Towards Automated Bot Detection in Political Social Networks

Berat Bicer
berat.bicer@bilkent.edu.tr
Bilkent University
Ankara, Turkey

Vahid Namakshenas
vahid@bilkent.edu.tr
Bilkent University
Ankara, Turkey

ABSTRACT

In this work, we study the effectiveness of some of the state of the art bot detection strategies from the literature that can be executed in parallel on recently proposed TwiBot-20 dataset and analyse how network science can contribute to improving the efficiency of bot detection algorithms. We report a contrastive performance overview of studied methods, and conclude with an analysis into obtained results regarding method efficiency and stability.

CCS CONCEPTS

• **Bot Detection**; • **Network Science**; • **Natural Language Processing**;

KEYWORDS

bot detection, network science, natural language processing

ACM Reference Format:
Berat Bicer and Vahid Namakshenas. 2022. Towards Automated Bot Detection in Political Social Networks. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Social media platforms, since their inception, have grown significantly in terms of both their popularity and wide spread usage, and their ability to impact the socio-economic, political, and cultural discourse of countries around the globe. Such an influence undoubtedly attracts agents with nefarious intent, ranging from individuals to political activists and state actors, who seek to obtain a platform to further their political agendas, which necessitates a unified effort to combat such actors seeking to alter the public's perception through various means.

This project seeks to study how the political discourse on Twitter is influenced by or is otherwise subjected to bot activity and whether it's possible to identify them through computational methods. Specifically, we're interested in a small political network based on the broad bot-detection corpus called TwiBot-20. Our network is focuses on prominent figures in contemporary figures in U.S. politics and their connections. Based on this dataset, we propose multiple bot detection strategies that can be operated independently, raging from graph and node-level analysis based on centrality metrics and community detection, to node-based context encodings and natural language processing for bot detection. Our goal is to offer an overview and test the effectiveness of influential strategies from the literature on TwiBot-20 dataset, while studying how network science can contribute to bot detection in hopes of contributing to the fight against misinformation.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 LITERATURE REVIEW

The earlier proposals for Twitter bot detection concentrate on feature engineering with user information [4]. Lee et al. proposed a suspicious URL detection system for Twitter which focuses on the correlations of multiple redirect chains that share the same redirection servers [10]. Thomas et al. presented a real-time system that crawls URLs as they are submitted to web services and determines whether the URLs are direct to spam also they explored the distinctions between email and Twitter spam, including the overlap of spam features, the persistence of features over time, and the abuse of generic redirectors and public web hosting [17]. Gao et al. provided online spam filtering for social networks by using text shingling and URL comparison to incrementally reconstruct spam messages into campaigns, which are then identified by a trained classifier. Yang et al. designed new and robust features to detect Twitter spammers based on an in-depth analysis of the evasion tactics utilized by Twitter spammers [18]. Other features are also exploited, such as information that is related to the user homepage [9], social networks [15], and timeline of accounts [2].

With the emergence of deep learning, neural networks are increasingly used for Twitter bot detection. Stanton et al. proposed spamGAN, a generative adversarial network that relies on a limited set of labeled data as well as unlabeled data for opinion spam detection [16]. Wei et al. presented an RNN model with word embedding to distinguish Twitter bots from human accounts. Kudugunta et al. proposed a deep neural network based on contextual long short-term memory (LSTM) architecture that exploits both content and metadata to detect bots at the tweet level [7]. Alhosseini et al. proposed a model based on graph convolutional neural networks (GCNN) for spam bot detection regarding its ability to leverage both the features of a node and aggregate the features of a node's neighborhood [1].

3 METHODOLOGY

In this section, we start by specifying distinctions between bots and real accounts, bot in terms of the characteristics of their neighbourhoods and the communities they tend to form; both of which offers clues in regards to how bot detection can be handled. Later, we describe the methodologies used in the rest of the paper starting with graph and node-level analysis, followed by community detection strategies, mapping neighbourhood characteristics as mentioned

before, and then concluding with natural language processing applications.

