AdaGraph: Adaptive Graph-Based Algorithms for Spam Detection in Social Networks

Amira Soliman(✉) and Sarunas Girdzijauskas

School of Information and Communication Technology,
Royal Institute of Technology (KTH), Stockholm, Sweden
{aaeh,sarunasg}@kth.se

Abstract. In the past years, researchers developed approaches to detect spam in Online Social Networks (OSNs) such as URL blacklisting, spam traps and even crowdsourcing for manual classification. Although previous work has shown the effectiveness of using statistical learning to detect spam, existing work employs supervised schemes that require labeled training data. In addition to the heavy training cost, it is difficult to obtain a comprehensive source of ground truth for measurement. In contrast to existing work, in this paper we present AdaGraph that is a novel graph-based approach for spam detection. AdaGraph is unsupervised, hence it diminishes the need of labeled training data and training cost. Particularly, AdaGraph effectively detects spam in large-scale OSNs by analyzing user behaviors using graph clustering technique. Moreover, AdaGraph continuously updates detected communities to comply with users dynamic interactions and activities. Extensive experiments using Twitter datasets show that AdaGraph detects spam with accuracy 92.3%. Furthermore, the false positive rate of AdaGraph is less than 0.3% that is less than half of the rate achieved by the state-of-the-art approaches.

Keywords: Unsupervised spam detection · Social networks · Distributed systems · Evolving graphs algorithms · Community detection

1 Introduction

With the widespread usage of user generated content in Online Social Networks (OSNs), spam has increased and has become an effective vehicle for malware and illegal advertisement distribution. Spam not only pollutes the content contributed by normal users, resulting in bad user experiences, but also misleads and even traps legitimate users. Furthermore, OSNs have also led to new methods of delivering spam, such as spammy apps, social bots, and fake accounts. These methods result in increasing social media spam to 355% in 2013 over 2012[1]. Spotting spammers is very challenging especially with the dynamic nature of social networks where activities and interactions among users evolve rapidly.

[1] http://nexgate.com/solutions/nexgate-social-media-security-stat-center/.

A. El Abbadi and B. Garbinato (Eds.): NETYS 2017, LNCS 10299, pp. 338–354, 2017.
DOI: 10.1007/978-3-319-59647-1_25

Additionally, the problem becomes more challenging due to the huge amount of data shared by users. Therefore, researchers have analyzed different spam strategies to design mechanisms to combat the spam activities from different prospectives, including studying the redirection chains of embedded URLs [1–3], analyzing textual content [4–6], as well as analyzing different friendship graph properties of spammers against those of legitimate users [4–6].

The research community has produced a substantial number of mechanisms for automated spam detection based on binary classification mechanisms. The design of such spam detection mechanisms in general is guided by the behavior dissimilarity exhibited by legitimate users than spammers. The central premise as proved in the existing work is that spammer behavior appears anomalous relative to normal user behavior along some features that could be extracted from textual content (i.e., content-based features such as number of URLs, hashtags and mentions used per post) and OSN friendship graph (i.e., graph-based features that are calculated from the friendship graph such as local clustering coefficient and betweenness centrality). However, all of the existing techniques rely on supervised binary classification methods [1,4,6–8].

Although the proposed binary classification methods succeed at detecting spam content, they implicitly require offline training with statistically sufficient and representative labeled training set of different user behaviors in order to achieve good detection coverage. This requirement itself is hard to satisfy, not to mention the difficulty of adapting to different behavior patterns that emerge in the future. Furthermore, the number of features required to discriminate spammers increases due to the diverse users activists in OSNs, the evolving spam patterns, as well as the limited the amount of labeled data. For example, Zhu et al. [8] use 1,680 different user activities in their supervised detection approach. Additionally, binary classification methods result in false positive rate that could range between 5.7% and 0.8% [7,9] resulting in some legitimate users are identified as spammers and get disconnected from the network. Particularly, derived from the remark that spammers hijack trending topics and include many URLs in their posts, content-based classification methods distinguish spammers by the extensive use of URLs, hashtags and mentions. Consequently, legitimate users such as the official news channels that continuously broadcast posts with diverse topics containing URLs and hashtags of the trending topics are going to be classified as spammers.

To address these issues, in this paper we propose AdaGraph[2] that is a novel unsupervised graph-based clustering technique for spam detection. Differently from existing work, AdaGraph constructs a *user similarity graph* that is created by connecting users with edges having weights that quantify their behavioral similarity. The essence of AdaGraph is to construct a user similarity graph that encodes within its topology a holistic view of all behavioral interactions and patterns of OSN users. Afterwards, AdaGraph performs graph clustering by applying community detection on top of the newly created graph.

[2] This work is under the umbrella of the iSocial EU Marie Curie ITN project (FP7-PEOPLE-2012-ITN).

