Discovering Opinion Spammer Groups by Network Footprints

Junting Ye

Leman Akoglu

Stony Brook University
Computer Science

ECML/PKDD 2015

Introduction

- Product reviews are one major source of information.

Advertisement

Reviews

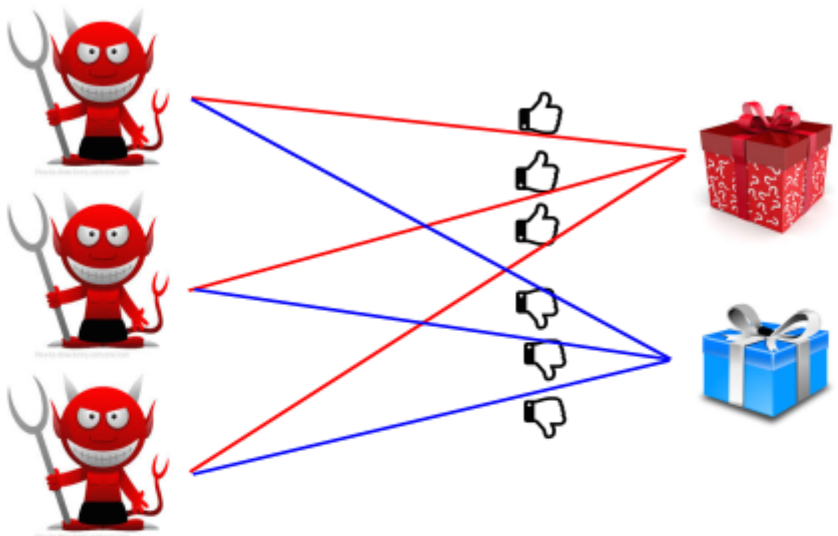
- Product reviews are important to businesses!

+1 star-rating increases revenue by 5-9%

Harvard Study by M. Luca
Reviews, Reputation, and Revenue: The Case of Yelp.com

Opinion Spam

- Opinion Spammers are hired to write fake reviews;



- Opinion spam is everywhere!
    - 14~20% in Yelp; [Mukherjee et al., ICWSM 2013]
    - 2~6% in Orbitz, Priceline, Expedia, Tripadvisor, etc. [Ott et al., WWW 2012]
- Challenges in detecting spammers:
    - Spammers camouflage, linguistic or behavioral methods might fail;
    - Lack of ground truth, difficulty in manual labeling; [Ott et al. ACL 2011]

Motivation

- Spamming in **groups** is common because:
  - Impact maximized: dominate the sentiments
  - Effort can be shared: workload split among members
  - Easier to hide: suspicious acts are balanced so no one stands out

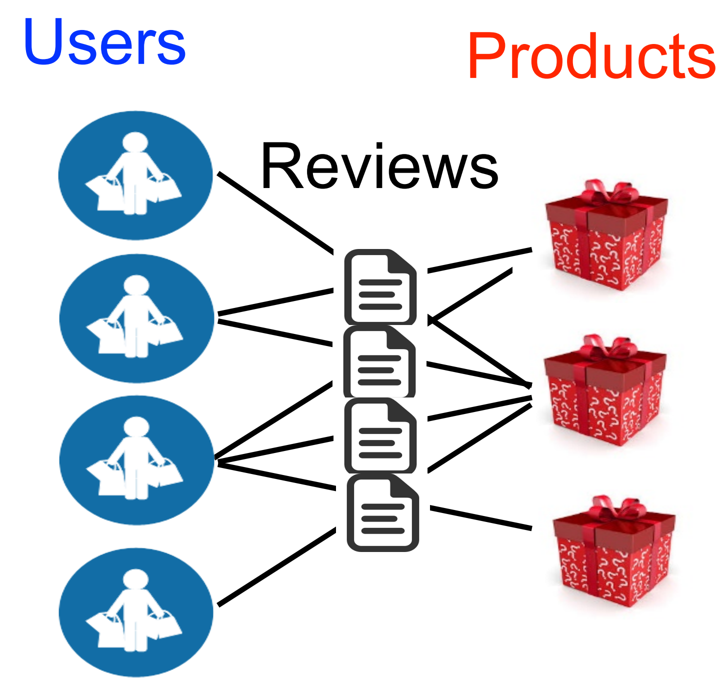
- Advantage of detecting with network footprints:
  - More **cost** for spammers to mimic local network features
  - Spammers **unaware** of the global network features

Problem Definition

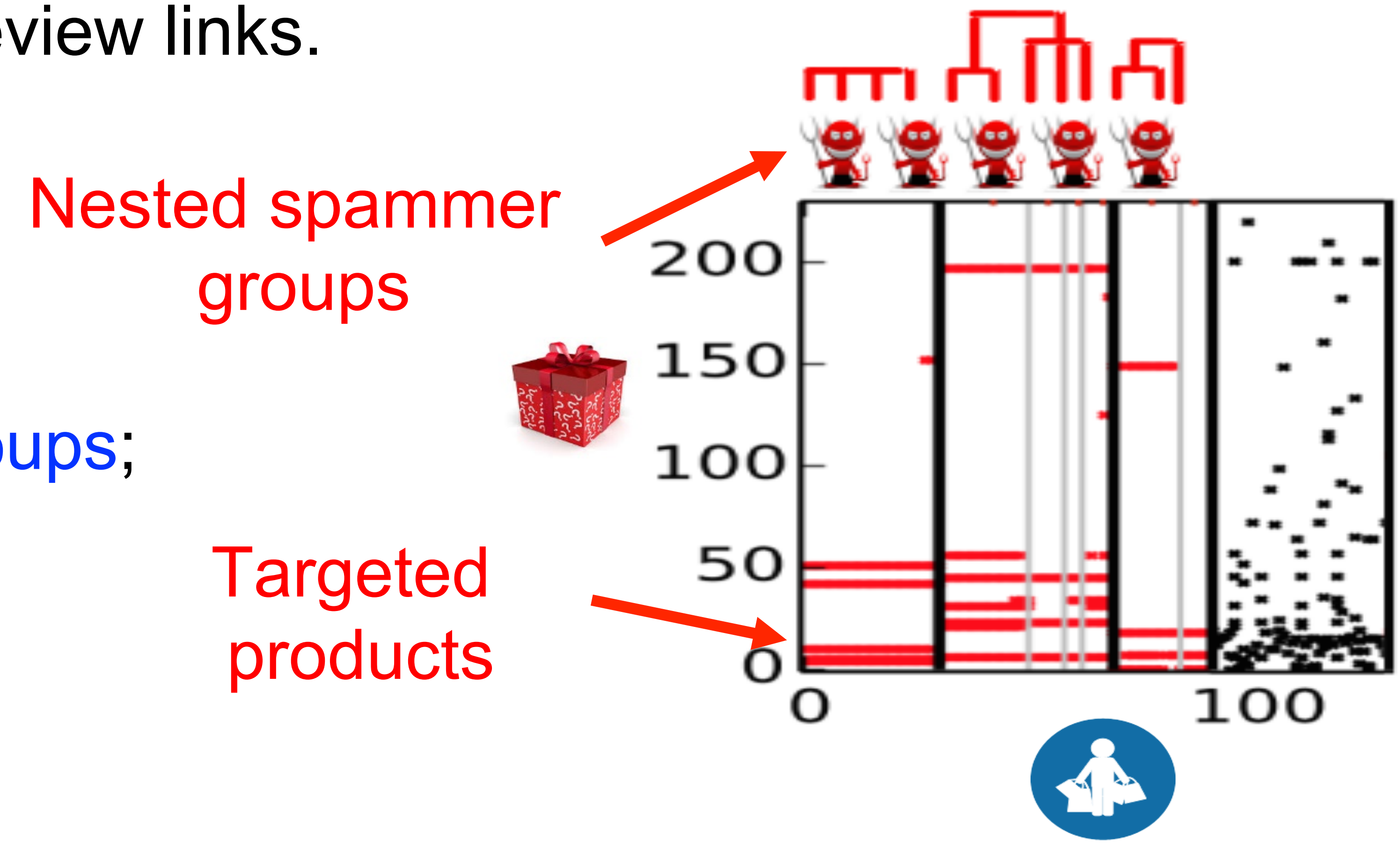
- Input:
  - A user-product bipartite review network

$$G = (U, P, E)$$

U: the set of users, P: set of products, E: set of review links.