3.1 Definitions

In this study, our goal is to separate bot accounts from accounts affiliated with real people or organizations. In this sense, a bot account is an account that is automated or has limited human intervention in its operations. A real account, on the other hand, is controlled by people directly. Such accounts' actions are directly undertaken by human operators whereas bot operations can be handled algorithmically.

In a political network, bots and real accounts have unique goals and principals that they operate on. For example, some of the probable goals of a bot account are reporting recent developments regarding public figures and elected officials, as well as their statements and political actions; spreading propaganda and ads in hopes of galvanizing a follower base for a certain political cause, etc. For this reason, automated accounts seek to expand their influence over other real accounts by following as many real accounts as possible, which in turn yields new followers for them. On the other hand, real accounts while they may share similar goals with bots, real accounts interact with real accounts more often than they do with bots; thus, their outgoing connections usually are other real-accounts while incoming connections are a mix of bots and real accounts. For this reason, real accounts generally have large betweenness and operate as hubs.

In some cases, a tightly-connected group of bots form what can be described as botfarms. These groups generally seek to boost the popularity of a certain post or attempt to spread an idea to deceive other real accounts into believing the idea is genuine and is widespread. Most of the time, hierarchical clustering and timestamp-based analysis on site-wide interactions are strong tools for detecting these botfarms. This is so because there are two main structure botfarms form: One resembles the small world phenomena and creates densely packed groups of bots. The other creates chains of connected bots forming a hierarchical network through which information passes down.

3.2 Graph and Node-Level Analysis

In this section, we share some probable strategies for bot detection based on graph-level and node-level metrics discussed during the semester. These candidates include density and node degree, average geodesic distance for a node, eigenvector centrality, betweenness, and lastly PageRank values.

As the density of a network refers to the number of dyadic linkages between its nodes, a low density manifests a sparse graph. Comparing the average density and density of each node can give us some ideas about determining whether the account associated with a node is a bot. For example, if a node has a high density compared to the average, it can be deduced that this node clamors for attention. In this case, we encounter three different modes. In the first mode, if the node has a high number of outgoing edges (followings) and only a mere number of incoming edges (followers), that node might be considered a bot. In the second mode, it is possible to have a high number of followers and a low number of followings. This situation can be a characteristic of a critical node: A node that is a touchstone and can be a basis for spreading information (such as certain politicians). In the third mode, we have a node that has a high number of followers and followings. In this case, we have to investigate reciprocity. If the connections are mostly symmetric, it can be concluded that the node belongs to a human; otherwise, the node can be considered a bot account. We assume that if an account belongs to humans, the followers and followings should be mostly the same. However, this deduction is not robust and needs to be scrutinized by other measurements. The density formula for our directed graph is as follows:

$$D = \frac{L}{N(N-1)}$$

where L is the number of edges and N is the number of nodes in the graph.

Another important metric is the characteristic path length, or more specifically, average geodesic distance for each node. Seemingly our network does not meet the "Six degrees of separation" theorem since the characteristic path length is roughly long. While it may be possible that, in a network containing many bot accounts, characteristic path length may be quite large, we do not have baselines for reliable comparisons. Thus, we instead opt-out for average geodesic distance instead; We observe a correlation between the average geodesic distance for a given node and the likelihood that the associated account is labeled as a bot based on the property that real accounts tend to be hubs or operate in a similar fashion. For computing the geodesic distance, the formula may be utilized where $d(i, j)$ is the geodesic distance from node i to node j:

$$d_i = \frac{\sum_{j=1}^{n} d(i, j)}{(n-1)}$$

Considering the definition in the previous section, a bot farm consists of many bots connected and can influence real accounts by over-advertising. One exciting metric which corresponds to this issue is eigenvector centrality. As per definition, an influential node that is highly connected to another strongly connected nodes is considered to have a high eigenvector centrality. In our case study, we can detect the important bot nodes that are connected to bot farms. Although not all dense clusters are bot farms, we can locate candidates for further investigations.