In particular, we create a user-based feature vector to summarize both content and graph features associated with every user. Accordingly, the edges are created connecting users having weights equal to the cosine similarity of feature vectors of source and destination nodes[3]. Afterwards, AdaGraph detects communities on top of similarity graph to identify different behavioral patterns existing in the social network, then spots the spam patterns among the detected ones by applying some lexical analysis. Spam detection using graph-based clustering not only diminishes the training cost, but also achieves low false positive rate. Graph-based clustering provides meaningful insights to the existing behavioral patterns, therefore, categorizes the existing patterns into more homogeneous and accurate clusters than binary splitting as illustrated in Fig. 1. Hence, grouping users into multiple communities minimizes the chances of high false positive rates, specially for legitimate users with diverse and highly active behaviors such as news channel accounts. Clustering will group such accounts into a separate cluster with a closer distance to users having legitimate behavior pattern with diverse topics rather than the spam pattern that exhibit high URL and hashtags rate, yet in the same time has high similarity in the content. Hence, graph-based clustering provides more accurate results compared to binary classification without the need of the repetitive cost of maintaining up-to-date labeled training dataset.

However, centralized graph-based clustering techniques are not realistically scalable due to the huge number of users in current OSNs. Therefore, graph-based clustering algorithms must be developed as massively parallel clustering that eliminates the need of single centralized aggregation point. Even better, graph-based clustering can be implemented as fully decentralized solution to be applicable with currently emerging Decentralized Online Social Networks (DOSNs). DOSNs operate as distributed information management platforms on top of networks of trusted servers or P2P infrastructures [10]. Thus, DOSNs provide a privacy preserving alternative to current OSNs, where users have full control of their data. Accordingly, in AdaGraph we allow every node to independently process its data and only communicate with its direct neighbors. Additionally, AdaGraph adaptively updates similarity connections among nodes in the detected communities based on the newly received information integrated with the previously known without the need of recomputing from scratch. Hence, AdaGraph is capable of monitoring the behavioral changes and dynamically adapts to the evolving social activities and interactions among users. We have performed experiments, using Twitter datasets, to show the effectiveness of our proposed approach. The results show that AdaGraph provides more accurate spam detection rate with accuracy up to 92.3% and false positive rate less than 0.3%. Thus, AdaGraph outperforms the state-of-the-art techniques not only in plain performance figures, but also by removing the need of labeled data and offline training effort (since AdaGraph is unsupervised) as well as removing the scalability issues due to the fully decentralized and distributed nature of the algorithm.

[3] Users and nodes refer to the same meaning and are used interchangeably.

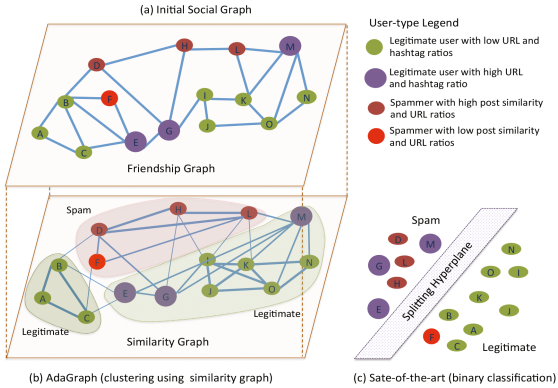


Fig. 1. Similarity-based clustering vs. Binary classification. (a) Initial social graph of OSN users having different behavioral patterns. (b) AdaGraph creates similarity graph and extracts communities that group users with similar behaviors. (c) Binary classification organizes all users in feature space to find the best splitting hyperplane.

Accordingly, our work offers the following contributions to the problem of spam detection:

- We propose unsupervised spam detection approach that requires no a priori labeling while maintaining low false positive rate,
- We propose a novel graph-based spam detection technique that detects spam using graph clustering on top of a constructed user similarity graph which encodes user behavioral patterns within its topology,
- We introduce adaptive algorithms that enable similarity-based community detection to evolve with respect to the behavioral changes of the users,
- Our proposed graph-based spam detection technique out-performs existing centralized binary classification,
- All of the above contributions are performed in purely distributed and decentralized manner.

The remainder of this paper is structured as follows. In Sect. 2 we list the features used for spam detection, whereas, in Sect. 3 we illustrate the core algorithms implemented in AdaGraph. Furthermore, in Section 3 we detail the lexical analysis method adopted to indicate the spammers communities among the detected ones. In Sect. 4 we present evaluation of AdaGraph. Finally, Sect. 5 shows the related work, then Sect. 6 concludes the paper.

2 Spam Detection Features

In this section, we first briefly describe the graph-based and content-based used in AdaGraph to compute user-based feature vectors.

2.1 Graph-Based Features

In this part we utilize the original social friendship graph connecting users. We consider the social network as undirected graph G = (V, E), where V is the set of nodes and E is the set of edges. $e_{ij} \in E$ denotes a relationship between nodes v_i and $v_j \in V$.

