DeBot: Twitter Bot Detection via Warped Correlation

Conference Paper · December 2016

DOI: 10.1109/ICDM.2016.0096

Nikan Chavoshi
University of New Mexico
13 PUBLICATIONS **643** CITATIONS

Hossein Hamooni
Visa Inc.
13 PUBLICATIONS **654** CITATIONS

Abdullah Mueen
University of New Mexico
109 PUBLICATIONS **5,499** CITATIONS

DeBot: Twitter Bot Detection via Warped Correlation

Nikan Chavoshi, Hossein Hamooni, Abdullah Mueen
Department of Computer Science, University of New Mexico
{chavoshi, hamooni, mueen}@unm.edu

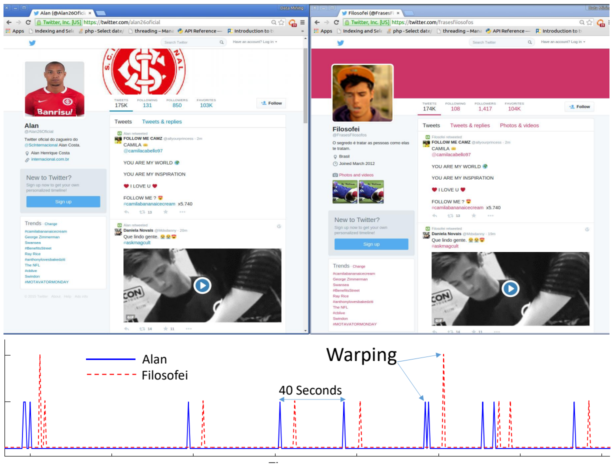
***Abstract*—We develop a warped correlation finder to identify correlated user accounts in social media websites such as Twitter. The key observation is that humans cannot be highly synchronous for a long duration; thus, highly synchronous user accounts are most likely bots. Existing bot detection methods are mostly supervised, which requires a large amount of labeled data to train, and do not consider cross-user features. In contrast, our bot detection system works on activity correlation without requiring labeled data. We develop a novel lag-sensitive hashing technique to cluster user accounts into correlated sets in near real-time. Our method, named DeBot, detects thousands of bots per day with a 94% precision and generates reports online everyday. In September 2016, DeBot has accumulated about 544,868 unique bots in the previous one year.**

We compare our detection technique with per-user techniques and with Twitter's suspension system. We observe that some bots can avoid Twitter's suspension mechanism and remain active for months, and, more alarmingly, we show that DeBot detects bots at a rate higher than the rate Twitter is suspending them.

I. Introduction

Social media websites allow users to communicate and share ideas. Automated accounts, called bots, abuse social media by posting unethical content [1], supporting sponsored activities [12], and selling accounts [17]. Social media sites, like Twitter, frequently suspend abusive bots [19]. However, current bot detection methods consider accounts independent of each other (i.e. per-user detection), and are mostly supervised [20][8]. We propose a novel *unsupervised* approach that identifies bots using correlated user activities. Figure 1 shows two Twitter accounts who do not follow each other but are correlated in tweeting activities. More examples of correlated accounts and a video capture of two correlated accounts are available in [2]. An analysis of significance of correlation in detecting bots is published in [4]. In this paper, we describe the DeBot system architecture and experimentally compare bot detection performance with respect to existing per-user methods.

On the data mining front, we develop a system called **DeBot** which correlates millions of users in near real-time to identify bot accounts. Traditional correlation coefficients such as Pearson's are non-elastic and are not suitable for Twitter activity time series because of *warping* induced by various factors: bot controllers, network delays, and internal processing delays in Twitter. An example of warping in activity time series is shown in Figure 1(bottom), where two users, `alan26offical` and `FiIosofei`, tweeted and retweeted many identical pairs of tweets with exactly ten seconds of lag and occasional warping. We allow time-warping by calculating warped correlation using the Dynamic Time Warping (DTW) distance for time series [10]. In our example, the warped correlation between the two users is 0.99, which is much higher than cross-correlation (0.72) and Pearson's correlation (0.07). To perform a large number of pair-wise warped correlation calculations efficiently, we develop a *lag-sensitive hashing* technique which hashes the users of the tweets into buckets of suspiciously correlated users. We show that lag-sensitive hashing is advantageous over regular random projection-based hashing in capturing warping correlations. The system then validates the correlations between the suspected users with account-specific listeners and outputs the valid bots. Our system collects tweets from the Twitter API at 48 tweets per seconds, which is the maximum rate allowed, and reports correlated accounts in daily batches.