- Output:
  - Nested spammer groups;
  - Targeted products;

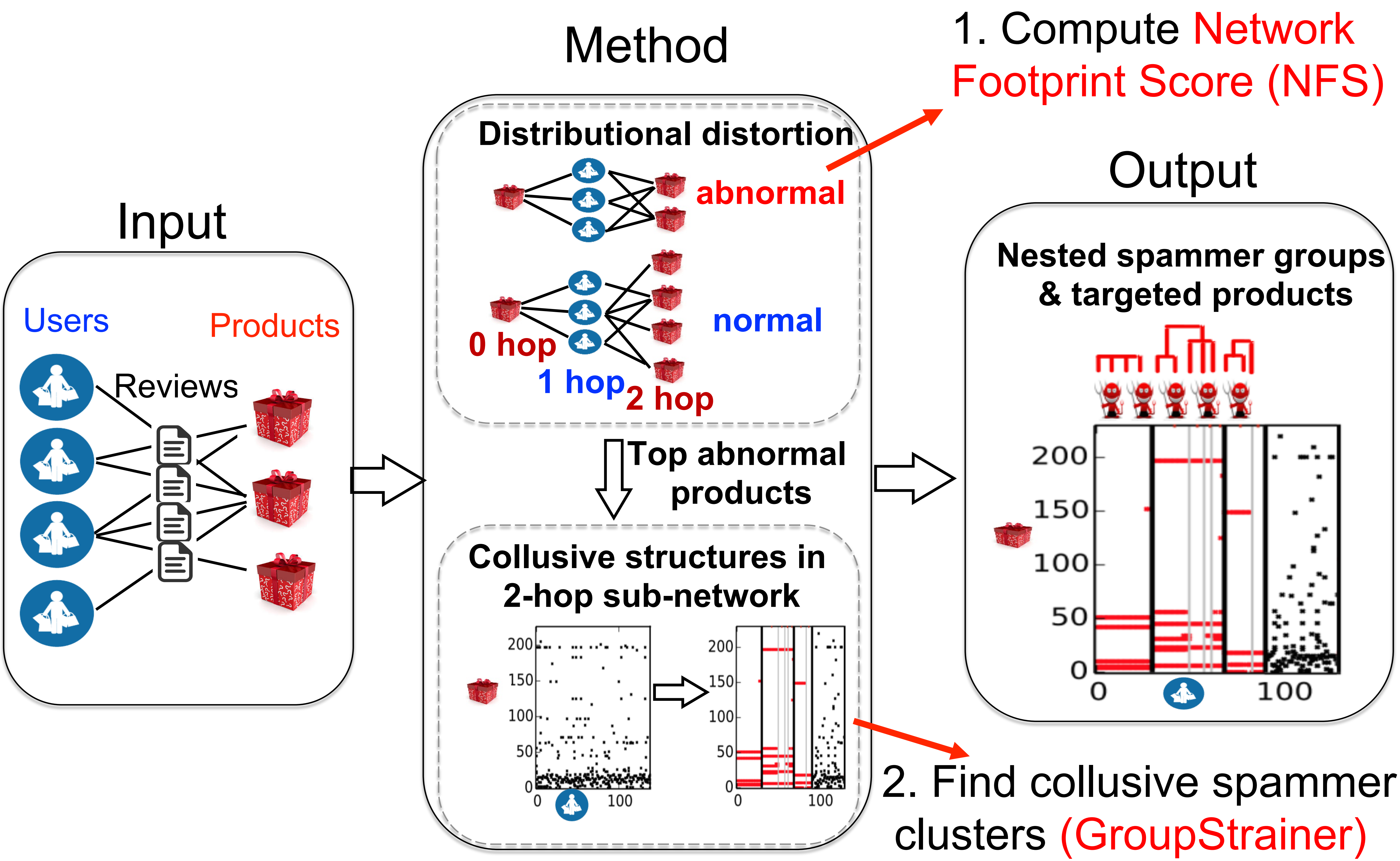


Previous Work

- Majority: Detecting **individual** spam(mer)s:
  - Supervised methods [Feng+ ACL 2012; Jindal&Liu WSDM 2008]
  - Semi-supervised methods [Li et al., IJCAI 2011]
  - Graph-based methods [Akoglu+ ICWSM 2013; Wang+ ICDM 2011]
  - Collective classification methods; [Li et al., ICDM 2014]
- Detecting **group** spam(mer)s:
  - Linguistic, rating and temporal data to compute user suspiciousness [Xu&Zhang SDM 2015; Xu+ CIKM 2013; Mukherjee+ WWW 2012]
  - Our work only utilizes the review network

Overview: 2 main steps

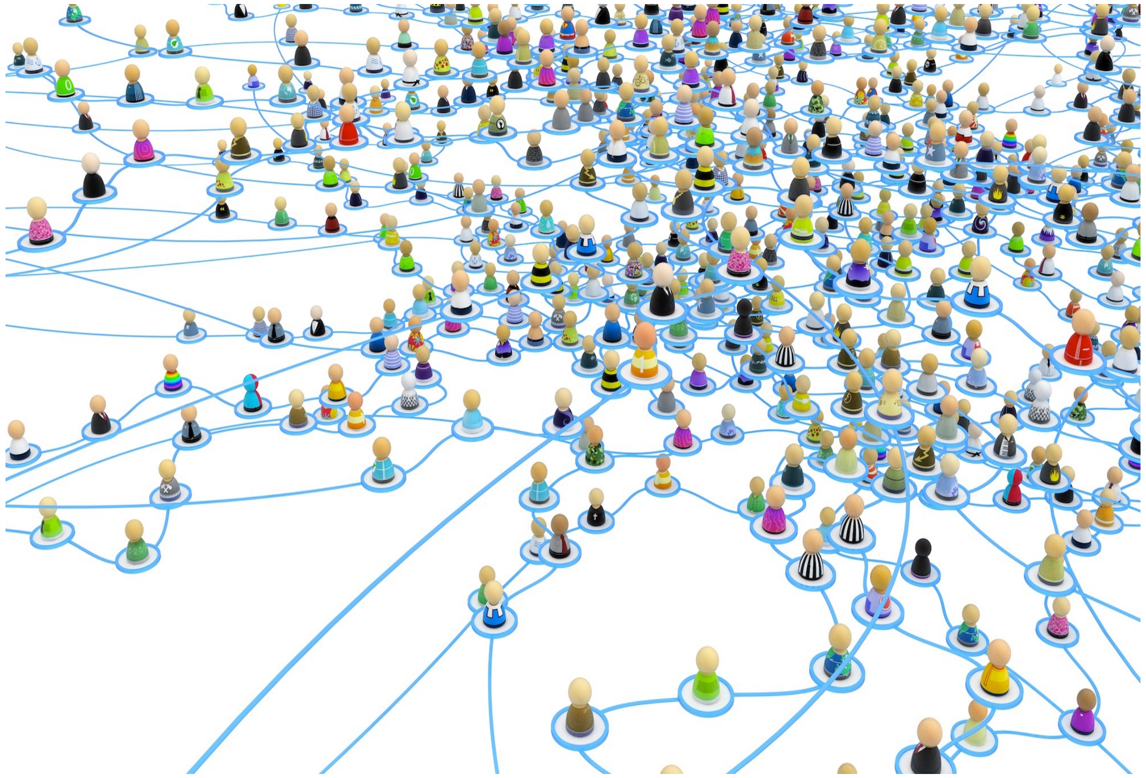
Stony Brook University
Computer Science



1. Network Footprint Score (NFS)

Stony Brook University Computer Science

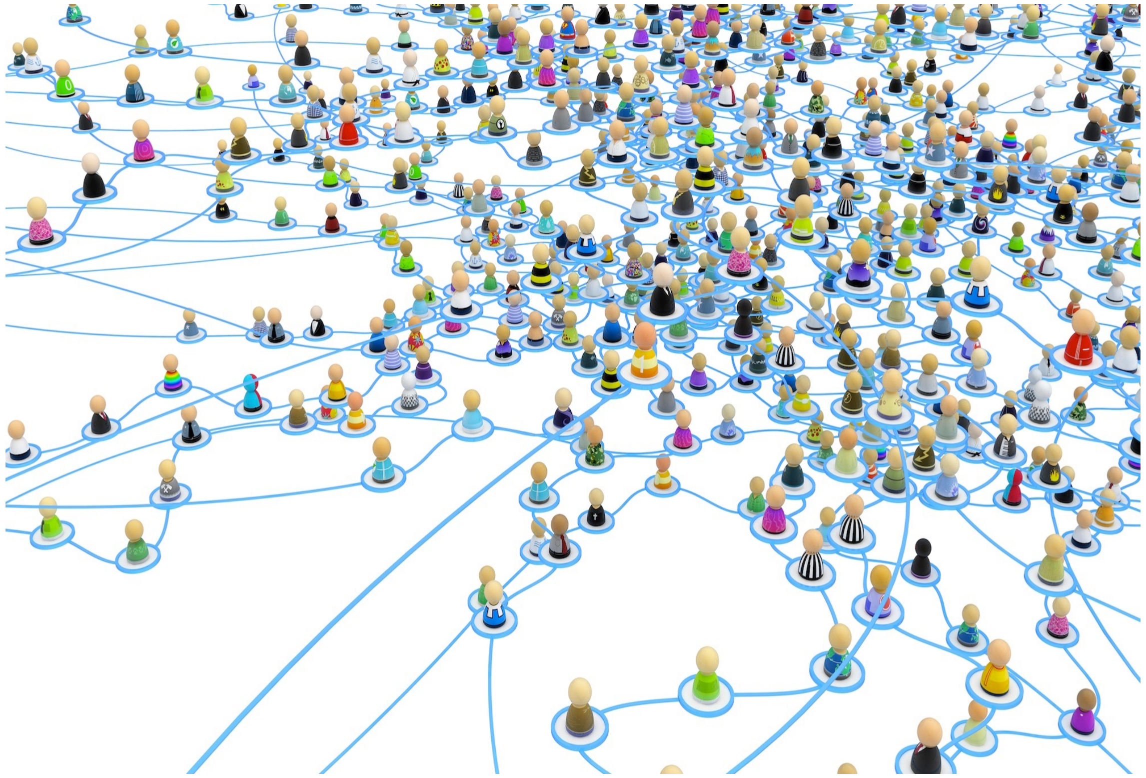
- **Observation 1:** Neighbor diversity
  - Varying levels of activities (i.e. centralities of nodes)
  - This measures the local network features