Definition 1. *Local Clustering Coefficient (LCC). Given $v_i \in V$, let $DF_i = \{v_j \in V | e_{ij} \in E\}$ be the direct friends of v_i. LCC_i represents the local clustering coefficient of v_i, and equals to:*

$$LCC(v_i) = \frac{|e_{jk} : v_j, v_k \in DF_i, e_{jk} \in E|}{|DF_i|(|DF_i| - 1)} \tag{1}$$

LCC is one of the graph-based features that are hard to fake [6]. LCC of a node is the ratio between the number of existing links among its direct neighbors and the number of links that could possibly exist among them [11]. LCC is used to quantify the extent to which the direct neighbors of a node are connected to each other. Due to decentralized nature of AdaGraph, we assume that every node calculates its LCC locally by keeping track of two-hop neighbors (i.e., neighbors of the neighbors).

Definition 2. *Average Neighbors Clustering Coefficient (ANCC). We define ANCC of node v_i as the average of LCCes computed by DF_i.*

ANCC is used to quantify the connectedness of the neighborhood of a node. Madden et al. [12] show that majority of OSN users are more skeptical regarding the acceptance of new friendship requests from strangers. Therefore, it is hard for spammers to have strongly connected neighborhood surrounding them. Thus, ANCCes of legitimate users are commonly higher than those of spammers.

2.2 Content-Based Features

A recent study [13] shows that spammers generate posts using complex templates such as finite-state machines to evade spam detection methods. Although, finite-state machines increase the number of different spam posts that can be generated, all of the generated posts follow a structured content, for example [mentions of other users + some text + URLs + hashtags of trending topics]. Furthermore, spam posts still have some words in common such as "look at this video" or "gain more", etc. Therefore, AdaGraph adopts the following content-based features.

Definition 3. *Average Posts Similarity (APS). Let P_i be the set of posts shared by v_i, and $pair_{(j,k)}$ be the pair of two posts p_j and p_k in P_i. We define the average posts similarity of v_i as follows:*

$$APS(v_i) = \frac{1}{\binom{|P_i|}{2}} \sum_{pair_{(j,k)} \in P_i} \frac{p_j \cap p_k}{p_j \cup p_k} \tag{2}$$

This feature leverages the similarity among the posts shared by a single user. We define post similarity using jaccard coefficient, such that for every post pair of a user, we divide the intersection (i.e., the number of common words in the post pair) by the total number of words in the post pair. Due to decentralized nature of AdaGraph, we assume that posts are publicly available and can be collected by other nodes.

Definition 4. *Mentions Ratio (MR). Spammers add mentions to random users to increase the visibility of their content. Accordingly, we define MR of a user u_i as the number of mentions which refer to a username not included in DF_i to the total number of posts generated by u_i.*

Definition 5. *URL Ratio (UR). Spammers embed malicious URLs in their posts to direct the users to their websites. Thus, we define UR as the the ratio of the number of posts containing a URL to the total number of posts a user has (i.e., $|URLs|/|P_i|$).*

Definition 6. *Hashtags Ratio (HR). Hijacking trending topics in OSNs has been a widely adopted strategy among spammers to reach wider audience. Therefore, we define HR as the number of trending topics associated with user posts to the total number of posts (i.e., $|Hashtahs|/|P_i|$).*

3 Graph-Based Spam Detection

In this section, we present the core of AdaGraph. First, we illustrate the construction of user similarity graph, followed by the details of the local clustering algorithm. Afterwards, we present the quick community adaptation algorithm used for tracing the evolution of users behaviors represented in detected communities over time. Furthermore, we discuss the computational complexity of AdaGraph and present the adopted lexical analysis approach to spot spammers communities among detected ones.

3.1 Similarity Graph Construction

The first step of AdaGraph is to construct users similarity graph from the social graph. To build a massively parallel approach, we allow every node in the social graph to participate in similarity graph construction. Initially, every node starts by creating similarity edges among itself and its social neighbors. The edges are created by connecting any pair of nodes having cosine similarity of their feature vectors greater than specific threshold. So as, the weight of an edge connecting node i and node j equals to:

$$w(e_{ij}) = \frac{x_i.x_j}{||x_i||.||x_j||} \tag{3}$$

where, x_i is the feature vector of node i and x_j is the feature vector of node j. Particularity, the feature vector of a node has the following values [LCC, ANCC,

APS, MR, UR, HR] as defined in Sect. 2. If the weight $w(e_{ij})$ is greater than the threshold ϵ, then an edge connecting node i and node j is added to the graph with weight equals to $w(e_{ij})$ (see Fig. 1, the thickness of an edge reflects its weight). Afterwards, every node enlarges the similarity graph further by exploring the possibility of creating more similarity edges with the neighbors of its currently direct neighbors.