Alan
@Alan26Oficial
Twitter oficial do zagueiro do @ScInternacional Alan Costa.
Alan Henrique Costa
internacional.com.br

Alan retweeted
FOLLOW ME CAMZ @allyourprincess · 2m
CAMILA 👑
@camilacabello97
YOU ARE MY WORLD 🌍
YOU ARE MY INSPIRATION
❤ I LOVE U ❤
FOLLOW ME ? 😍
#camilabananaicecream x5.740

Alan retweeted
Daniela Novais @Mdsdanny · 20m
Que lindo gente. 😭😭😍
#askmagcult

Filosofei
@FrasesFilosofos
O segredo é tratar as pessoas como elas te tratam.
Brasil
Joined March 2012

Filosofei retweeted
FOLLOW ME CAMZ @allyourprincess · 2m
CAMILA 👑
@camilacabello97
YOU ARE MY WORLD 🌍
YOU ARE MY INSPIRATION
❤ I LOVE U ❤
FOLLOW ME ? 😍
#camilabananaicecream x5.740

Filosofei retweeted
Daniela Novais @Mdsdanny · 19m
Que lindo gente. 😭😭😍
#askmagcult

Fig. 1. (top) Two highly correlated Twitter accounts: Alan (left) and Filosofei (right). (bottom) Six-minutes of correlated activities from Alan and Filosofei.

Our contribution in this work is mainly twofold: developing the warped correlation finder and using this finder to detect bots. Specifically:

- We develop a near real-time system, DeBot, which is the first (to our knowledge) unsupervised method to detect bots in social media. Our system detects more bots than existing supervised techniques.
- We develop a novel lag-sensitive hashing technique to quickly group correlated users based on their warping correlations. This allows us to cross-match millions of activity series under time warping, which has never been attempted before.
- We empirically show that DeBot has 94% precision and is able to detect bots that other methods fail to spot. We find that bots can be functionally grouped and that their number is growing at a high rate.

The rest of the paper is organized as follows: We start with a quick background on correlation computation in Section II. We describe our core techniques in Section III, including the never-ending correlation tracker and bot clustering algorithm. We perform a comprehensive evaluation of our method in Section IV and finally conclude in Section V. An expanded version of our paper is available in [2] containing more experiments and discussion of related work.

II. Background and Notation

The activity signal of a user in social media consists of all the actions the user performs in a sequence of observations. Actions include posting, sharing, liking, tweeting, retweeting, and deleting. The sequence of timestamps of the activities of a user-account (or simply, a user) typically forms a time series with zero values indicating no action, and occasional spikes representing number of actions in that specific second. Throughout the paper, we assume a one-second sampling rate, though the method does not have any specific sparsity or sampling rate requirements. We define the problem we solve as follows.

Problem: *Find warped-correlated groups of users from activity signals at every T hours.*

The core of the above problem is comparing pairs of users to determine correlated groups, which is an unsupervised, pair-wise (i.e. quadratic) matching process. We first define terms and functions that provide necessary background for the algorithm.

Correlation: To capture users with highly synchronous posting activities, the correlation coefficient between two activity time series is a strong indicator. There are several measures of correlation. The most commonly used coefficient is Pearson's coefficient. For a time series x and y of length m, Pearson's correlation coefficient can be defined using the Euclidean distance between z-normalized time series $\hat{x}$ and $\hat{y}$ [13].

$$C(x,y) = 1 - \frac{d^2(\hat{x},\hat{y})}{2m} = \frac{\sum xy - m\mu_x\mu_y}{m\sigma_x\sigma_y}$$

Cross-correlation: Cross-correlation between two signals produces the correlation coefficients at all possible lags. For two signals x and y of length m, cross-correlation takes $O(m \log m)$ time to compute $2m-1$ coefficients at all lags. For $\tau \in [-m, m]$, a discrete version of cross-correlation ρ_{xy} is defined for integer lag τ as follows,

$$\rho_{xy}(\tau) = \begin{cases} C(x_{1:m-\tau}, y_{\tau+1:m}) & ,\tau \geq 0 \\ C(x_{|\tau|+1:m}, y_{1:m-|\tau|}) & ,\tau < 0 \end{cases}$$

Here the : operator is used to represent an increment-by-one sequence. Note that $\rho_{xy}(\tau) = \rho_{yx}(-\tau)$.

Typically, for large lag (τ), cross-correlation is meaningless for lack of data. In reality, every domain has a range of interesting lags.