1. Network Footprint Score (NFS)

Stony Brook University Computer Science

- **Observation 1:**
  Neighbor diversity
  - Varying levels of activities (i.e. centralities of nodes)
  - This measures the local network features



- Quantification:
  - Shannon Entropy (H) of neighbors' centrality;

p: centrality density histogram

i: index of products

$$H_c(i) = -\sum_{k=1}^{K} p_k^{(i)} \log p_k^{(i)}$$

c: type of centrality (Degree or Pagerank)

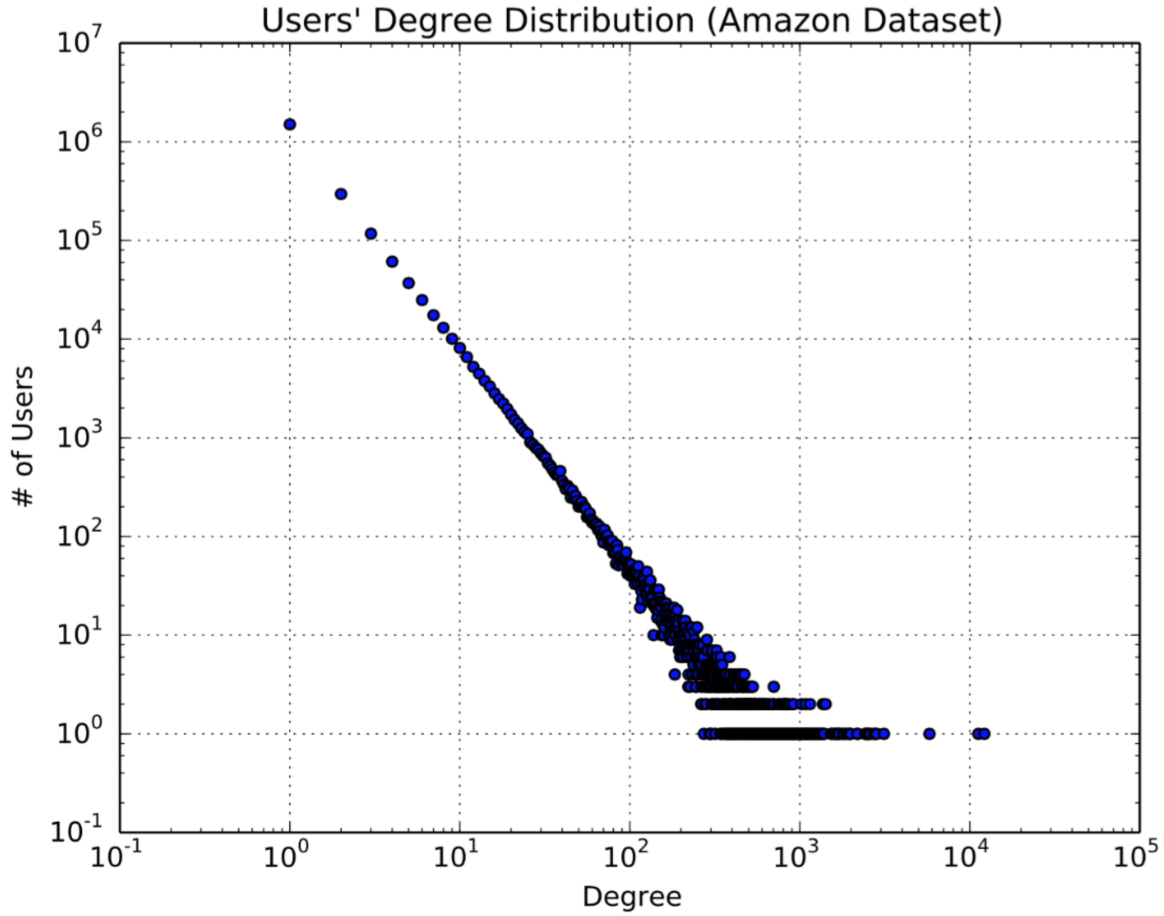
k: index of bins

1. Network Footprint Score (NFS)

Stony Brook University
Computer Science

- **Observation 2:** Self-similarity
  - Graph portions should have similar distribution as the whole graph

    → Product's neighbors should follow power-law-like distribution as the global distribution of all users;

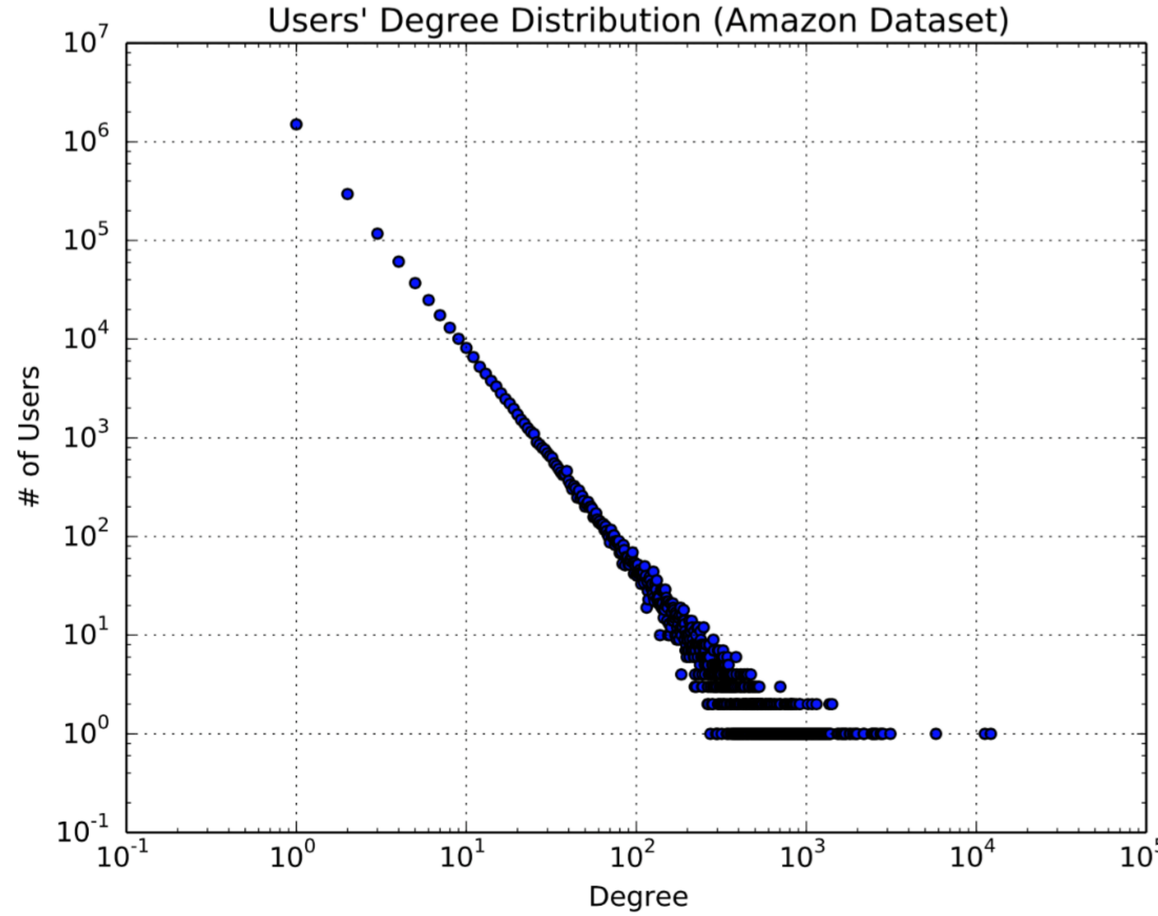


1. Network Footprint Score (NFS)

Stony Brook University
Computer Science

- **Observation 2:** Self-similarity
  - Graph portions should have similar distribution as the whole graph