3.2 Clustering by Community Detection

Our objective is to find the topological communities inside the constructed similarity graph. Let us first define similarity graph as an undirected weighted graph G = (V, E), where V is the set of nodes and E is the set of similarity edges, where $e_{ij} \in E$ denotes cosine similarity between nodes v_i and $v_j \in V$ that is computed as defined in Eq. 3. Commonly, finding communities is well-know as community detection and is defined as:

Definition 7. *Community Detection. A community detection C, also known as graph clustering, is a mapping*

$$C : G \rightarrow G'_1 \times \ldots \times G'_n \tag{4}$$

that partitions G into n non-empty, node-disjoint subgraphs $G'_1 \times \ldots \times G'_n$ representing a set of communities or clusters. A widely used quality measure for community detection is the modularity Q of the clustering C(G) [14], that assigns a quality value q to the clustering C(G) defined by

$$q := \sum_i (w(e_{ii}) - b_i^2) \tag{5}$$

where $b_i = \sum_j w(e_{ij})$, where e_{ij} represents an edge in community i for which the target node of the edge lies in community j. The higher the quality value q is, the better the detected community is. One possible definition for C is to maximize Q over all clustering C(G) [14].

AdaGraph employs recently developed decentralized diffusion-based community detection strategy [15]. In particular, every node in the similarity graph starts by joining the node with the maximum cosine similarity among its direct friends to form a community. Afterwards, in successive iterations every node chooses to quit its current community and join one of its neighbour's if this brings some modularity gain. As described in method *selectCommunity* in Algorithm 1, nodes select the dominant community in their neighborhood to join (i.e., the community with the highest sum of weights). This step is iteratively repeated until no node wants to change its community as it already represents the dominant one of all its neighbors. Thereafter, the topological communities detected in the similarity graph represent the different user behavioral patterns.

Algorithm 1. Community Detection Methods

Result: Community Structure C_{t+1}
Procedure `selectCommunity(`*node u*`)`
  forall the $C \in NeighborCommunity(u)$ **do**
    $q(C) \leftarrow sum(w_{e_{uj}}) | C_j = C$
  end
  $C_u \leftarrow C_j | q(C_j) = max(q(C))$
Procedure `changeCommunity(`*node u*`)`
  $C_{u_{new}} \leftarrow$ selectCommunity(u)
  if $C_u \neq C_{u_{new}}$ **then**
    $C_u \leftarrow C_{u_{new}}$
    forall the $x \in Neighbor(u)$ **do**
      changeCommunity(x)
    end
  end

3.3 Community Structure Adaptation

OSNs are dynamic by nature due to rapidly evolving social activities and interactions among users. Therefore, the constructed similarity graph must be continuously updated to cope with evolving users' behaviors. Thus, we have integrated adaptive modularity-based methods for identifying and tracing the changes in the communities structure of the constructed similarity graph. The similarity graph is updated by either inserting or removing a node or set of nodes, or by either introducing or deleting an edge or set of edges. We have modeled these graph changes as a collection of simple events namely: *newNode*, *removeNode*, *newEdge* and *removeEdge*. AdaGraph starts by extracting initial community structure C_0, by detecting the communities exist in the first snapshot of the network. Thereafter, this initial structure is continuously updated for the successive snapshots by applying different adaptation methods as illustrated in Algorithms 2 and 3.

Algorithm 2. Node Simple Events

Result: An updated Community Structure C_{t+1}
Procedure `newNode(`*node u*`)`
  $C_u \leftarrow$ selectCommunity(u)
  Update C_{t+1} : $C_{t+1} \leftarrow (C_t \setminus C_u) \cup (C_u \cup u)$
Procedure `removeNode(`*node u*`)`
  $C_u \leftarrow (C_u \setminus u)$
  forall the $v \in Neighbor(u)$ **do**
    removeEdge(e_{vu})
  end

- **newNode(V+u)**: a new node u with its associated edges are introduced, such that u could come with no or more than one new edge(s). When u joins the similarity graph, it assigns itself to the dominant community in its neighborhood as illustrated in method *newNode*.
- **removeNode(V−u)**: node u with its adjacent edges are removed from the graph. As shown in methods *removeNode*, when an existing node u is of a community C is removed, all of its adjacent edges are removed as a result. Consequently, the resulting community structure might change, hence, neighbors of that removed node re-evaluate their community memberships as illustrated in the next method *removeEdge*.

Algorithm 3. Edge Simple Events

Result: An updated Community Structure C_{t+1}
Procedure `newEdge(`*edge* e_{vu}`)`
  if *v and u are new nodes* **then**
    $C_{t+1} \leftarrow C_t \cup \{v, u\}$
  else if $C_v = C_u$ **then**
    $C_{t+1} \leftarrow C_t$
  else
    changeCommunity(u)
    changeCommunity(v)
Procedure `removeEdge(`*edge* e_{vu}`)`
  if *(v, u) is a single edge* **then**
    $C_{t+1} \leftarrow (C_t \setminus \{v, u\}) \cup \{v\} \cup \{u\}$
  else if *either v (or u) is of degree one* **then**
    $C_{t+1} \leftarrow (C_t \setminus C_v) \cup \{v\} \cup \{C_v \setminus v\}$
  else if $C_v \neq C_u$ **then**
    $C_{t+1} \leftarrow C_t$
  else
    changeCommunity(u)
    changeCommunity(v)