Dynamic Time Warping: Correlation can capture high synchronicity among users, however, real bots often show warping. Dynamic time warping (DTW) distance is a well-studied distance measure for time series data mining. DTW allows signals to *warp* against one another, and constrained DTW allows warping within a window of w samples. Similar to lag in cross-correlation, the constraint window (w) for bot detection should not be more than a few seconds.

Warped Correlation: We extend the notion of warping to correlation. If x and y are *z-normalized*, DTW distance can be converted to a warped correlation measure with a range of [-1,1]. If the number of squared errors that are added to obtain a distance (i.e. the path length), is P then the warped correlation is defined as below.

$$wC(x,y) = 1 - \frac{DTW^2(\hat{x},\hat{y})}{2P}$$

The finite range of warped correlation is useful in measuring the significance of a match. A very strong warped correlation of 0.995 indicates almost identical behavior from two users. In this paper, we use a threshold of 0.995 warped correlation to identify bots.

Random Projection: Random projection has been used in high dimensional K-nearest neighbor search for over a decade [3]. Random projection has also been shown to work for time series similarity search in real time [6]. The key idea is to project each high dimensional time series on k random directions. By Johnson-Lindenstrauss lemma, it is probabilistically guaranteed that the points that are similar in the high dimensional space will be close/similar in the projected space, *and* that dissimilar points will be far/dissimilar [3]. In this project, we use cross-correlation-based random projection. Simply put, we generate one random vector and rotate the dimensions in both clockwise and anti-clockwise manners to produce the remaining random vectors.

III. DeBot Correlation Finder

A. DeBot Architecture

In this section, we describe the architecture of the DeBot system which detects bots every T hours. The system consists of four components which are shown in Figure 2.

The four components of the process are: *collector, indexer, listener, and validator*. The **collector** collects tweets that match with a certain set of *keywords* for T hours using the `filter` method in the API. The matching process in the Twitter API is quoted from the Twitter's developer's guide for clarity. "*The text of the Tweet and some entity fields are considered for matches. Specifically, the text attribute of the Tweet, expanded_url and display_url for links and media, text for hashtags, and screen_name for user mentions are checked for matches.*" The collector forms the time series of the number of activities at every second for all of the user-accounts. The collector filters out users with just one activity, as correlating one activity is meaningless. The collector then passes the time series to the indexer.

Note that, as we are using the `filter` method, we may not receive all the activities of a given user in the T hour period. This clearly challenges the efficacy of our method, as subsampled time series may add false negatives. Even though

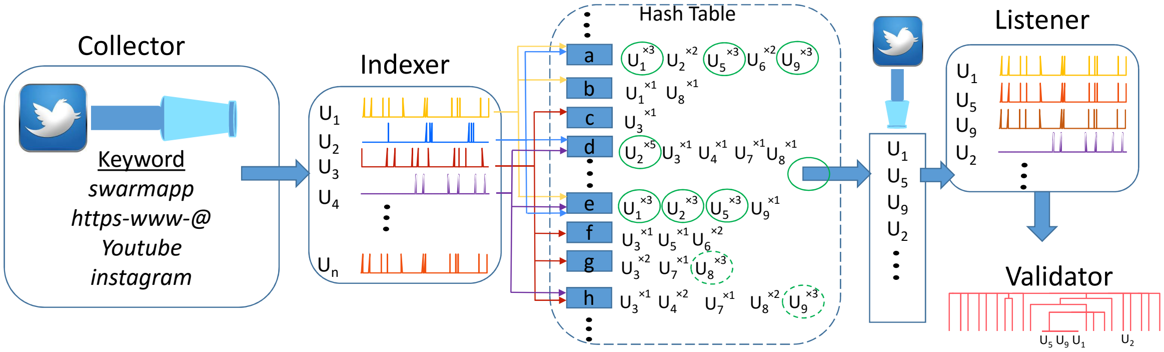

Collector
Keyword
swarmapp
https-www-@
Youtube
instagram
Indexer
U_1
U_2
U_3
U_4
U_n
Hash Table
a $U_1^{\times 3}$ $U_2^{\times 2}$ $U_5^{\times 3}$ $U_6^{\times 2}$ $U_9^{\times 3}$
b $U_1^{\times 1}$ $U_8^{\times 1}$
c $U_3^{\times 1}$
d $U_2^{\times 5}$ $U_3^{\times 1}$ $U_4^{\times 1}$ $U_7^{\times 1}$ $U_8^{\times 1}$
e $U_1^{\times 3}$ $U_2^{\times 3}$ $U_5^{\times 3}$ $U_9^{\times 1}$
f $U_3^{\times 1}$ $U_5^{\times 1}$ $U_6^{\times 2}$
g $U_3^{\times 2}$ $U_7^{\times 1}$ $U_8^{\times 3}$
h $U_3^{\times 1}$ $U_4^{\times 2}$ $U_7^{\times 1}$ $U_8^{\times 2}$ $U_9^{\times 3}$
U_1
U_5
U_9
U_2
Listener
U_1
U_5
U_9
U_2
Validator
U_5 U_9 U_1
U_2