    → Product's neighbors should follow power-law-like distribution as the global distribution of all users;



- Quantification:
  - KL-Divergence (*KL*) between neighbors and all users'

P^(i)^: centrality histogram of product *i*'s neighbors

Q: centrality histogram of all users

$$KL_c(P^{(i)}\|Q) = \sum_k p_k^{(i)} \log \frac{p_k^{(i)}}{q_k}$$

c: type of centrality (Degree or Pagerank)

k: index of bins

1. Network Footprint Score (NFS)

- NFS: integrating 4 observations

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

i: index of products

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

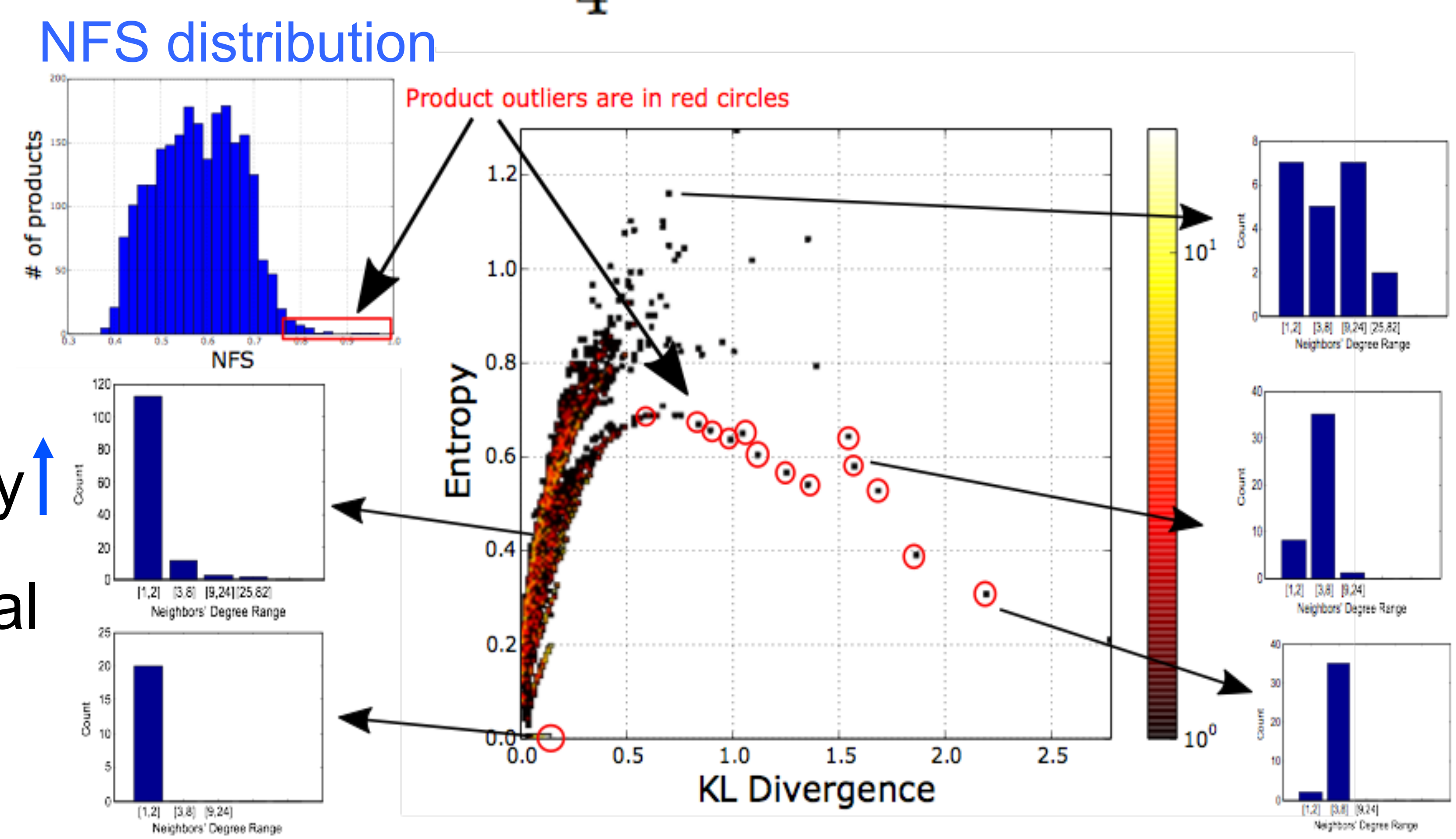
$$NFS(i) = 1 - \sqrt{\frac{f(H_{deg}(i))^2 + f(H_{pr}(i))^2 + f(KL_{deg}(i))^2 + f(KL_{pr}(i))^2}{4}}$$

- Interpretation:

  Entropy ↓ Abnormality ↑

  KL Divergence ↑ Abnormality ↑

  Right-bottom: more abnormal



Degree entropy vs. KL-divergence in iTunes

2. GroupStrainer

- Induce local **sub-network**:
  1. k products with highest NFS, **k** chosen by mixture modeling [Gao et al. ICDM 2006]
  2. Induce a 2-hop sub-network: **k** abnormal products as seeds

Distribution of NFS

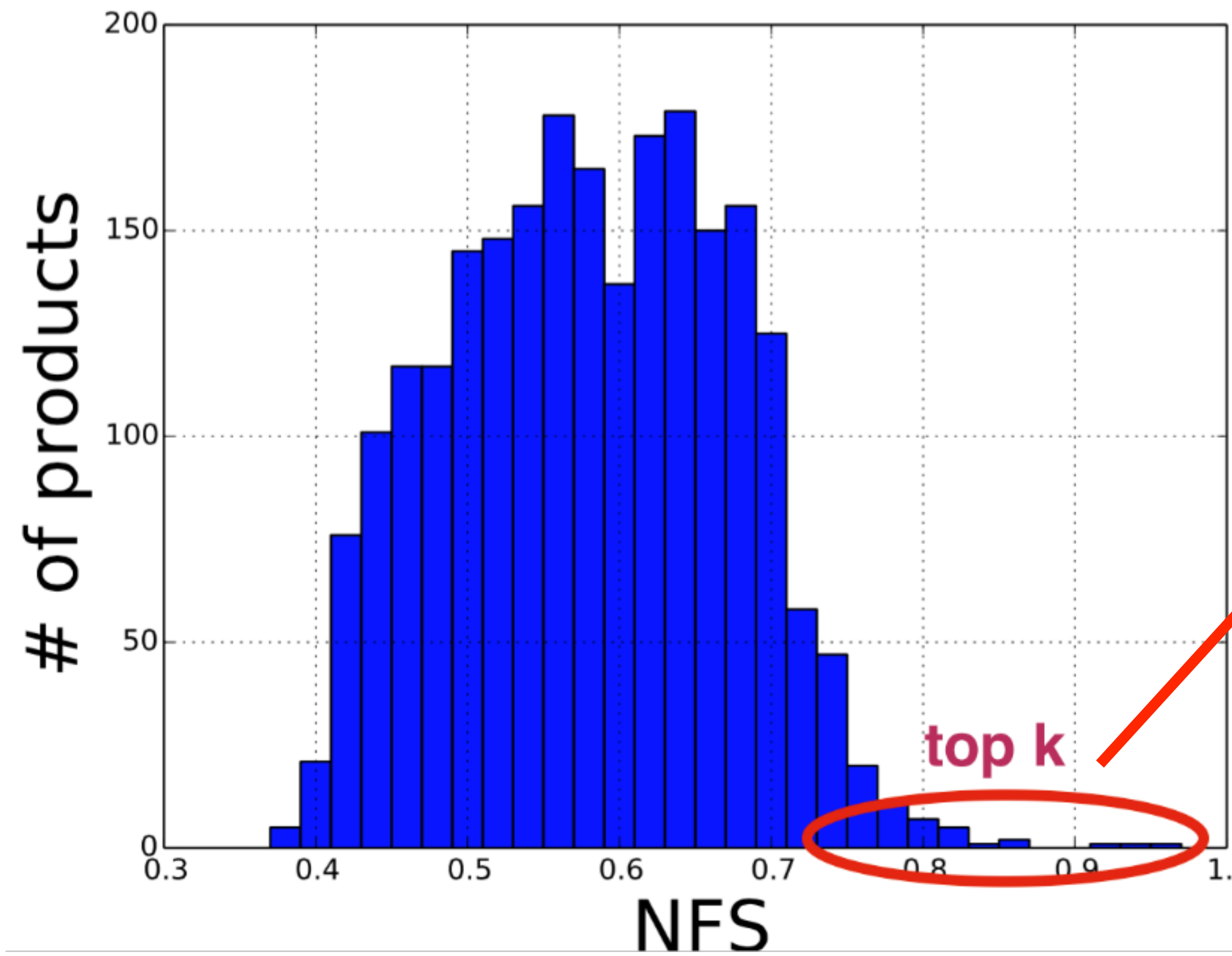
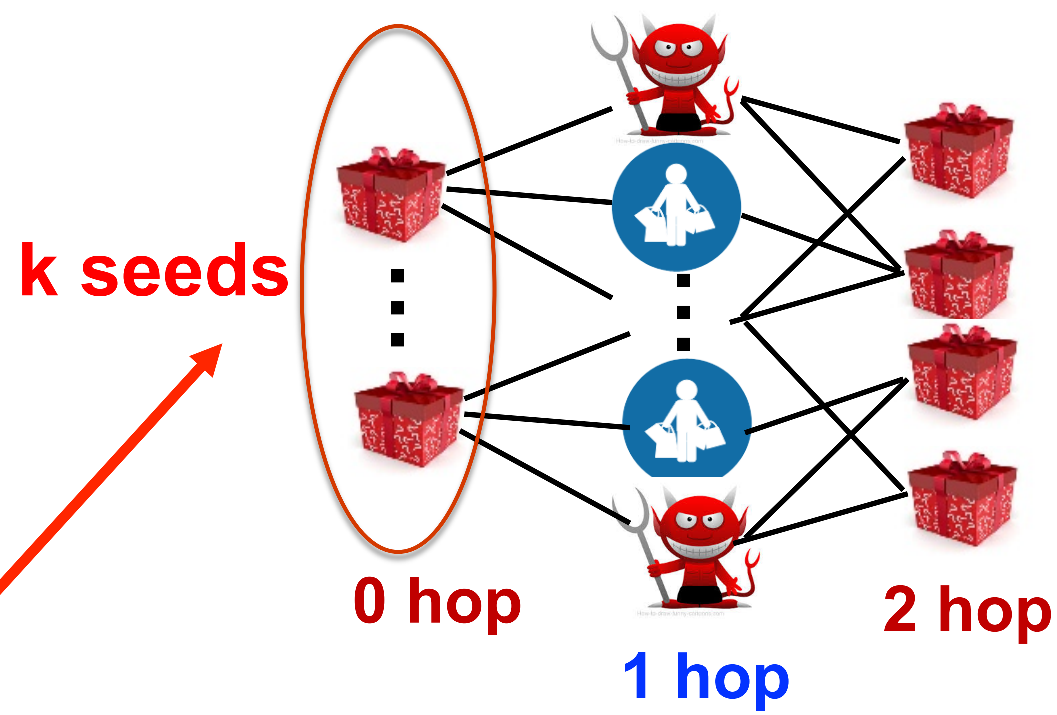