- **newEdge(E+e)**: a new edge e is introduced, we can divide it further into two cases: an intra-community edge (both nodes belong to the same community) or an inter-community edge (connecting two communities). In the first case, no change happens to the community structure (as detailed in method *newEdge*). Yet, the interesting situation happens when e is an inter-community edge, as its presence could possibly make source and destination nodes change their community memberships. Consequently, these nodes notify their neighbors in case of change, so as cascading updates could take place if further changes are required (as detailed in method *changeCommunity*).
- **removeEdge(E−e)**: an existing edge e in the graph is removed. Similarly to edge addition, edge removal can be divided into two case, such that the edge to be removed e is either an inter-community edge or intra-community edge. In the first case, the removal of e will strengthen the current community structure and cause no change to it. However, in the second case,

edge removal might cause community split. Therefore, the edge source and destination nodes re-evaluate their community memberships and notify their neighbors in case of change.

3.4 Lexical Analysis of Posts

As aforementioned, the core of AdaGraph is to detect different behavioral patterns in the user similarity graph by performing community detection. However, spotting spam patters among detected ones is not straightforward. So as, further lexical and semantic analysis is required to efficiently spot spammers communities among extracted ones. Specially, spammers can use automated spinning to avoid duplicate detection, such that they can create new versions with vaguely similar meaning but sufficiently different appearance. Therefore, in AdaGraph we apply lexical analysis of the most frequent words to determine if those words or their synonyms are commonly used by spammers. AdaGraph integrates Gavagai lexicon[4] that learns the words synonyms and their related n-grams terms. Accordingly, we identify a set of trusted nodes in the social graph and these nodes are responsible for labeling any of detected communities as spam if the majority of the users belonging to these communities frequently use spam words or their lexical related terms.

3.5 Complexity Analysis

The model cost is expected to be low given that every node performs its local computation independently of the other nodes. We discuss the complexity of AdaGraph in terms of communication traffic among all the nodes in the OSN. By our adopted work for decentralized community detection, the algorithm complexity is $\mathcal{O}(N * D * R)$, where N is the total number of users in the similarity graph, D is the average node degree, and R is the total number of rounds needed for the algorithm to converge[5] [15]. This step requires that all the nodes are online at the time of its execution; however, it is also a process that is performed once and that is incrementally updated only. Moreover, as we demonstrate through experiments on real OSN data, the convergence time of our solution is very realistic and achievable (see Sect. 4.3).

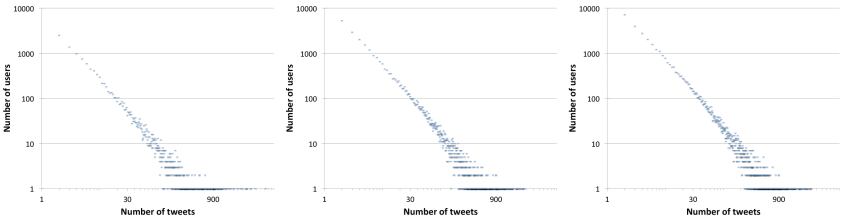
4 Evaluation

AdaGraph applies vertex-centric approach which is proved to be scalable, efficient and fast. Our algorithms are implemented in GraphLab[6], with two different distributed execution modules. In the first module, nodes participate in creating the similarity graph using their feature vectors. Thereafter, the control is moved

[4] Available via http://lexicon.gavagai.se/.
[5] R depends on the topological properties of the underlying graph.
[6] https://turi.com/products/create/.

Table 1. Twitter datasets used in our experiments.

Twitter dataset	US_Active	UK_Active	US_Passive
Tweets	453,519	489,484	360,927
Legitimate accounts	17,322	19,312	12,128
Suspended accounts	2,072	1,617	3,109
Social-graph edges	1,357,806	1,187,036	2,349,314
Similarity-graph edges	2,149,414	2,297,150	3,339,617



(a) US_Active (b) UK_Active (c) US_Passive

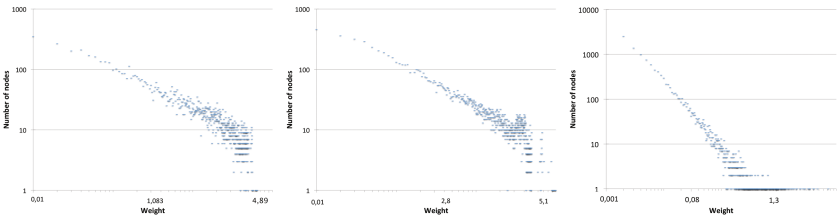
Fig. 2. The tweeting distribution in Twitter datasets in log scale.

to the second module that performs the community detection algorithm. In the following subsections, we thoroughly evaluate the performance of AdaGraph in terms of the accuracy of spam detection. We compare AdaGraph with different centralized and supervised binary classification approaches, utilizing the Weka tool[7], namely: K-means (KM) with number of clusters = 2, Decision Tree (DT) and Random Forest (RF).