Fig. 2. Four phases of our bot detection process. The system takes a stream of activities (e.g. Twitter Firehose) as input and produces groups of correlated users in a pipelined manner. *Collision scenario*: Assume $w = 12$, then each user is hashed into buckets (a to h) 25 times. The number of occurrences of a user is denoted by the superscript. We need $\lfloor \frac{w}{4} \rfloor$=3 occurrences of a user in the same bucket to qualify, e.g. U_2 is a qualified user in bucket d. Qualified users are marked with green ellipses. However, bucket d is not a qualified bucket as it does not have three qualified users. Buckets a and e are qualified. Thus from the hash table, we extract suspicious users: U_1, U_5, U_9, and U_2 which are circled with solid line.

we may have false negatives, our method outperforms existing bot detection techniques by far (see Section IV). Moreover, this issue disappears when site-owners use our method on the complete set of user activities.

The **indexer** takes the activity time series of all the users as input, hashes each of them into multiple hash buckets, and reports sets of suspicious users that collide in the same hash buckets. In order to calculate the hash buckets for a given set of time series, the indexer uses a pre-generated random time series r, calculates the cross-correlation between each time series, and r, and finally calculates $2w + 1$ hash indexes for different lags. Here, w is a user-given parameter representing the maximum allowable lag.Once hashed, the indexer finds a list of *suspicious users* which are qualified users in qualified buckets. *Qualified users* are those who have more than $\lfloor \frac{w}{4} \rfloor$ occurrences in a specific bucket. Similarly, *qualified buckets* have more than $\lfloor \frac{w}{4} \rfloor$ qualified users. We go through each qualified bucket and pick qualified users in them to report as suspicious users. The minimum number of occurrences ($\lfloor \frac{w}{4} \rfloor$) to qualify is made dependent on w to avoid an explicit parameter setting. We test the sensitivity of the parameter w in the experiment section.

The **listener** listens to the suspicious users exclusively. In this step, instead of using keywords, the Twitter stream is filtered on suspicious user accounts. The listener is different from the collector in a principled way. The listener receives all the activities of a suspicious user over a period of T hours, while the collector obtains only a sample of the activities. The listener will form the activity time series of the suspicious users and send them to the validator. The listener filters out users with less than *forty activities* as discussed in [4].

The **validator** reads the suspicious time series from the listener and verifies their validity. The validator calculates a pairwise warped correlation matrix over the set of users and clusters the users hierarchically up to a very restrictive distance cutoff. A sample of hierarchical clusters is shown in Figure 2. After clustering, every singleton user is ignored as a false positive, and the tightly connected clusters are reported as bots. For clarity we describe the clustering process separately in the next subsection.

B. Lag-sensitive Hashing

In this section, we describe our novel lag-sensitive hashing technique. For this techinque, we adopt the concept of structured random projection. We project activity signals in $2w+1$ directions, which are lagged vectors of a random vector r. We achieve this by simply calculating the cross-correlation between r and a given signal and picking the $2w + 1$ values around the symmetry. The following theorem describes the best-case hashing scenario.

Theorem 1. *If two infinitely long time series x and y are exactly correlated at a lag $l \leq w$ then they must collide in exactly $2w - l$ buckets.*

Proof: Let us assume r is the reference object of the same length as of x and y. Without losing generality, let us assume $\rho_{xy}(l) = 1.0$ and $l \geq 0$ (if $l < 0$, we can swap x and y). Every alignment of r with x has a corresponding alignment of r with y at lag l. Both of these alignments produce the same correlation and result into a collision in the hash structure. Formally, $\rho_{xr}(i) = \rho_{yr}(i - l)$ for any $i \in [-w, w]$. There are exactly three ways that this can happen.

- If $i < 0$, $\rho_{xr}(-i) = \rho_{rx}(i)$ and $\rho_{yr}(-i - l) = \rho_{ry}(i + l)$ are equal because r_i is aligned with x_i and y_{i+l}.
- If $0 < i < l$, $\rho_{xr}(i)$ and $\rho_{yr}(i - l) = \rho_{ry}(l - i)$ are equal because r_i is aligned with x_1 and y_l.
- If $i > l$, $\rho_{xr}(i) = \rho_{yr}(i - l)$ is trivially true because r_i is aligned with x_1 and y_l.