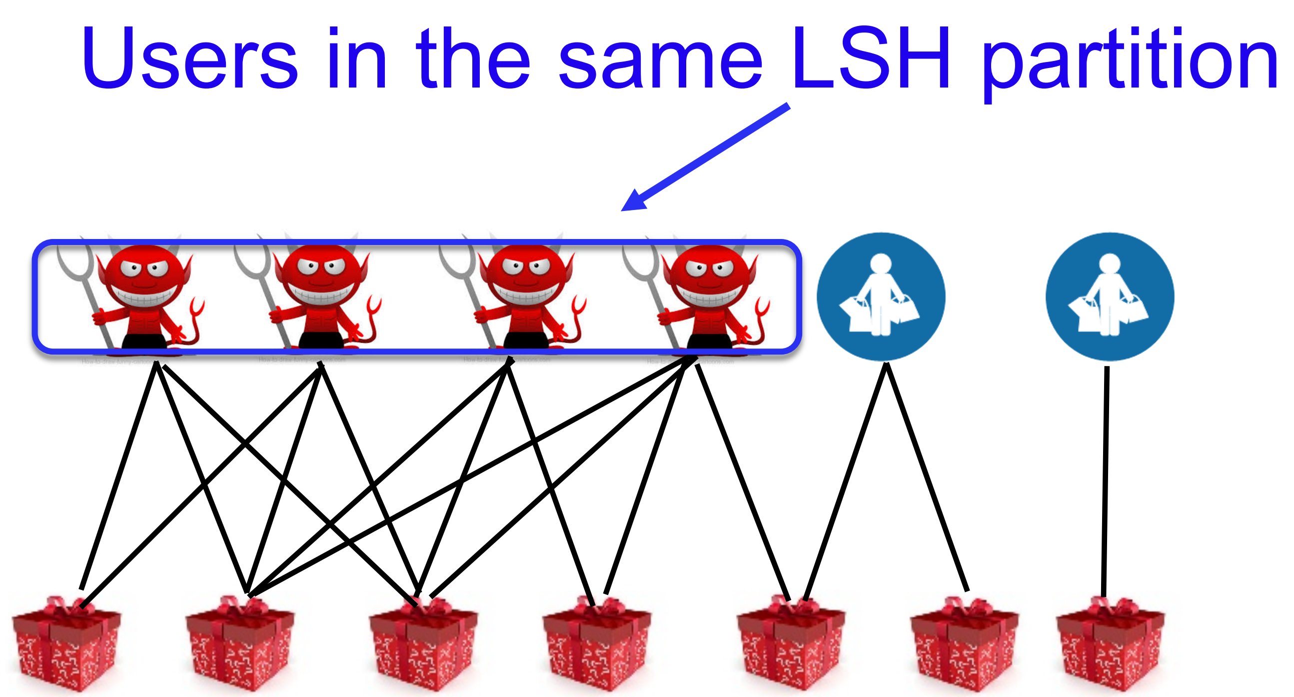
Fig4: 2-hop induced sub-network



2. GroupStrainer

- Efficient clustering
  1. Init similarity thresholds

     $S = \{s_1, s_2, \ldots, s_n\}$
  2. For each iteration i, use Locality Sensitive Hashing (LSH) to partition users
  3. In each partition, merge user groups if all pair-wise similarities are larger than s_i
  4. Terminate if no new merges, otherwise go to step 2

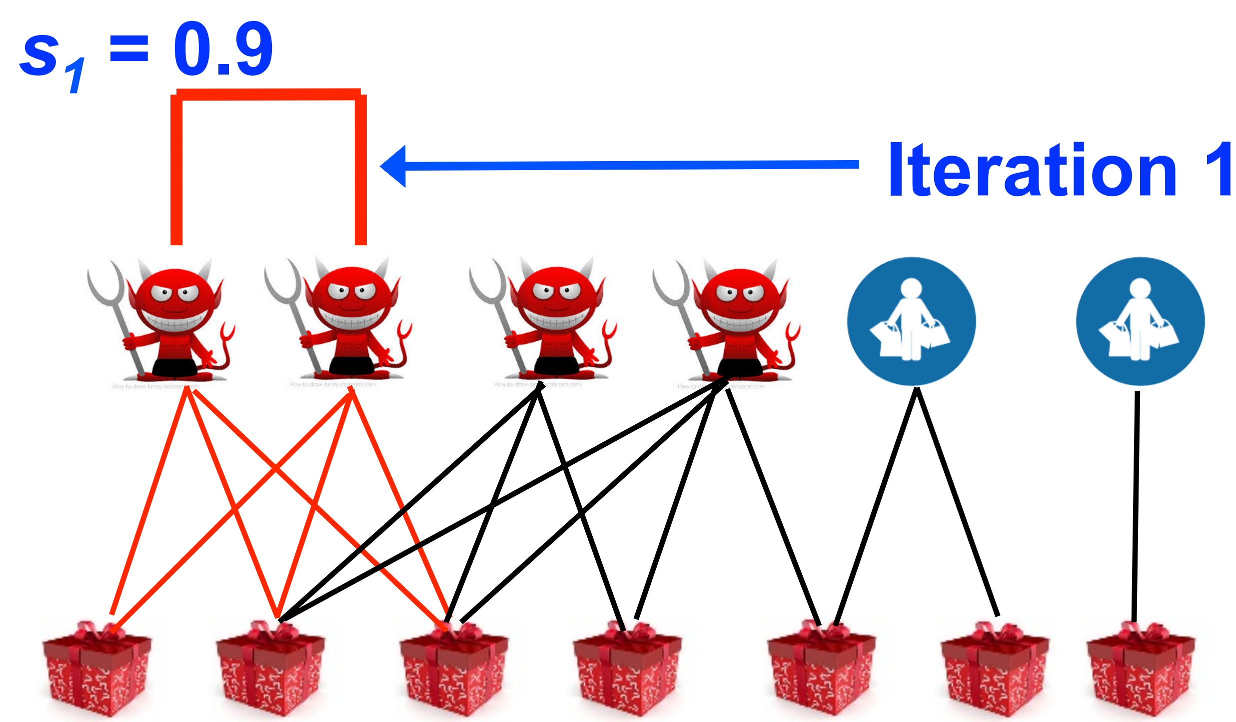


Example illustrating GroupStrainer

2. GroupStrainer

- Efficient clustering
  1. Init similarity thresholds

     $S = \{s_1, s_2, \ldots, s_n\}$
  2. For each iteration i, use Locality Sensitive Hashing (LSH) to partition users
  3. In each partition, merge user groups if all pair-wise similarities are larger than s_i
  4. Terminate if no new merges, otherwise go to step 2