4.1 Datasets

We have collected our dataset from Twitter using Twitter streaming API[8] from May 2015 to July 2016. We have accessed Twitter's API using privileged accounts, collecting users' tweets and the social graph connecting these users. In order to identify the spammers, we have queried the status of all accounts regularly to check if any got suspended for abusive behavior. Upon suspension, we identify suspended accounts as spammers. Table 1 lists the details of the collected datasets. The first two datasets (US_Active and UK_Active) are collected from users with high level of posting tweets located in United States and United Kingdom, respectively. Yet, the third dataset (US_Passive) is collected from users located in United States with low level of posting activity. Accordingly, in Fig. 2 we show the tweeting distribution for the collected datasets in log scale. As shown, tweeting distribution follows power law probability distribution, such that there is uneven distribution of number of tweets being posted

[7] http://www.cs.waikato.ac.nz/ml/weka/.
[8] https://dev.twitter.com/rest/public.



(a) US_Active (b) UK_Active (c) US_Passive

Fig. 3. The weight distribution in the generated similarity graphs for Twitter datasets in log scale.

by users. Majority of users post few tweets, whereas there is small number of highly active users who post large number of tweets.

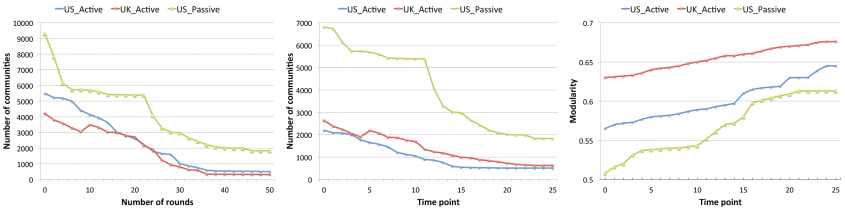
4.2 Generated Similarity Graph

As aforementioned, the user similarity graph is constructed in a fully decentralized manner, such that each node explores its surrounding neighborhood progressively to add further similarity edges. Furthermore, as mentioned in Sect. 3.1, nodes add similarity edges if the similarity weight is greater than the threshold ϵ. We allow nodes to determine freely the value of ϵ, such that each node computes the average weight of its current edges, and sets the average weight as value for ϵ. As shown in Table 1, the average number of added edges in the similarity graph is almost equal to 50% of the existing edges in the social graph. Accordingly, AdaGraph connects only highly similar nodes instead of creating a fully connected graphs.

Figure 3 depicts the similarity weight distribution obtained for each dataset in log scale. As shown, the similarity weight distribution follows power law probability distribution similarly to the tweeting distribution. Furthermore, the similarity weight distribution spans over wider range in US_Active and UK_Active compared to US_Passive. Particularly, in US_Passive 91.5% of the similarity weight is less than 0.25, and this resulted from the low post frequency of users in this dataset. Therefore, we can infer that the more active posting behavior of users, the more strong edges are be added in the similarity graph. Additionally, AdaGraph can successfully adapt to different social activities of the users and accordingly create the user similarity graph to reflect the underlying user behavior.

4.3 Adaptive Community Detection

As aforementioned, every node repeatedly runs the community detection, until communities structure does not change any more (i.e., the convergence is reached). Figure 4(a) depicts the number of rounds required till convergence, and number of extracted communities per round. As shown, in the very beginning



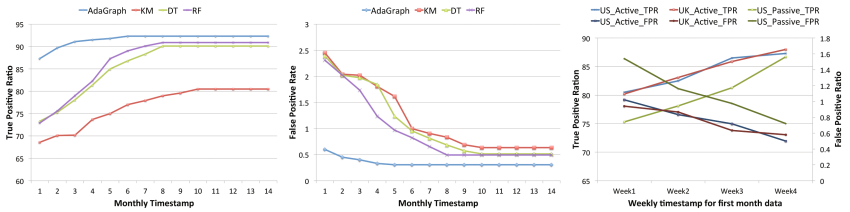
(a) Convergence Rate (b) Detected Communities (c) Modularity Gain

Fig. 4. Community detection results of AdaGraph. (a) Number of iterations required for convergence. (b) Number of detected communities per each snapshot. (c) The modularity gain obtained per each snapshot.

the number of communities is very large, every node starts to form a community with one of its direct neighbors. However, over time nodes join the dominant communities in their neighborhood, as a result the communities start to merge and the number of communities continues to decrease. In order to identify the communities that contain spammers, we have construct a list of 500 words that are commonly used by spammers associated with their semantically similar terms and n-grams (see Sect. 3.4). Further, for every node we select the most frequent words used in its tweets. Accordingly, the collected word list per community is checked against a list of common spam words. A community is identified as spam if majority (i.e., more than 50%) of its members use common spam words in their tweets. The results show that the percentage of spam communities is 17.3%, 21.6% and 23.5% in US_Active, UK_Active and US_Passive, respectively.