For $i < -(w - l)$, $\rho_{yr}(i - l)$ is not calculated by our hash function. Therefore, the only valid range for i is $[-(w-l), w]$, which gives us $2w - l$ collisions. ■

How well do cross-correlation coefficients capture DTW distance or warped correlation? We calculate the DTW distances and cross-correlation between 5000 pairs of random walks. We use the same w as the constraint window size and the maximum allowable lag. We plot the maximum cross-correlation (for lags in $[-w, w]$) against the DTW distance in Figure 3(left), which shows the reciprocal relationship that we exploit in our lag-sensitive hashing scheme.

We analyze the goodness of lagged projection compared to classic random projection when hashing signals. There are two

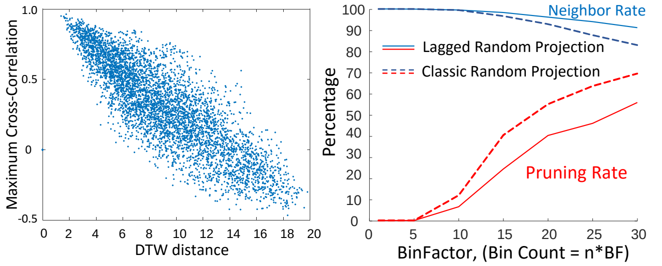


Fig. 3. (left) Maximum cross-correlation shows reciprocity with DTW distance.(right) Comparison with classic random projection based on pruning and neighbor rates.

important metrics that we consider: The *pruning rate* is the percentage of similarity comparisons that we can avoid while finding any of the top-5 nearest neighbors without the hash structure. The *neighbor rate* is the percentage of times we can retrieve any of the top-5 nearest neighbors under warped correlation using the hash structure. Ideally we want both the metrics to be close to 100%.

We evaluate the neighbor rate and pruning rate on several datasets from the UCR archive [11] for various bin counts. We show a representative chart for the Trace dataset in Figure 3(right). This chart identifies the trade-off between the two techniques. Classic random projection is better in pruning (i.e. speed) while our proposed lagged projection is better in finding the neighbors (i.e. accuracy). This is an understandable trade-off between structured and true random projections. Since we would like to find highly correlated groups, accurate lagged projection is our method of choice.

C. Clustering

The validator calculates the pairwise constrained warped correlation for all of the suspicious users. We use the maximum allowable lag (i.e. w) from the indexer as the constraint size.

The validator then performs a hierarchical clustering on the pairwise DTW distances using the "single" linkage technique, which merges the closest pairs of clusters iteratively. A sample dendrogram is shown in Figure 4, which shows the strong clusters and the numerous false positives that we extract from the time series.

We use a very strict cutoff threshold to extract highly dense clusters and ignore all the remaining singleton users. For example, in Figure 2, U_1, U_5 and U_9 are clustered together and U_2 is left out as false positive. The cutoff we use is 0.995 warped correlation. The extracted clusters, therefore, contain significant bot accounts.

As we pass more rounds of T hours, we can merge these clusters to form bigger clusters. This is an important step, because bots form correlated groups and may disband them dynamically. Therefore, an already detected bot can reveal a new set of bots in the next round of T hours. While merging these clusters, we use a simple *friend-of-friend* technique. If two clusters share one user in common, we merge them. Although it may sound very simple, we see that such a simple method can retain high precision because of the overwhelming number of existing bots.

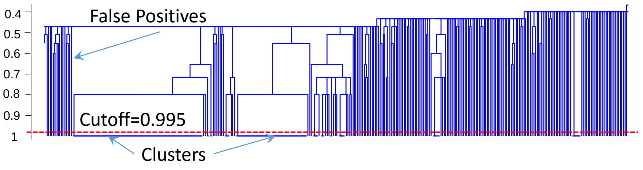


Fig. 4. A sample dendrogram of suspicious users' activities. Only a few users fall below the restricted cutoff. The rest of the users are cleared as false positives.

IV. Empirical Evaluation

All of the experiments in this section are exactly reproducible with code and data provided on the supporting page [2]. DeBot produces a daily report of the bot accounts by analyzing the activities of the previous day. The daily reports and detected bots are available at [2]. We have three inter-dependent parameters: the number of buckets (B=5000) in the hash table, the base window (T=2 hours), and the maximum lag (w=20 seconds). Unless otherwise specified, the default parameters are used for all experiments.