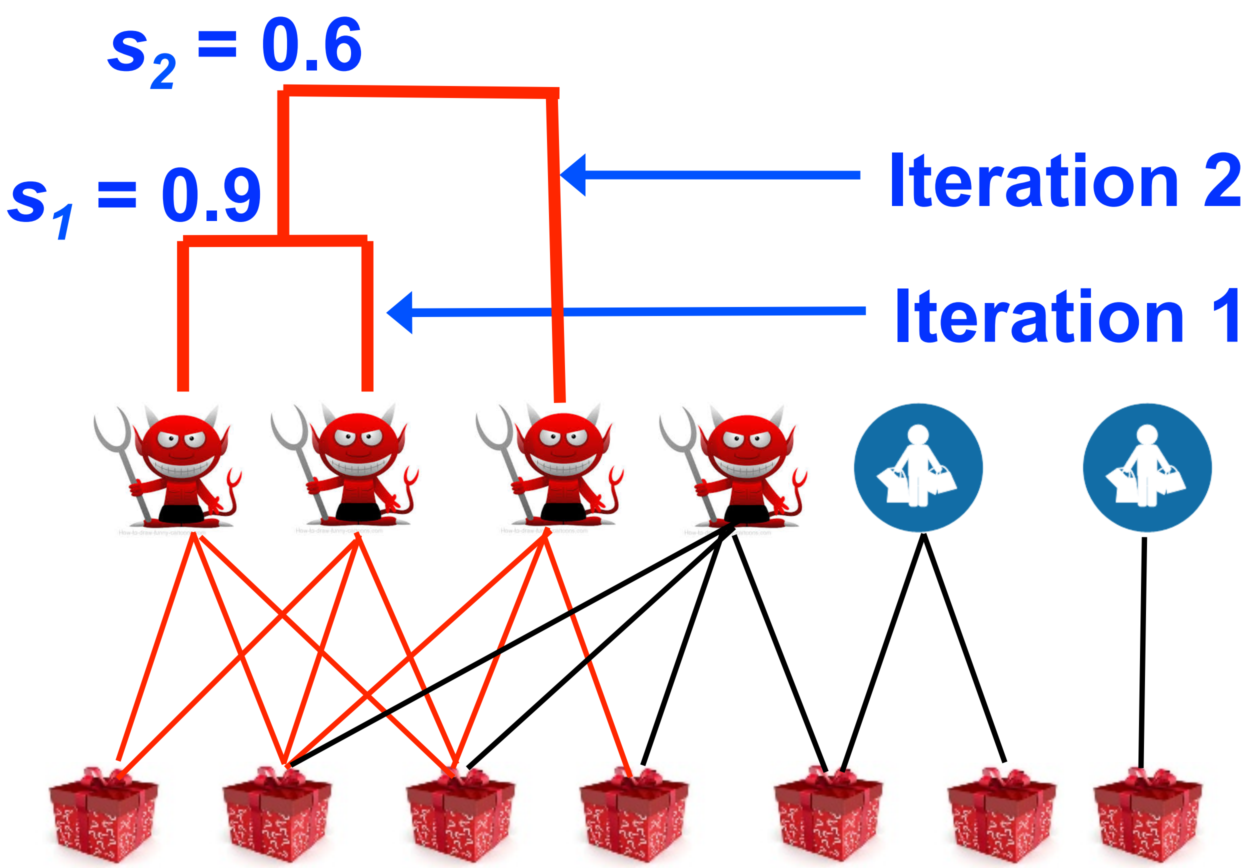


Example illustrating GroupStrainer

2. GroupStrainer

- Efficient clustering
  1. Init similarity thresholds

     $S = \{s_1, s_2, \ldots, s_n\}$
  2. For each iteration i, use Locality Sensitive Hashing (LSH) to partition users
  3. In each partition, merge user groups if all pair-wise similarities are larger than s_i
  4. Terminate if no new merges, otherwise go to step 2

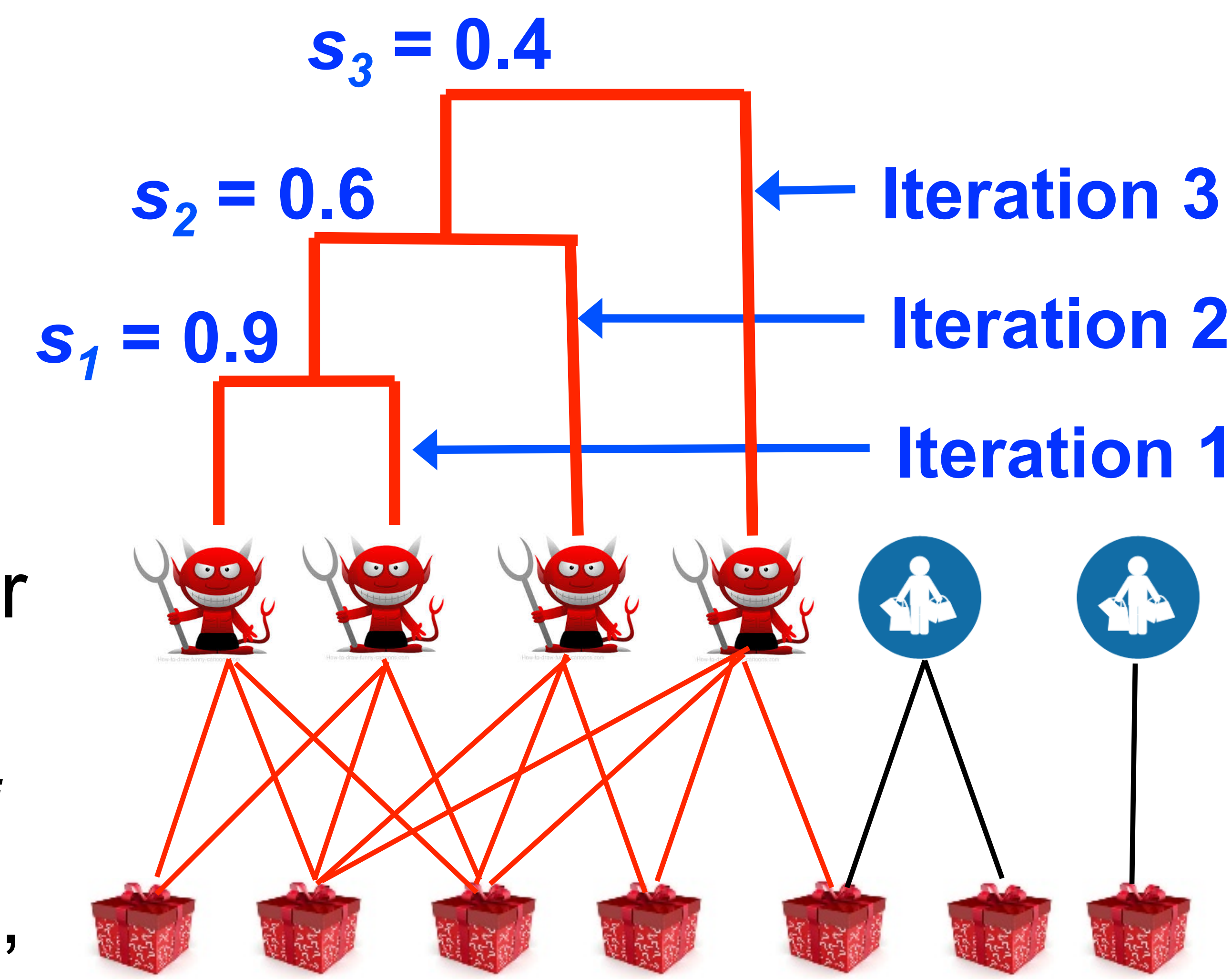


Example illustrating GroupStrainer

2. GroupStrainer

- Efficient clustering
  1. Init similarity thresholds

     $S = \{s_1, s_2, \ldots, s_n\}$
  2. For each iteration i, use Locality Sensitive Hashing (LSH) to partition users
  3. In each partition, merge user groups if all pair-wise similarities are larger than s_i
  4. Terminate if no new merges, otherwise go to step 2