Additionally, we study the adaptability of AdaGraph with dynamic and evolving graphs. We started by loading 50% to form the basic structure, such that we constructed the similarity graph using only 50% of nodes from the social graph and 50% of their associated tweets. Afterwards, we simulated the network evolution by adding the remaining nodes/tweets via a series of 25 growing snapshots. Figure 4(b) depicts the number of detected communities per snapshot as well as Fig. 4(c) shows the resulted modularity of the detected communities per snapshot. Furthermore, we have noticed that the changes caused by incrementally adding the snapshots are localized, such that on average 15% to 17% of old nodes got affected by the change and re-evaluate their communities membership. Consequently, AdaGraph dynamically adapts to the topological changes of evolving graphs. Moreover, AdaGraph adapts incrementally with no need to start community detection from scratch.

Furthermore, we study the effect of ϵ as a graph sparsification parameter, as well as the ability of AdaGraph to extract communities in denser graphs. Accordingly, we have repeated the community detection experiments with another generated user similarity graphs in which all edges are added even though the weight is less than ϵ. Consequently, the new generated user similarity graphs are denser than those generated having edges with weight greater than ϵ. The obtained results show that AdaGraph maintains the same convergence rate and



(a) True positive rate (b) False positive rate (c) AdaGraph with limited data

Fig. 5. The performance gain achieved by AdaGraph compared with centralized and supervised methods. The reported values are the average of achieved performance across the three datasets. (c) AdaGraph performance using only the first month snapshot of the data.

structure of detected communities with denser graphs, though the execution time is almost double the execution time of ϵ sparsificated graphs. Specifically, with ϵ sparsificated graphs the execution time is 15.4, 17.7, and 13 min for US_Active, UK_Active and US_Passive, respectively. On the other hand, with denser graphs the execution time is 39, 35.9, and 26.9 min for US_Active, UK_Active and US_Passive, respectively. Therefore, by disregarding a large fraction of low-weight edges that are insignificant for the task, running times of community detection algorithms are reduced.

4.4 Performance Comparison

We calculate the accuracy of AdaGraph using True Positive Rate and False Positive Rate, that are defined as the following: (1) True Positive Rate (TPR): we calculate TPR as the fraction of spammers that are successfully detected. (2) False Positive Rate (FPR): we calculate FPR as the fraction of legitimate users that are identified as spammers.

We have updated all of the supervised machine learning algorithms to be performed in online learning fashion. Instead of executing them in batch learning manner that uses the entire training dataset at once, we have used the monthly updates of the data in a sequential order to update the predictors by retraining them with misclassified data points from future data. On the other hand, AdaGraph is already developed to capture the evolving changes in social network. For the comparison, we update the user similarity graph with sequence of monthly data.

Figure 5 depicts the detection performance comparison of AdaGraph with the different centralized and supervised classification methods. As shown, AdaGraph outperforms all binary classification methods especially when limited data is available. Specifically, the gap between AdaGraph and other methods in prediction accuracy for the first month is around 14,1%. Namely, the performance of supervised classification methods gets increased as the available training data increases (i.e., starting from the sixth month). On the other hand, AdaGraph creates an evolving similarity graph and continuously clusters users more accurately

from the first timestamp, such that TPR of AdaGraph is the highest (92.3%). Furthermore, AdaGraph follows a decentralized approach that enables to process small chunks of data in parallel, as well as AdaGraph requires no retraining as supervised classification methods do. Thus, AdaGraph rapidly adapts to concept drift that occurs in the system (user behavior), while other approaches require retraining with the new emerging patterns.

Furthermore, AdaGraph has the lowest FPR, which means that graph-based clustering successfully detect spammers with minimum effect on the legitimate users. Specifically, we can see that FPR in AdaGraph can be steadily maintained under 0.3%, as shown in Fig. 5(b), while the rate of RF method (the best binary classification method) starts with 2% and drops to 0.39% with increase of the training data. Consequently, the community detection approach adopted in AdaGraph perfectly categorizes the existing behavioral patterns into more homogeneous and accurate clusters than binary classification.

Additionally, we have further analyzed AdaGraph considering only the data collected in the first snapshot. In this experiment, we want to study the minimum number of posts needed to achieve good TPR meanwhile the FPR is kept low. Figure 5(c) depicts the weekly detection performance of AdaGraph in the first month. As shown, lowest TPR of AdaGraph is more than 75% while the FPR is less than 1.6% during the first week when the average number of available posts is 14 post across the three datasets. Hence, AdaGraph has an acceptable accuracy with very limited data. The first key reason behind AdaGraph good performance is the hybrid features that AdaGraph employs, i.e., the graph-based and content based features. Secondly and most importantly, the community detection algorithm categorizes user into more homogeneous and accurate clusters than binary classification.

5 Related Work

The first family of spam detection mechanisms includes techniques using blacklists to identify URL on OSNs websites directing to spam content [1,2]. However, URL blacklisting has several practical challenges. First, those blacklists are publicly available, hence spammers can evade them by changing their domain names or hiding them behind some redirecting pages. Second, URL blacklisting becomes ineffective with the spread usage of URL shortening services such as bit.ly and t.co. Therefore, different techniques have been proposed to analyze the redirection chains of URLs and their correlations [3]. Yet, those techniques are not designed as online detection tools, since they either have long lag-time or limited efficiency.