A. Bot Quality: Precision

Our method produces a set of clusters of highly correlated users based solely on their temporal similarity. As mentioned earlier, we find correlated users (> 0.995) who have more than forty synchronous activities in T hours. In this subsection we empirically validate the quality of the bots that we detect using several methods.

1) Comparison with existing methods: Typically, there are three approaches to evaluating the detected bots. The first approach is to sample and evaluate the accounts manually [9]. The second approach is to set up "honeypot" in order to produce labeled data by attracting bots, and then to evaluate a method by cross validation [15]. The last approach is to check whether or not the accounts are suspended by Twitter at a later time [16]. The first two approaches are suitable for supervised methods and only produce static measurements at one instance of time. Our major evaluation is done against Twitter over three months. We also compare DeBot with two other static techniques from the literature.

Comparison with Twitter and Bot or Not?: Twitter suspends accounts that do not follow its rules [18]. To compare the results of DeBot with Twitter's suspension process, we design two segments: static and dynamic. In the static segment, we ran DeBot every 4 hours for sixteen days (May 18 - June 3, 2015) and linked all the clusters into one integrated set of clusters using the *friend-of-friend* technique. We picked the top ten clusters (9,134 bot accounts) to form our **base set**. From June to August 2015, we probed the Twitter API every few days to check if these detected accounts were suspended. As you can see in Figure 5 (left), the number of bots suspended by Twitter increased over time, and by the end of 12 weeks **45%** of the bot accounts identified by DeBot were suspended.

In addition to this static segment, where we kept the set of bots detected by DeBot fixed and probed Twitter over time, we also performed dynamic detection. On August 28, 2015, we started running DeBot every week and added the newly

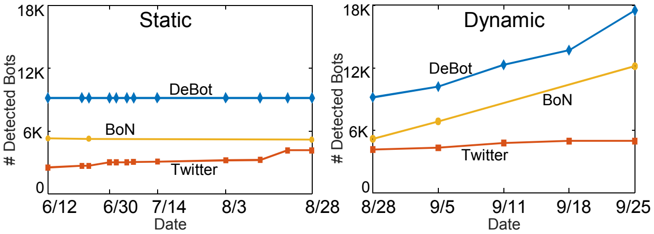


Fig. 5. Comparison between the number of bots detected by DeBot, Twitter, and *Bot or Not?* project over time. (Note that we probed Twitter and *Bot or Not?* only for the accounts in the base set.)

discovered bots to the *base set* of bots. In every run, we listened to Twitter for 7 successive days. The results are shown in Figure 5 (right). DeBot consistently found new bots every week. We then continuously probed Twitter to check the status of the newly detected bots. The results clearly show that the number of bots we detect is increasing at a higher rate than Twitter's suspension rate. It is very likely that Twitter detects more bots than they suspend, and that several relevant factors such as country specific laws, need for graceful action etc. may contribute to Twitter's suspension system. The outcome of our experiments is a reminder that Twitter may need to be more aggressive in their suspension process. At the time this is written, DeBot has accumulated a set of close to 500,000 bots (at a rate of close to 1500 bots per day!). The identities of these bots are available in our supporting page [2].

An obvious question one might ask is: *how many bots that are not suspended by Twitter are worth detecting?* We answer this question by comparing our method with a successful existing technique developed in the Truthy project [8], *Bot or Not?*. *BotOrNot* is a supervised method that estimates the probability of *"being bot"* for a given account. Having 50% or more as the threshold, we got **59%** relative support from *Bot or Not?* in June 2015. We probed *Bot or Not?* two more times in the static segment (see Figure 5) and notice no significant change in detection performance. We also probed *Bot or Not?* twice in the dynamic segment and observe that *Bot or Not?* detected increasingly more bots as DeBot was growing the base set. This supports our original argument that Twitter is falling behind in detecting bots.

The reason why *Bot or Not?* is only half as accurate as DeBot is that the method was trained for English-language tweets, while DeBot catches all languages just based on temporal synchronicity. Another reason is that *Bot or Not?* is a supervised technique trained periodically. In contrast, DeBot detects bots every day in a completely unsupervised manner. *Bot or Not?* probably misses some recent dynamics of bots, resulting in a smaller overlap.

A complementary question is: *which method (DeBot or Bot or Not?) produces bots that Twitter preferentially suspends?* We calculate the fraction of accounts that Twitter suspends for *Bot or Not?* and DeBot exclusively. We see that Twitter suspends more bots that are supported by *Bot or Not?* (37.43%) than are supported by DeBot (21.06%). This bias to a feature-based supervised method possibly indicates that temporal synchronicity should be used by Twitter's detection and suspension mechanism.