Betweenness centrality identifies the nodes that can function as brokers or gatekeepers. Hence, from a semantics perspective, the nodes connecting two groups to each other have to be meaningful. They have to share some common aspects with both groups. Hence, in our case, the bot nodes should be the nodes with a low betweenness centrality value. A bot account cannot be a broker between two groups of real accounts. However, in this case, we assume that both clusters contain real accounts. The formula for the betweenness centrality is:

$$b(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

$b(v)$ is the betweenness of node v, $\sigma_{st}(v)$ is the number of shortest paths between s and t that pass through v, and σ_{st} is the number of shortest paths between s and t.

According to PageRank's definition, the Nodes are important that have a high number of incoming edges. Therefore, one possible

methodology will be calculating the PageRank metric to describe a node's importance. Obviously, a node having a high PageRank value is a reference node. Such a node has the least possibility of being a bot account.

3.3 Community Detection For Bot Inference

Cluster Analysis or clustering, also referred to as community detection, is a process that can be used to separate a set of objects into distinct groups called clusters such that members of each cluster display similar properties amongst themselves. For this reason, The members of each cluster show strong similarities to one another compared to the the degree of similarity between members of distinct clusters. The purpose of clustering is, therefore, to assign labels to objects that indicate the membership of each object to their respective cluster. One key distinction between clustering and classification is that during clustering, we assume datasamples lack their primary labels, thus, cluster assignments are performed without this knowledge:; contrary to classification. As a result, clustering is considered an unsupervised learning strategy whereas classification is referred to as a supervised learning stretegy.

While there are many different clustering algorithms, these may be categorized broadly as hierarchical clustering and expectation-maximization based algorithms. In hierarchical clustering such as DBSCAN[3], nodes and clusters are iteratively merged to form larger clusters. Expectation-maximization based approaches such as k-nearest neighbours (k-nn) algorithm [12], on the other hand, attempt to solve the problem of partitioning the samples into m distinct cluster such that within-cluster variances are minimized, which is in fact a NP-Hard problem. Thus, such algorithm offer heuristic solutions that converge to local maxima rather than an optimal solution.

For the purpose of this study, we believe attempting to construct few high-level clusters may not be informative for bot detection since we expect the inter-cluster variance will be quite high. In other words, using k-nn to construct two clusters, one for bots and one for real accounts, likely will fail to construct pure clusters. Thus, we instead propose employing hierarchical clustering to detect many unique communities that share similar properties within, which can also help locating bot farms discussed earlier. These farms are mainly described as a chain or cluster of connected nodes attached to a few hub-like anchor points. Since hierarchical clustering starts at the leaf nodes, in a few iterations the entire structure can be grouped together and studied later for verification.

3.4 Neighbour Embedding For Bot Detection

As mentioned in Section 3.1, characteristics of the connections of a certain node is a rich source of information when inferring bots. Based this, we propose utilizing the neighbours of a node as follows: Let S be a sample in the dataset and N_S be the set of vertices that are the immediate neighbours of the node S. Assuming that label $y_S = 1$ if S is a bot and $y_S = 0$ otherwise, we can construct the following binary string NE_S where the length of this string is equal to the cardinality of the neighbour set, i.e., $|NE_S| = ||N_S||$ and

$$NE_S[i] = \begin{cases} 1 & if\ y_{N_S,i} = 1 \\ 0 & if\ \text{otherwise} \end{cases} \quad (1)$$

where $NE_S[i]$ is the ith character of the string and $y_{N_S,i}$ is the label of ith neighbour of S. Afterwards, we can convert NE_S into a base-10 value, and apply thresholding over the dataset for classification. In this case, let BS be the base-10 value of NE_S. Then, for a threshold δ,

$$y_{BS} = \begin{cases} 1 & if\ BS > \delta \\ 0 & if\ \text{otherwise} \end{cases} \quad (2)$$

Note that the optimal value of δ is determined through experiments. Alternatively, the number of 1s or their frequency in NE_S can be used to determine a threshold. Lastly, instead of utilizing BS, we can directly employ NE_S as a feature vector for S and train a classifier over the dataset. In this case, representations need to be padded to match the node having the highest degree. Afterwards, any classifier can be fitted to the input dataset for predictions.