Example illustrating GroupStrainer

Datasets

- **Synthetic** datasets: (4 datasets, various generators and sizes)
  - Chung-Lu Generator [Chung et al., Internet Mathematics, 2003]
  - Random Typing Generator (RTG) [Akoglu et al., PKDD, 2009]
- **Real-world** datasets:
  - iTunes [Akoglu et al., ICWSM 2013]
  - Amazon [Jindal and Liu, WSDM 2008]

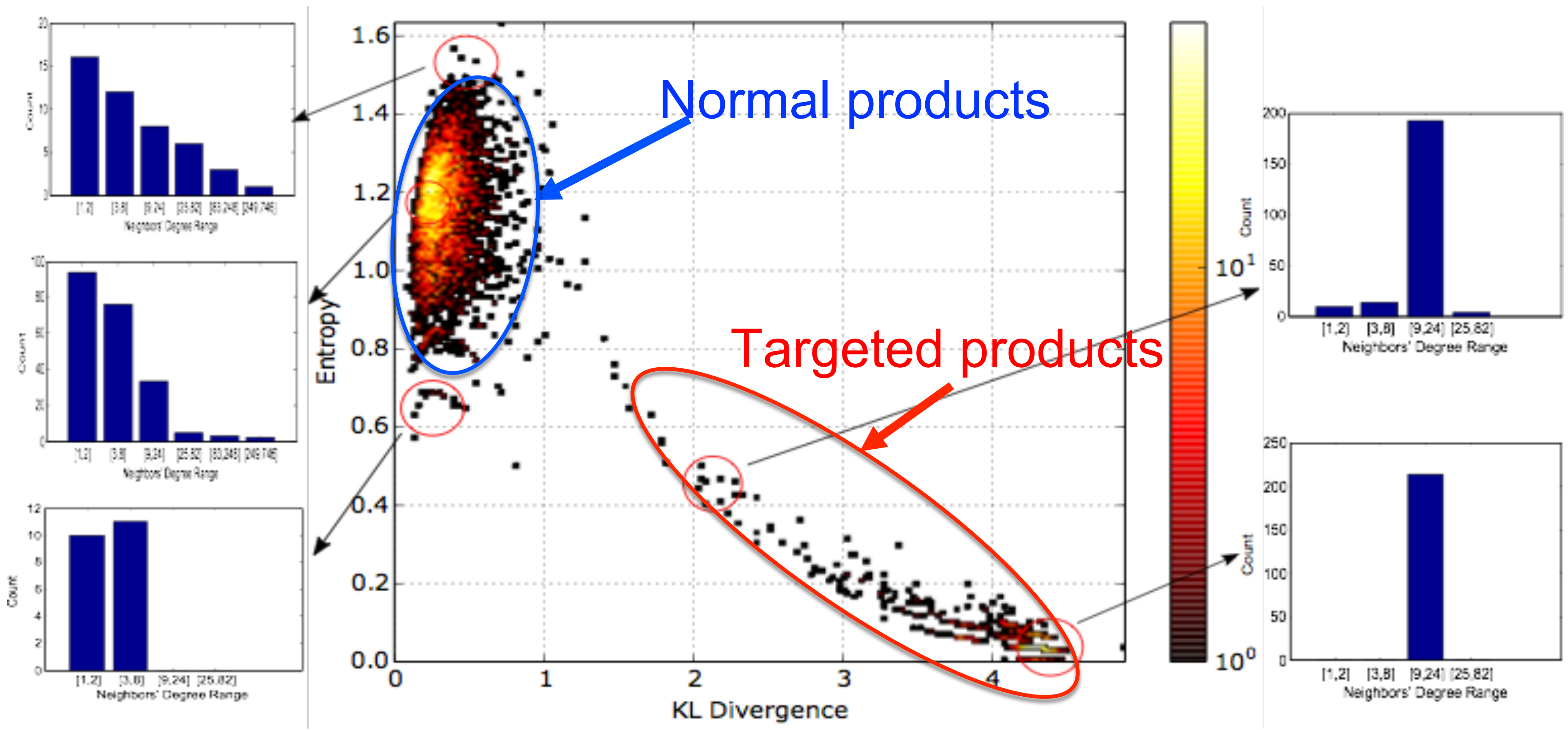
Table 1. Summary of synthetic and real-world datasets used in this work.

	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

NFS on Synthetic Graphs

Stony Brook University
Computer Science

- Different region, different shape of centrality histograms



Degree Entropy vs. KL Divergence in Chung-Lu1
(10% pop. Camouf.)

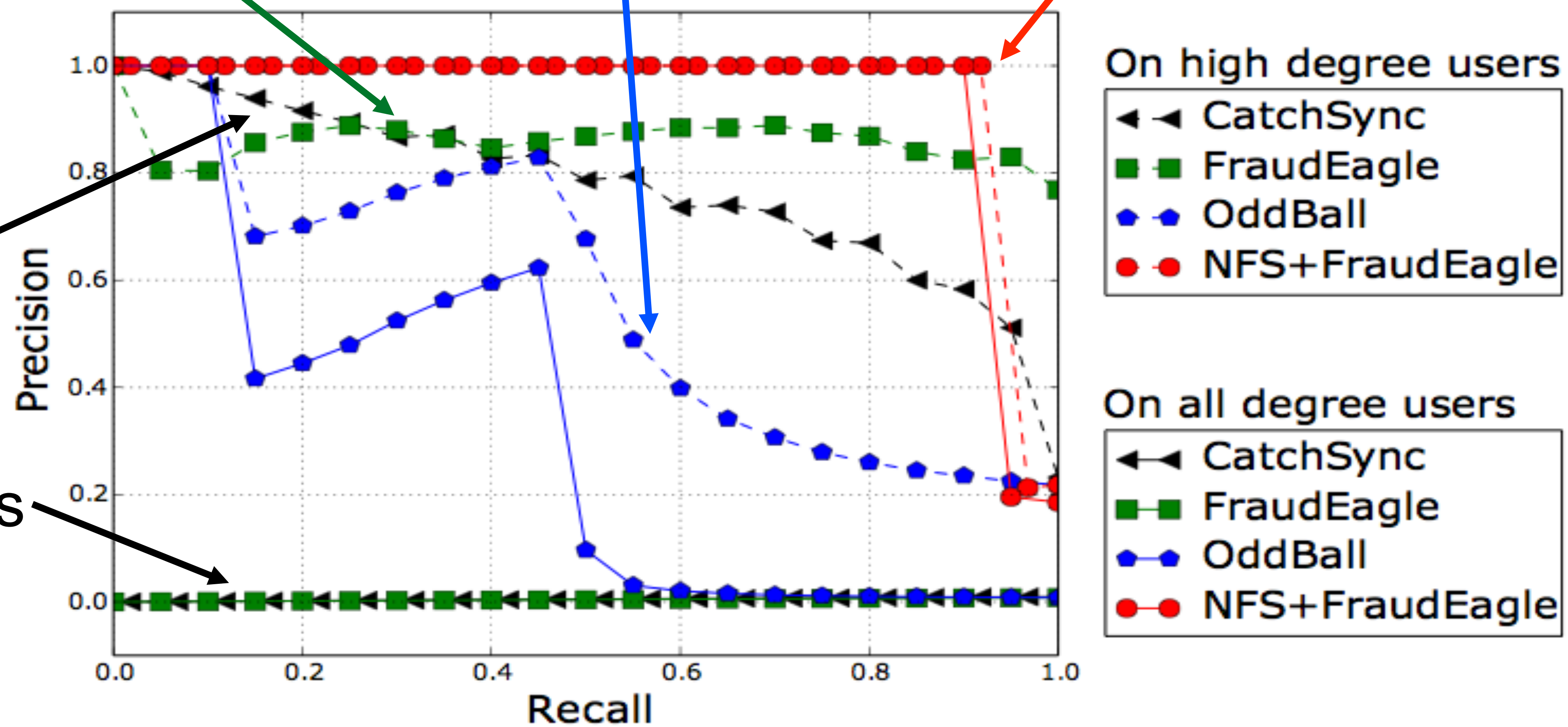
NFS on Synthetic Graphs

FraudEagle:
- Label propagation

OddBall:
- Detects near-cliques and star shapes

NFS + FraudEagle
- Solid line: all users
- Dash line: high-deg users

CatchSync
- Good on high-degree users
- Poor on all users



AUC of **Pre-Rec curve** on RTG2 (30% random camouflage)

NFS on Synthetic Graphs

Stony Brook University Computer Science

AUC of Pre-Rec Curve (Range [0, 1]; larger is better)

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

AUC on high-degree users

AUC on all users

GroupStrainer on Synthetic Graphs

Stony Brook University Computer Science

- Synthetic data generator **(SCI, ε)**:
  - Collusion with **Spammer Collusion Index (SCI) =** camouflage index