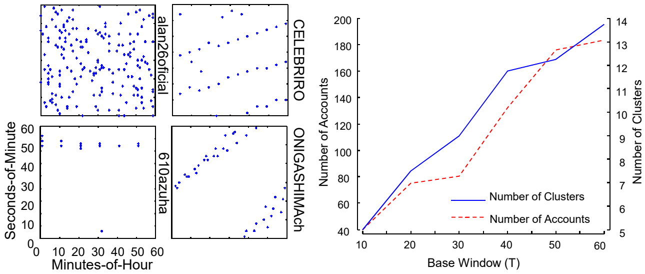


Fig. 6. (left) Four bots showing different patterns in the minutes-of-hour vs. seconds-of-minute plot. (right) Effect of base window on the detection performance

.

Comparison with per-user method: Per-user methods are being developed actively by researchers. We compare DeBot to a per-user method in [21], which tests the independence of minute-of-an-hour and second-of-a-minutes with χ^2 test. Figure 6 (left) shows a set of bots and their second-of-minute vs. minute-of-hour plots. The test cannot detect bot accounts with uniformly distributed activities. **76%** of the detected bots by DeBot are supported by the χ^2 test on average. There are other per-user methods [5][15][7] that use machine-learned classifiers to detect bots. The method in [7] is similar to ours in considering temporal behavior. However, the method is a supervised per-user method trained on a small dataset of around a few thousand accounts. We do not compare DeBot with this method since DeBot is unsupervised, works in real time, and identifies several hundred bots every day.

2) Contextual Validation: One-quarter of the bots detected by DeBot are not yet supported by Twitter, or *Bot or Not?*, or the χ^2 test. Are they worth finding? An exact answer to this question really does not exist, due to the lack of ground truth. To alleviate this concern, we evaluate the bots using contextual information, such as tweet content, and get an average of **78.5%** relative support. We also employ human judges to compare the content of our bots against each other and achieve **94%** support. You can find the details of these experiments and recall estimation in [4].

B. Parameter Sensitivity

We have three inter-dependent parameters that we analyze in this section. We iterate over each parameter, while keeping the remaining parameters fixed. For the experiments in this section, we use the keywords (`swarmapp` | `youtube` | `instagram`) as our filter strings.

Base Window (T)**:** We change the size of the base window, T, to observe the change in detection performance. We see consistent growth in number of clusters and bot accounts as T increases. A larger base window ensures that more correlated users can show up and be hashed. The end effect is that we have higher quality clusters at the cost of a longer wait. Figure 7 (left) shows the results.

Number of Buckets (B)**:** We change the number of buckets in the hash structure. Too few buckets will induce unnecessary collisions, while too many buckets spread users sparsely. Figure 7 shows that the maximum number of clusters and bot accounts can be achieved by using 2000 to 4000 buckets.

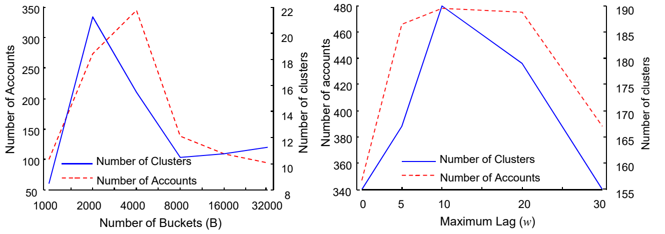


Fig. 7. Effect of parameters on the detection performance, (left) number of buckets and (right) maximum lag in seconds.

Maximum Lag (w): We check the impact of maximum lag over detection performance. As previously described, user activities require lag and warping sensitive correlation measures. For zero lag (essentially Euclidean distance), we obtain significantly fewer clusters and bot accounts. For the lag of 30 seconds, the number of clusters is again low because the hash structure is crowded with copies of each user, resulting in lots of spurious collisions. Results are shown in Figure 7.

C. *Scalability*

Online methods depend on several degrees of freedom. This makes analyzing and comparing scalability difficult. Two quantities are most important: *data rate* and *window size*. The Twitter streaming API has a hard limit on the data rate; we receive tweets at a 48 tweet-per-second rate at the most. Even if we increase the generality of the filter string, we cannot increase the data rate.