3.5 Language-Based Bot Detection

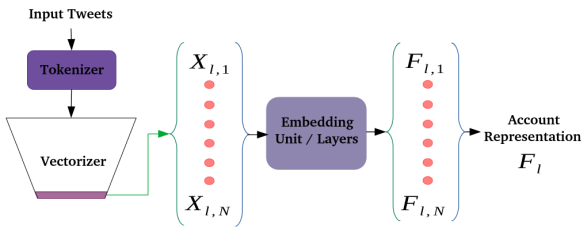


Figure 1: Visualization of the preliminary design of the text model used for tweet-based bot detection.

Natural language processing (NLP) is a fundamental part of our analysis. Our method seeks to convert a sequence of tweets into a 1-dimensional tensor representation for inferring whether an account is a bot based on the following pipeline.

Firstly, just like many NLP applications, we first filter out the dataset to eliminate potentially problematic language elements mentioned in Section 4. These include but not limited to non-unicode characters and emojis, special characters; Twitter tags, HTML character codes; mentions of and links to other accounts and tweets, etc. As described previously, regular expression matching is heavily utilized for this purpose. Afterwards, we remove the stop words from the filtered tweets and apply lemmatization to simplify the tokens, matching words having similar origins and functions together to reduce inter-tweet variations and variations caused by synonyms. This entire process can be roughly described as preprocessing.

Second, after applying preprocessing we tokenize the tweets for detecting language elements based on semantics and grammatical rules, and convert the detected tokens into vectors by a vectorization algorithm for further processing. At this stage, there are many alternatives we can choose from. In particular, *word2vec* [13, 14] and *doc2vec* [8] are popular choices for this purpose. However, recently transformer-based alternatives such as RoBERTa [11] have been studied extensively in the literature. While transformers are the current state of the art in many language tasks, they are hard

to train and require strong hardware support which we lack. Therefore, traditional approaches such as TF-IDF vectorization can be employed as well.

After vectorization, we design an encoder that generates account embeddings for classification based on information content of the tweets of a certain account. For this purpose, we designed the pipeline shown in Figure 1. Here, we also summarize the previous steps as well: When a batch of tweets are observed for a particular account, they are first subjected to preprocessing and are vectorized by one of the options mentioned above. Then, the resulting embeddings are merged, usually by averaging across the tweet and then the account in sequence, to output a single account-level feature vector F_l. This vector is fed to the classifier consisting of fully-connected layers to predict whether an account is a bot. Alternatively, instead of averaging over the tokens and the tweets, we can employ an LSTM [6] unit, however, this introduces additional complexity to the design and require stronger hardware support than we currently have. Thus, investigating LSTM usage is left for future work.

4 DATASET & EXPERIMENTAL SETUP

TwiBot [4, 5] is a bot detection dataset collected from Twitter users. In [4], which is referred to as TwiBot-20, there are unique accounts including prominent figures from contemporary U.S. politics and their followers engaged in political discourse and their Twitter statistics. Each node has a certain number of neighbours that form follows/followed by relationships in form of outgoing and incoming edges. There are also 200 tweets associated with each node for text-based classification, as well as ground truth annotations to indicate whether a Twitter account is a bot. These labels form the basis of our validation strategy: predictions from the methodologies mentioned previously will be compared to these annotations for computing classification accuracy, as well as class-wise classification accuracy for bots and real accounts separately. In total, there are around 10000 accounts available for tweet-based classification; however, for graph-level analysis, when disconnected nodes are removed, we are left with approximately 2600 unique accounts involved in politics.

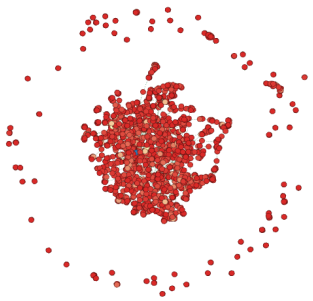


Figure 2: Layout of the vertices in TwiBot-20 under politics category visualized after removal of disconnected nodes.

While TwiBot-20 is useful for political bot detection, it has some shortcomings that we need to address. Firstly, node degrees in the dataset aren't uniform. There are disconnected regions that may, in truth, be connected to the rest of the graph which is caused by limited number of connections the dataset includes for each node. This requires a manual filtering of isolates by eliminating the nodes having zero total degree before the graph-based analysis can be made. Moreover, some accounts that do not participate in political discourse are also removed from the dataset.

