify review spam. In *IJCAI*, 2011.
21. H. Li, Z. Chen, B. Liu, X. Wei, and J. Shao. Spotting fake reviews via collective positive-unlabeled learning. In *ICDM*, 2014.
22. A. Mukherjee, A. Kumar, B. Liu, J. Wang, M. Hsu, M. Castellanos, and R. Ghosh. Spotting opinion spammers using behavioral footprints. In *KDD*, 2013.
23. A. Mukherjee, B. Liu, and N. S. Glance. Spotting fake reviewer groups in consumer reviews. In *WWW*, 2012.
24. M. Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, 46(5):323–351, 2005.
25. M. Ott, Y. Choi, C. Cardie, and J. T. Hancock. Finding deceptive opinion spam by any stretch of the imagination. In *ACL*, pages 309–319, 2011.
26. G. Wang, S. Xie, B. Liu, and P. S. Yu. Review graph based online store review spammer detection. In *ICDM*, pages 1242–1247, 2011.
27. C. Xu and J. Zhang. Combating product review spam campaigns via multiple heterogeneous pairwise features. In *SDM*. SIAM, 2015.
28. C. Xu, J. Zhang, K. Chang, and C. Long. Uncovering collusive spammers in Chinese review websites. In *CIKM*, pages 979–988. ACM, 2013.