Proposal for the Investigation into Twitter Bot Identification and Impact Analysis for the U.S. 2018 Midterms

Richard Marquet

Georgia State University  
Student  
Big Data Analyst  
1-(678)-956-0475

rmarquet1@student.gsu.edi

Yubao Wu, Ph. D

Georgia State University  
Professor of Data Science  
Data Mining Adviser  
1-(404) 413-6125

Ywu28@gsu.edu

Noah Yasarturk

Georgia State University  
Student  
Big Data Analyst

1-(678) 230-4696

nyasarturk1@student.gsu.edu

**ABSTRACT**

In this paper, we describe the intention of our research endeavor: to mine Twitter and see to what extent we’ll be able to identify “bots” and examine, on a shallow level, the aims of these autonomous entities with regards to political discourse.

**CCS Concepts**

# • Information systems~Data mining   • Information systems~Data cleaning   • Information systems~Data stream mining   • Applied computing~Evidence collection, storage and analysis   • Applied computing~Digital libraries and archives   • Mathematics of computing~Statistical graphics   • Mathematics of computing~Exploratory data analysis

**Keywords**

Twitter; Python; tweepy; bot; election integrity; cybersecurity

# INTRODUCTION

SAMPLE: Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*Conference’10*, Month 1–2, 2010, City, State, Country.

Copyright 2010 ACM 1-58113-000-0/00/0010 …$15.00.

DOI: http://dx.doi.org/10.1145/12345.67890

While election integrity has been an enduring issue for democracies, republics, and other forms of government where voting matters, the United States of America’s 2016 Presidential Election brought the proliferation of deceptive information, internet echo chambers, and autonomous social media accounts to the forefront of our national dialogue and remains a subject matter that will likely be investigated for years to come. Though the spread of insincere news articles on the social media platform Facebook has been subject to a high degree of analysis (likely due to its widespread use at varying demographic makeups) less is known about the effects on U.S. elections that bad actors on Twitter have had.[[1]](#footnote-1) Twitter provides an environment where bots can flourish. Most obviously, the Twitter Developers’ “tweepy” package provides for ease of account automation. Due to Twitter’s format: limited characters, more public interaction and less “friend-to-friend” networking, along with the tendency to use usernames that act as aliases rather than as true identities, Twitter is a place where bots- programmed, autonomous profiles designed to fulfill a prespecified purpose- have been allowed to thrive.

The timing of our investigation is of importance to the study itself. We began fleshing out this problem in late August of 2018, just a few months ahead of the U.S. midterm elections (excluding early ballots the day to vote will be November 6th) during which all seats of the House of Representatives will be available to whoever secures the most votes within their respective district. The reason this is relevant is that due to our temporal proximity to the elections, dialogue concerning the topic found on Twitter will be heated and frequent. Additionally, for foreign agents interested in influencing the outcome of the U.S. elections (as has been substantiated by various sources, perhaps most importantly by the U.S.’s very own Central Intelligence Agency[[2]](#footnote-2)) the weeks leading up to the election itself present the most opportune time for exerting influence via social media as that is when the subject matter will be most apparent and fresh within the mine of the electorate.

We streamed a vast quantity of tweets and analyzed them at multiple levels. The streaming occurred in the week just prior to the election day itself- November 6th- with the goal of “catching” as many bots as possible.

We ended up detecting some autonomous, or at least semi-autonomous, Twitter accounts using a highly conservative approach to labelling entities as bots or not. Using the same methods and a less conservative approach, more detection may be possible, but considering the relatively small amount of tweets scraped, our results demonstrate a novel approach to detecting automated Twitter accounts.

# METHODS

## Data Stream Mining

We made use of the tweepy package (available as an open source module for Python users) to create a “stream”- by implementing a listener to wait for tweets to be posted using a few key words we specified beforehand, we can copy and route posted tweets we’d be concerned with to our local machines.

The terms were be concerned with were manifold. For the focus of our investigation, politically charged bots, we narrowed the keywords of our stream to: 'midterm', 'midterms', 'election', 'elections', 'vote', 'voting', 'votes', 'Vote', 'ballot', ‘poll’, ‘polls’- all of which are general terms concerned with the midterms- and 'Trump', 'democracy', 'Resistance', 'Blue Wave', 'BlueWave'- all of which are terms concerned with the 2018 midterms specifically. While President Trump is not being elected into or out of office this election, it is well known that the midterms are considered as somewhat of a referendum on the current president.5 ‘Resistance’ and ‘Blue Wave’ refer to a couple Twitter trending hashtags whose subject matter is concerned with deposing the current ruling Republican party.

### Stream\_Election\_twts.py

The tweet streaming phase corresponds to Stream\_Election\_twts.py. Here, I’ll outline a bit of pseudocode for how this program works.

This python program requires an installation of tweepy to already be native to the host machine (achievable using pip or anaconda). We also import os so as to check the file size later on.