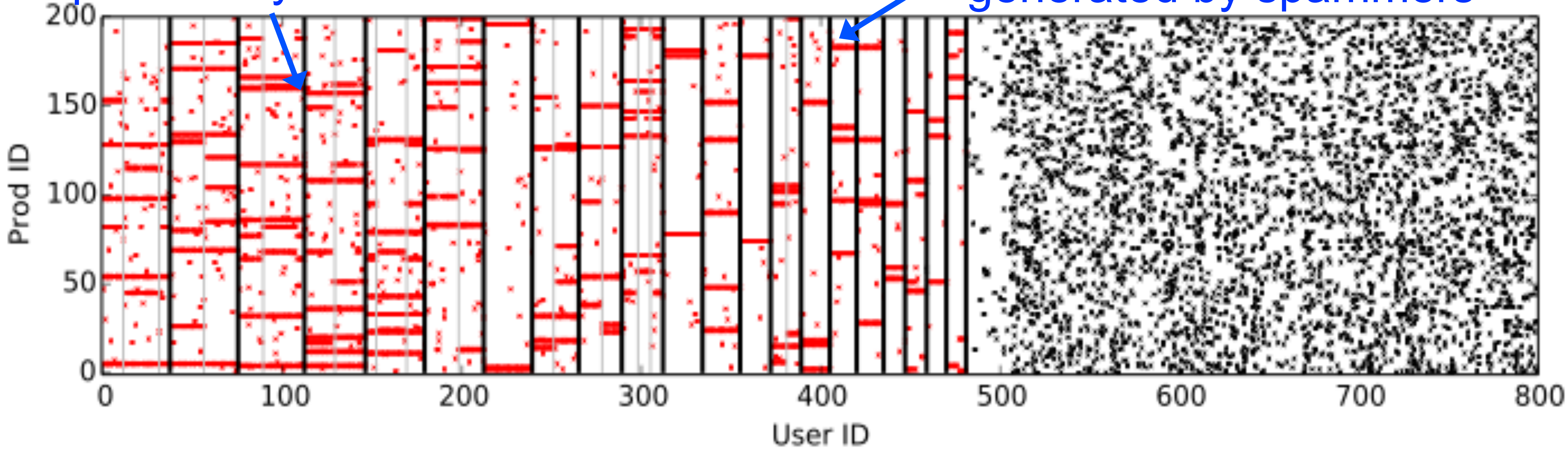
$$SCI(g) = \sum_{g_i, g_j \subset g, i \neq j} \frac{|t(g_i) \cap t(g_j)|}{|t(g_i) \cup t(g_j)|} / \binom{n}{2}$$

SCI equivalent to avg Jaccard similarity of groups' targets sets

  - **ε:** fraction of noise reviews (i.e. camouflage) over spam reviews.

Spammer groups separated by black bars

Red dots : reviews generated by spammers



Output of GroupStrainer on synthetic dataset (SCI = 0.5, ε = 0.2)

GroupStrainer on Synthetic Graphs

Stony Brook University
Computer Science

Performance of GroupStrainer for varying ε and SCI

Normalized Mutual Information (NMI) (range [0,1]; larger, better)

Corresponding 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

NMI is large even if large noise & little collusion

Performance on Real Datasets

Stony Brook University
Computer Science

All detected groups found suspicious (synchronized behaviors) in at least one aspect of time, rating, text

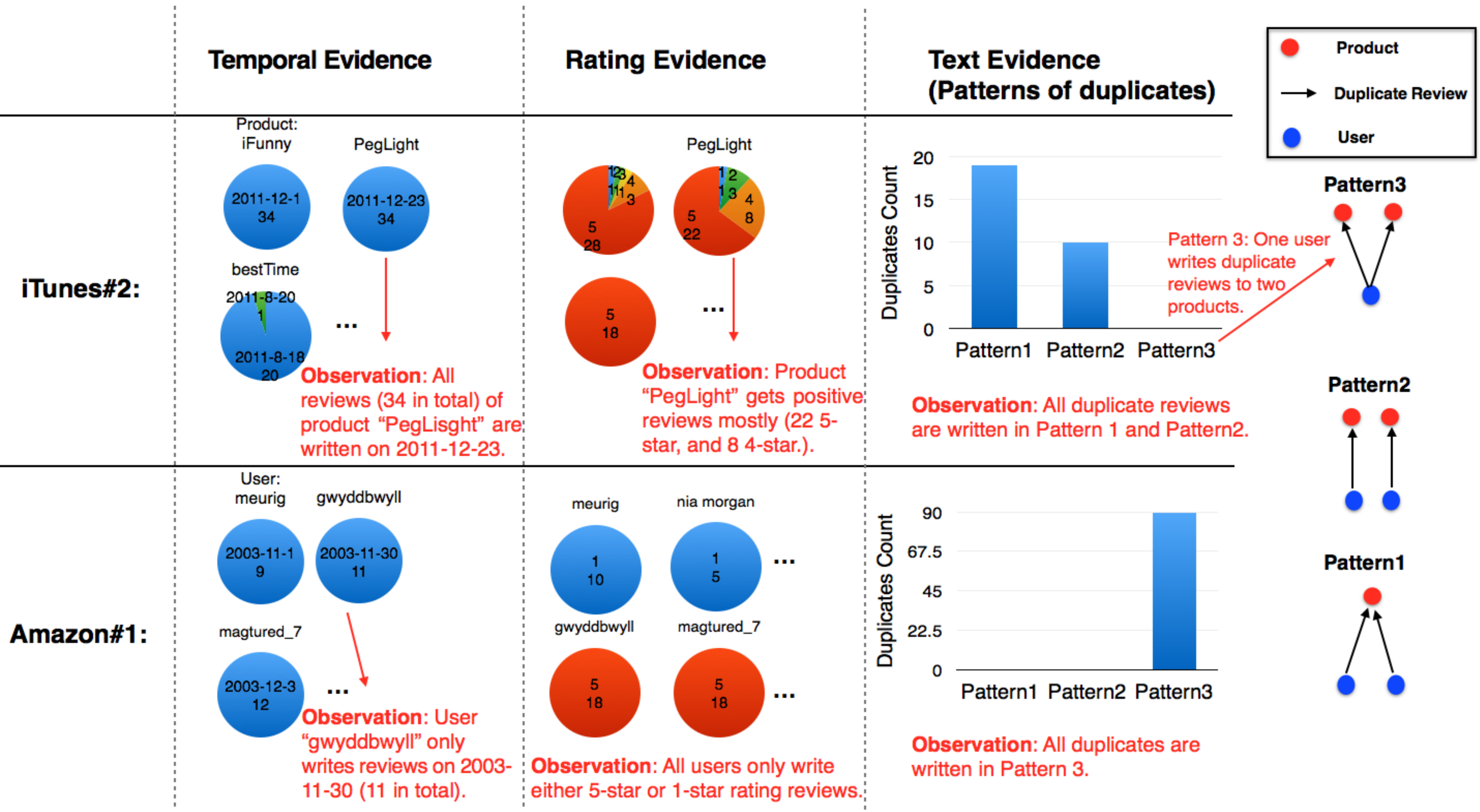
List of detected groups in Amazon

P: products, *U*: users, *t*: time, *: rating star, Dup: duplicates

Misnomer

	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					

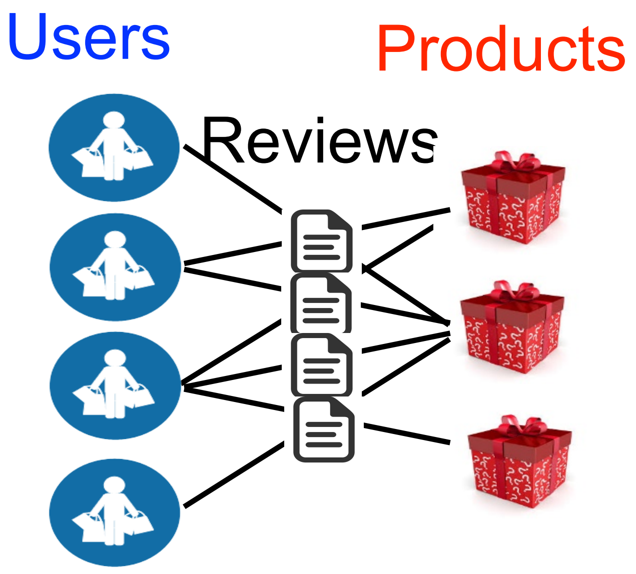
Case Study



Abundant evidence of suspicious behaviors in various patterns

Conclusion

- Two-step method to detect **spammer groups**:
  - 1. NFS: a measure of suspiciousness for products based on network footprints
  - 2. GroupStrainer: an efficient clustering algorithm to detect collusive spammers
- **Advantages**: unsupervised detection, adversarial robustness, sensemaking, and efficiency
- Validated on both synthetic and real-world data



Thank you!

Code and Data available:

http://www3.cs.stonybrook.edu/~juyye/

http://www3.cs.stonybrook.edu/~datalab/