Second, the dataset contains two broad, disconnected clusters of nodes: a disk-like region outside the center cluster containing disconnected communities and the center cluster itself which houses the bulk of the nodes and their connections (see Figure 2). This implies that any detected cluster larger than a certain size is unlikely to be informative for the context of bot detection. In other words, since high-level clusters are expected to be strongly intertwined based on visualizations obtained, bot detection based on these clusters would likely result in poor detection performance.

Thirdly, while there seem to be a sufficient amount of tweets for natural language processing-based (NLP) bot detection, many problems exist regarding these tweets. For example, there are many non-unicode characters such as emojis and Twitter tags; many informal language elements, slangs and abbreviations which need to be filtered out. Also, links to other tweets and websites, as well as HTML character codes can be found in some tweets. Lastly, due to the restrictions on their length, some tweets may be be too short after filtering, which can be problematic.

In terms of experimental setup, we have two main configurations. For NLP, the tweets are split into the train set and the test set by default. We further split the training set into 90-10 train-validation splits, and use the test set as is. Splits are based on the accounts, and not the tweets; which prevents information leak between data splits. For other methodologies, we split the available nodes into 80-10-10 train-validation-test splits based on accounts. Note that these configurations are fixed and are reused accross all experiments.

5 EXPERIMENTS & RESULTS

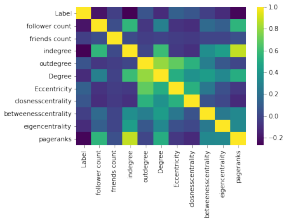
5.1 Graph and Node-Level Analysis

Table 1: Graph and Node-Level metrics of our data set

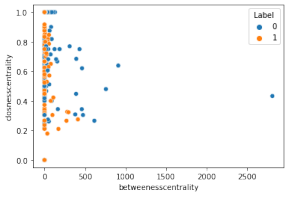
Metric	Value
# Nodes	2605
# Links	2070
Degree Distribution	.795
Density	.001
Clustering Coefficient	.005
Mean Path Length	2.794
Diameter	9

Table 1 manifests the calculated metrics for our data set. The metrics are calculated using GEPHI. As we can observe, the density is very low, so our network is sparse. Next, their diagrams are displayed to investigate the metrics and their effectiveness in the task.

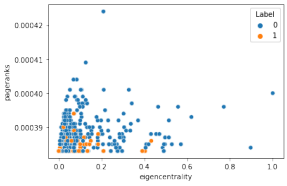
Firstly the data set is pre-processed to obtain cleaner data. Then in order to map the data set to our proposed network, we have demonstrated the centralities and other measurements. To compare



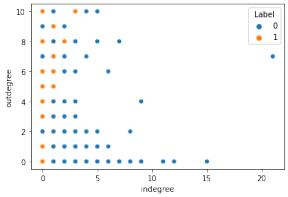
(a) Hitmap Correlations



(b) Closeness-Betweenness Centrality



(c) Eigenvalue Centrality-PageRank



(d) In-out degree measurements

Figure 3: Distribution of various node- and graph-level metrics over bots and real accounts. Best viewed zoomed.

them and find the correlation, the correlation hitmap is displayed in Figure 3 (a). As we can observe, some of the measurements have a roughly high correlation. For example, In-degree has a high correlation with PageRank, which is feasible. Figure 3 (b) shows that closeness centrality approximately has nothing to do with bot accounts; in contrast, there is an obvious relation between betweenness centrality and type of account. The more betweenness centrality, the less possible to be a bot account. Furthermore, as shown in Figure 3 (c) eigenvalue centrality and PageRank have straightforward relations with the bot accounts. The more eigen centrality and PageRank, the less possibility of being a bot. Figure 3 (d) shows that the accounts with high In-degree value are mostly real accounts and the Out-degree does not correlate with the class of the account.

5.2 Neighbour Embedding For Bot Detection

Table 2: Contrastive view on correct classification rate (CCR) computed over the test split. These results are obtained via experiments exploring neighbourhood information as feature representation.