Furthermore, a rich corpus of research work lies in adopting supervised machine learning based methods using hybrid features extracted from textual content and OSN friendship graph. For example, Hongyu et al. [4] propose to train a binary classifier with hybrid features including user social degree, yet spammers can increase their social degree by purchasing more followers. Thus, Yang et al. [6] employ graph-based features that are hard to fake such as local

clustering coefficient and betweenness centrality. More recently, [16] suggests an unsupervised solution to spam detection based on sybil defense mechanism. The proposed scheme starts by identifying non-spammers (i.e., non-sybils) by applying a clustering algorithm on social graph. The authors focus their analysis on intensive URL sharing, yet instead of using URL blacklisting, they add new user-link edges to the social graph by connecting users sharing the same URL. However, the assumption that sybil nodes form tight-knit communities does not presist as shown in recent studies [17].

6 Conclusion

In this paper, we have introduced AdaGraph that is a novel decentralized and unsupervised spam detection framework in contrast to existing centralized and supervised approaches. AdaGraph resembles graph-based spam detection technique that detects spam using graph clustering on top of a newly constructed user similarity graph which encodes within its topology a holistic view of all behavioral interactions and patterns of OSN users. More importantly, AdaGraph integrates community detection algorithm that categorizes the existing user behavioral patterns into more homogeneous and accurate clusters than binary classification. The proposed approach achieves detection accuracy upto 92.3% and false positive rate less than 0.3%. Additionally, AdaGraph is scalable and massively parallel that suitably fits DOSNs and OSNs environments.

References

1. Thomas, K., Grier, C., Ma, J., Paxson, V., Song, D.: Design and evaluation of a real-time url spam filtering service. In: SP Symposium, pp. 447–462. IEEE (2011)
2. Levchenko, K., Pitsillidis, A., Chachra, N., Enright, B., Félegyházi, M., Grier, C., Halvorson, T., Kanich, C., Kreibich, C., Liu, H., et al.: Click trajectories: end-to-end analysis of the spam value chain. In: SP Symposium, pp. 431–446. IEEE (2011)
3. Lee, S., Kim, J.: Warningbird: detecting suspicious urls in twitter stream. In: NDSS (2012)
4. Gao, H., Chen, Y., Lee, K., Palsetia, D., Choudhary, A.N.: Towards online spam filtering in social networks. In: NDSS (2012)
5. Yang, C., Harkreader, R.C., Gu, G.: Die free or live hard? empirical evaluation and new design for fighting evolving twitter spammers. In: Sommer, R., Balzarotti, D., Maier, G. (eds.) RAID 2011. LNCS, vol. 6961, pp. 318–337. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23644-0_17
6. Yang, C., Harkreader, R., Gu, G.: Empirical evaluation and new design for fighting evolving twitter spammers. Inf. Forensics Secur. **8**(8), 1280–1293 (2013)
7. Amleshwaram, A.A., Reddy, N., Yadav, S., Gu, G., Yang, C.: Cats: characterizing automation of twitter spammers. In: COMSNET 2013, pp. 1–10. IEEE (2013)
8. Zhu, Y., Wang, X., Zhong, E., Liu, N.N., Li, H., Yang, Q.: Discovering spammers in social networks. In: AAAI 2012 (2012)
9. Martinez-Romo, J., Araujo, L.: Detecting malicious tweets in trending topics using a statistical analysis of language. Expert Syst. Appl. **40**(8), 2992–3000 (2013)

10. Datta, A., Buchegger, S., Vu, L.-H., Strufe, T., Rzadca, K.: Decentralized online social networks. In: Handbook of Social Network Technologies and Applications, pp. 349–378. Springer, New York (2010)
11. Dorogovtsev, S.N., Mendes, J.F.: Evolution of networks. Adv. Phys. **51**(4), 1079–1187 (2002)
12. Madden, M., Lenhart, A., Cortesi, S., Gasser, U., Duggan, M., Smith, A., Beaton, M.: Teens, social media, and privacy, vol. 21. Pew Research Center (2013)
13. Chen, C., Zhang, J., Xiang, Y., Zhou, W., Oliver, J.: Spammers are becoming "smarter" on twitter. IT Prof. **18**(2), 66–70 (2016)
14. Newman, M.E.: Modularity and community structure in networks. Proc. Natl. Acad. Sci. **103**(23), 8577–8582 (2006)
15. Rahimian, F., Girdzijauskas, S., Haridi, S.: Parallel community detection for cross-document coreference. In: WIC, vol. 2, pp. 46–53. IEEE/ACM (2014)
16. Tan, E., Guo, L., Chen, S., Zhang, X., Zhao, Y.: Unik: unsupervised social network spam detection. In: CIKM 2013, pp. 479–488. ACM (2013)
17. Yang, Z., Wilson, C., Wang, X., Gao, T., Zhao, B.Y., Dai, Y.: Uncovering social network sybils in the wild. In: TKDD 2014, vol. 8, no. 1, p. 2 (2014)