The program first authenticates the user as one confirmed to access the tweepy package by providing access tokens. These strings are obtained by creating a Twitter Developer account.

A class is then made, in our case ‘StdOutListener’, that accepts as a parameter ‘StreamListener’ (this is specific to the tweepy package). Two methods are then defined in our case: on\_data and on\_error. on\_data defines what to do with statuses and retweets from users that we pick up from our stream.

Internal to our on\_data method, we write statuses to our local text warehouse repository along with a newline in beween each status. We additionally included a ‘return false’ check to stop the stream when the file size exceeded 100 megabytes so as to allow for quick processing later down the line and to allow for the text warehouse files to still be viewable using a text editor such as NotePad++.

Internal to out on\_error method, we simply print the error.

The main method of this program asks for input of the number of the stream currently being run. We do this so as to track each warehouse repository; the number inputs will be later used to identify warehouses to Preprocess in our 2nd program, Preprocess\_Twts.py.

The tweepy stream picks up tweets that have the keywords we provided either as text in the actual tweet or as hashtags.

### Securing Unadulterated Data

It has been (subjectively speaking, but generally nationally recognized) a characteristic of social media posts in recent times to be deeply concerned with and animated towards subject matter relevant to our political discourse. This fact has been exasperated with increased intensity with the election of our 45th president, Donald Trump, who is the subject of a large amount of controversy directly attributable to his Twitter activity. Additionally, his rhetoric understandably causes a large amount of rancorous commentary on all social media platforms. Most importantly, his actions are under a constant stream of analysis and debate. Acknowledging this, there is a certain degree of a lack of “normalcy” in terms of the discourse surrounding American politics mirroring the “maverick”-like nature of our current leader. This includes various political controversies of varying degrees of deviation from the traditional Oval Office “statesmanship”.

Throughout our investigation in the Fall of 2018, we made sure to pay attention to news headlines so as to check for anything that would amplify the severity of discourse to an unprecedented degree. Luckily, nothing of this nature (subjectively speaking) appeared to occur- no coups, no wars, no revolutions. While the situation with the journalist Khashoggi assassinated at the Saudi consulate in Istanbul, Turkey, and the U.S. Presidents somewhat milquetoast response, along with the recent findings of the Mueller investigation- that Manafort was known to have lied on Trump’s account during investigation- had provided heated news cycles, nothing which occurred in the political realm seemed to have an effect of consequence to our investigation.

### The tweepy Status Object

The items mined from Twitter using these methods, tweepy Status objects, come with many benefits. The Status object is a sort of list of lists containing information on the user who composed the tweet as well as information on the tweet itself. Should the tweet of concern be a retweet, information on the original tweet and composer is provided, as well.

Given that each of these objects has relevant information on users- follower count, following count, favourite count, tweet count, etc.- we can garner much from these objects without needing to make calls to Twitter’s database using tweepy simultaneously whilst streaming (i.e. call for the users’ follower count while writing their tweet). This is of importance to us in that Twitter puts a rate limit on how many calls are possible to about 1 per second maximum (or more technically and precisely, 900 calls/requests every 15 minutes).

With this understanding, our next step would be to develop methods by which we might analyze the raw data that is these Status objects to obtain relevant insight about the users whose tweets we’ve streamed.

## Initial Analysis of Accounts’ Attributes

### 2.2.1 Preprocess\_Twts.py

Now that the tweets were stored locally to our host machines using the aforementioned program, we move on to process the raw data in the text warehouses.

Here, I’ll explain at a high level how Preprocess\_Twts.py accomplishes this goal, with more detail on the individual methods following. This program has package dependencies including ‘csv’ and ‘datetime’.

The program focuses its analysis on a single warehouse appropriately numbered in the prior program. Upon user input, Preprocess\_Twts.py opens the text data warehouse and iterates through it line by line (each tweepy Status object is encoded to a single line; for a line to be a tweet, it only needs to have ‘"retweeted\_status":’ in it which occurs only once per tweet no matter how many times a given tweet has been retweeted). Each line is added to a list if it is in fact a tweet.

Once this tweet list is constructed, we iterate through it and run each tweet through one of eight test methods to determine which test a tweet triggers. Should a tweet trigger a test, the screen name (the name attached to the ‘@’ character traditionally) is isolated from the tweet using getScreenName() (which uses the triple split method described in 2.2.2 following) and appended to a list specific to that test. A final list stores all screen names.