Setup	Classifier	CCR
Neighbour BS	Threshold	*None*
Neighbour BS	Weighted Threshold	*None*
Neighbour Sum	Weighted Threshold	*None*
Neighbour Frequency	Weighted Threshold	0.474
Neighbour Embedding	MLP	*None*
Neighbour Embedding	SVM	*None*
Neighbour Embedding	**Random Forest**	**0.583**

As discussed in Section 3.4, we have experimented with various neighbour embedding strategies as shown in In Table 2. Here, BS uses the base-10 value of NE_S. *sum* indicates that values for thresholding are obtained by summing the labels of the neighbours. *Frequency* indicates the ratio of 1s over the neighbourhood size is used to embed the neighbourhood. In *neighbour embedding*, we employ NE_S as a feature vector directly. Lastly, *weighted threshold* indicates the selection is done based on a weighted average CCR over the validation set where coefficients are inversely proportional to class sizes. In our experiments we observed that since the real accounts vastly outnumber the bots, the models that fail to differentiate the samples show a strong bias towards this crowded category instead, which means no learning occurs. These cases are marked with *None* in the table. Interestingly, the only acceptable result came from a random forest classifier which is significantly better than random chance and shows that our initial assumptions regarding the neighbourhood characteristics were partially accurate, yet far from reliable.

5.3 Community Detection For Bot Inference

Table 3: Clustering results

Cluster ID	0	1	2	3	4	5	6	8	10
# bots	986	339	24	6	75	0	0	4	3212
# nodes	1744	492	37	256	105	38	14	5	5585
% bots	56	69	65	2.3	71	0	0	80	58

Table 3 shows the results of the clustering. For this purpose, we first pre-processed the data and removed the correlations. Then we applied hierarchical clustering to form our clusters. We performed the clustering with different settings and the number of clusters. We have ignored the low-density clusters to clearly show our results. Some of the small clusters are exempt from any bots. But, as a result, we can observe that most of the bots are in the most prominent clusters and have a dominant majority. In one tiny cluster, we have an approximately high number of bots, which can be considered a bot farm. The surprising part of the clustering is that most clusters with high nodes contain high numbers of the bot. It can be inferred that bots are spread broadly and randomly.

5.4 Language-Based Bot Detection

Table 4: Contrastive view on correct classification rate (CCR) computed over the test split. Results are obtained when specified algorithm is used for account-level representations.

# Features	Method	CCR
64	doc2vec	0.5822
128	doc2vec	0.5916
256	doc2vec	0.6036
512	**doc2vec**	**0.6122**
1024	doc2vec	0.5959
64	word2vec-avg	0.7158
128	word2vec-avg	0.6901
256	**word2vec-avg**	**0.7166**
512	word2vec-avg	0.7080

In Table 4, we display a contrastive view of the effect of vectorization algorithm on CCR computed over test split. Note that since we employ the same multilayer perceptron (MLP)-classifier and the same experimental setup, the results are one-to-one comparable. We observe that despite the fact that *doc2vec* algorithm is designed to work with long documents, *word2vec* token average (*word2vec-avg* in Table 4) works significantly better. We believe this is because tweets have a maximum character limit and after preprocessing tweets tend to grow shorter in terms of the number of tokens they contain. In the end, these results show that there's a correlation between the content of the tweets an account posts and the likelihood of that account belonging to a bot account.

We also observe averaged word2vec representations significantly overperform doc2vec representations likely because the tweets, after preprocessing, have a shorter length than before; which is unideal. For doc2vec specifically, up until 512-features the performance increases. After this point, likely overfitting starts to occur. For word2vec, there seems to be less variation in performance based on number of features. In the end, we conclude that there's a correlation between the content of tweets of an account and the likelihood of that being a bot. Note that these results are obtained by employing the same MLP-classifier having an adaptive input size, thus the results are one-to-one comparable. Lastly, due to hardware restrictions, TF-IDF based vectorization cannot be applied in our case. Moreover, a transformer-based vectorizer without finetuning performed significantly worse than the results presented here. We believe finetuning a pretrained transformer or training one from the scratch might result in significantly better detection performance, however, once again hardware limitations are obstacles towards this goal. Thus, we leave this investigation for future work.