Therefore, scalability depends on much of a user's history we can store and analyze. This is exactly the parameter T in our problem definition. We set our largest experiment to collect 1 million user accounts. This is a massive number of time series to calculate the warping-invariant correlation for all pairs. Note that it is easier to do trillions of subsequence matching [14] in a streaming fashion at a very high data rate by exploiting overlapping segments of successive subsequences. Calculating pairwise DTW distances for a million users is equivalent to a trillion distance calculation without overlapping substructures. We exploit the efficiency of cross-correlation, which enables our hashing mechanism to compute the clusters and identify bots.

It takes $T = 9.5$ hours to collect 1 million users. The indexer then takes 40 minutes to hash all the users. 24,000 users are qualified for the listener, and the validator detects 93 clusters of 1,485 accounts.

V. Conclusion

We illustrate that the presence of highly synchronous cross-user activities reveals abnormalities and is a key to detecting automated accounts. We develop an unsupervised method which calculates cross-user activity correlations to detect bot accounts in Twitter. We evaluate our method with per-user method and Twitter suspension process. The evaluation shows that Twitter suspends automated accounts with lower rate than our method finds them. DeBot also detects more bots when compared to per-user methods. DeBot is running and detecting thousands of bot daily. Our future goal is to extend this work to further understand bot behavior in social media to improve trustworthiness and reliability of online data.

References

[1] How twitter bots fool you into thinking they are real people. http://www.fastcompany.com/3031500/how-twitter-bots-fool-you-into-thinking//-they-are-real-people.

[2] Supporting web page containing video, data, code and daily report. www.cs.unm.edu/~chavoshi/debot.

[3] E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. ACM SIGKDD 2001.

[4] N. Chavoshi, H. Hamooni, and A. Mueen. Identifying Correlated Bots In Twitter. SocInfo 2016.

[5] Z. Chu, S. Gianvecchio, H. Wang, and S. Jajodia. Detecting Automation of Twitter Accounts: Are You a Human, Bot, or Cyborg? *IEEE Transactions on Dependable and Secure Computing*, 9(6):811–824, Nov. 2012.

[6] R. Cole, D. Shasha, and X. Zhao. Fast window correlations over uncooperative time series. ACM SIGKDD 2005.

[7] A. F. Costa, Y. Yamaguchi, A. J. M. Traina, C. T. Jr., and C. Faloutsos. RSC: mining and modeling temporal activity in social media. ACM SIGKDD 2015.

[8] C. A. Davis, O. Varol, E. Ferrara, A. Flammini, and F. Menczer. Botornot: A system to evaluate social bots. In *Proceedings of the 25th International Conference Companion on World Wide Web*, 2016.

[9] J. Jiang, C. Wilson, X. Wang, P. Huang, W. Sha, Y. Dai, and B. Y. Zhao. Understanding latent interactions in online social networks. IMC 2010.

[10] E. Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB 2002.

[11] E. Keogh, X. Xi, L. Wei, C. A. Ratanamahatana, T. Folias, Q. Zhu, B. Hu, and H. Y. The UCR time series classification/clustering homepage, 2011.

[12] H. Li, A. Mukherjee, B. Liu, R. Kornfield, and S. Emery. Detecting Campaign Promoters on Twitter Using Markov Random Fields. ICDM 2014.

[13] A. Mueen and E. Keogh. Online discovery and maintenance of time series motifs. ACM SIGKDD 2010.

[14] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. ACM SIGKDD 2012.

[15] G. Stringhini. *Stepping Up the Cybersecurity Game: Protecting Online Services from Malicious Activity*. Thesis, UNIVERSITY OF CALIFORNIA Santa Barbara, 2014.

[16] K. Thomas, C. Grier, D. Song, and V. Paxson. Suspended accounts in retrospect: an analysis of twitter spam. IMC 2011.

[17] K. Thomas, V. Paxson, D. Mccoy, and C. Grier. Trafficking Fraudulent Accounts : The Role of the Underground Market in Twitter Spam and Abuse Trafficking Fraudulent Accounts. In *USENIX Security Symposium*, 2013.

[18] Twitter. About suspended accounts. https://support.twitter.com/articles/15790.

[19] Twitter. The Twitter Rules. https://support.twitter.com/articles/18311.

[20] A. Wang. Detecting Spam Bots in Online Social Networking Sites: A Machine Learning Approach. In *Data and Applications Security and Privacy XXIV*, volume 6166 of *Lecture Notes in Computer Science*, pages 335–342. Springer Berlin Heidelberg, 2010.

[21] C. M. Zhang and V. Paxson. Detecting and analyzing automated activity on twitter. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bio informatics)*, volume 6579 LNCS of *PAM*, 2011.