Table 5: Top-10 most frequently used tokens by the bot and real accounts across the train split.

Word	Account Type	#Occurrences
de	real	49329
la	real	32716
que	real	25539
u	real	23445
one	real	19868
en	real	19365
new	real	19151
el	real	18971
get	real	17308
time	real	17201
one	bot	26075
like	bot	25769
get	bot	23863
new	bot	23293
people	bot	22498
trump	bot	21962
time	bot	20594
day	bot	20574
today	bot	17137
year	bot	16921

5.4.1 Variations in Vocabulary Between Bots and Real Accounts. When compared to real accounts, we observed that a significant portion of their tweets are eliminated after preprocessing, which suggests that information content of these tweets are rather sparse. On the contrary, while some tweets are completely discarded due to their lengths, tweets from real accounts generally maintain a large portion of their tokens. In Table 5, we also display some of the most frequently employed tokens by both the bot and real accounts. Note that one obvious distinction is the fact that top most-frequent words are all lemmatized ancestors or abbreviations that have many derived word classes (adj.,adv., etc.) whereas for bot accounts very few tokens show such a property and instead use words without obvious meaning.

REFERENCES

[1] Seyed Ali Alhosseini, Raad Bin Tareaf, Pejman Najafi, and Christoph Meinel. 2019. Detect me if you can: spam bot detection using inductive representation learning. In *Companion Proceedings of The 2019 World Wide Web Conference*, 148–153.

[2] Stefano Cresci, Roberto Di Pietro, Marinella Petrocchi, Angelo Spognardi, and Maurizio Tesconi. 2016. Dna-inspired online behavioral modeling and its application to spambot detection. *IEEE Intelligent Systems*, 31, 5, 58–64.

[3] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd* number 34. Vol. 96, 226–231.

[4] Shangbin Feng, Herun Wan, Ningnan Wang, Jundong Li, and Minnan Luo. 2021. Twibot-20: a comprehensive twitter bot detection benchmark. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 4485–4494.

[5] Shangbin Feng et al. 2022. Twibot-22: towards graph-based twitter bot detection. *arXiv preprint arXiv:2206.04564*.

[6] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9, 8, 1735–1780.

[7] Sneha Kudugunta and Emilio Ferrara. 2018. Deep neural networks for bot detection. *Information Sciences*, 467, 312–322.

[8] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*. PMLR, 1188–1196.

[9] Kyumin Lee, Brian Eoff, and James Caverlee. 2011. Seven months with the devils: a long-term study of content polluters on twitter. In *Proceedings of the international AAAI conference on web and social media* number 1. Vol. 5, 185–192.

[10] Sangho Lee and Jong Kim. 2013. Warningbird: a near real-time detection system for suspicious urls in twitter stream. *IEEE transactions on dependable and secure computing*, 10, 3, 183–195.

[11] Yinhan Liu et al. 2019. Roberta: a robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

[12] J MacQueen. 1967. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, 281–297.

[13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

[15] Amanda Minnich, Nikan Chavoshi, Danai Koutra, and Abdullah Mueen. 2017. Botwalk: efficient adaptive exploration of twitter bot networks. In *Proceedings of the 2017 IEEE/ACM international conference on advances in social networks analysis and mining 2017*, 467–474.

[16] Gray Stanton and Athirai A Irissappane. 2019. Gans for semi-supervised opinion spam detection. *arXiv preprint arXiv:1903.08289*.

[17] Kurt Thomas, Chris Grier, Justin Ma, Vern Paxson, and Dawn Song. 2011. Design and evaluation of a real-time url spam filtering service. In *2011 IEEE symposium on security and privacy*. IEEE, 447–462.

[18] Chao Yang, Robert Harkreader, and Guofei Gu. 2013. Empirical evaluation and new design for fighting evolving twitter spammers. *IEEE Transactions on Information Forensics and Security*, 8, 8, 1280–1293.

Received December 26, 2022; revised XXXXX; accepted XXXXX