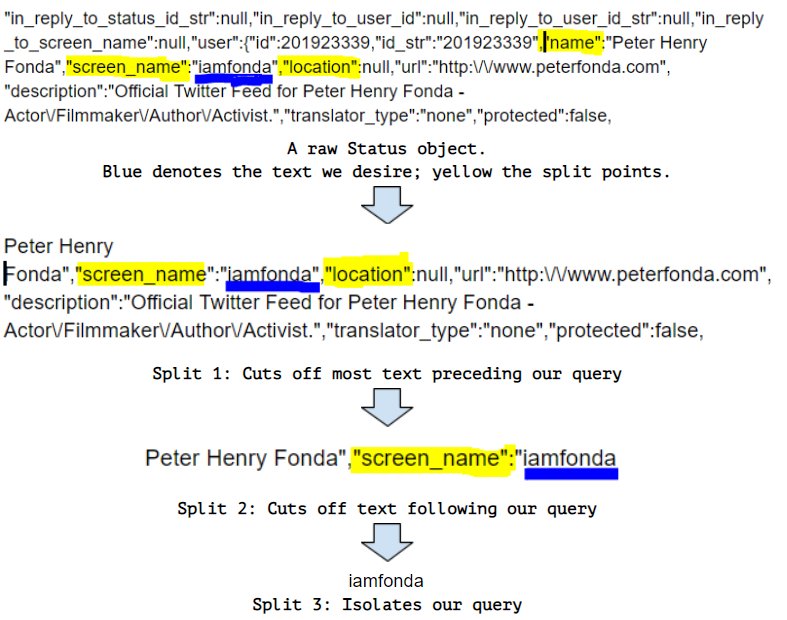
Following this list construction, 8 csv files are created and opened (or simply opened if this is not the first run of Preprocess\_Twts.py) and the screen names from the associated lists are iteratively added to the csvs.

*2.2.2 The Triple Split Method*

For many of the following test methods to be described, we rely on a technique to isolate relevant text from a raw Status object in our text repositories. As a model method to demonstrate this, we choose getScreenName().

The account owner of a tweet’s screen name immediately follows ‘”screen\_name”:’ in a Status object. The value that immediately precedes it is the location (which itself follows ‘”location”:’) and the value that immediately precedes that is account name, denoted by ‘”name”:’. To isolate the screen name, we make 3 cuts using these 3 strings as the blade. Python’s .split() method takes a string as a parameter and cuts the object string its called on along the parameter string, creating an array of the front portion of the split and the back portion. and handled the implementation of this procedure.

The following figure explains how this works:



**Figure 1: The Triple Split Method**

We make 3 cuts in order to isolate the precise value desired. The reason why we do not simply cut at points before and after the query is that in these mined Status objects a dictionary key (‘”screen\_name”:’, ‘”location”:’, etc.) is often used twice, especially if we are examining a retweet. For our text isolation to work, for each method used we had to examine the structure of our mined tweets and understand the order in which our query is found, else we may isolate the original owner of a tweet when we truly desired the retweeter, for example.

We implemented the triple split for the methods getScreenName(), getUserName(), getDateTwtd(), getTwtCnt(), getFlwrs(), getFavs(), getFlws(), and getAccntCreateDate().

*2.2.3 Initial Tests*

The following describes each of the 8 tests implemented on tweets from our warehouses in Preprocess\_Twts.py.

*2.2.3.1 Check if Location Provided*

The motivation for our first test was simple- someone created a vast quantity of accounts to automate may do the bare minimum and exclude their location. This test is hardly adequate on its own, however. To implement, we simply check if ‘”location”:null’ is in the user object internal to the Status object being analyzed.

*2.2.3.2 Check if a Bio Provided*

The motivation here is the same as the prior test. Implementation involves checking to see if ‘”description”:null’ is in the user object internal to the Status object being analyzed.

*2.2.3.3 Check if Profile Picture is Default Twitter Profile Picture*

Once again, this method shares the same motivation as the preceding two. We implement by checking if ‘https://abs.twimg.com/sticky/default\_profile\_images/default\_profile\_normal.png’ is in the user object internal to the Status object being analyzed.

*2.2.3.4 High Likes to Follower Count*

An automated account would interact with the surrounding Twitter ecosystem at a higher rate than the average user. What’s more, since the account is automated, it would not be interacting with users in an original, creative, or meaningful manner. An account automated to ‘like’ and therefore amplify selected accounts would have a high like to follower ratio because it is liking so many posts without actually gaining many natural users’ attention. We therefore isolate out accounts that have 100 times as many likes as followers. 100 is a somewhat arbitrary threshold, but 10 is too small, 1000 too large (e.g. an account with 203 followers would need to have 203,000 likes to trigger the test, giving is the idea that that threshold may be too high).

We isolate the favorite (like) count using the triple split method, isolate the follower count using the triple split method, and compare the ratio.

*2.2.3.5 High Tweet to Follower Count*

The motivation is very much similar for this method. We were however able to set the threshold much higher than the previous method- to 500 times as many tweets as followers- as it seemed that automated accounts were more likely to automate creating statuses by retweeting others or cycling through a set of tweets than they were to simply like a status. This may be because liking a status does less to amplify it than retweeting it does.

We isolate follower count using the triple split method and tweet count using the triple split method and compare the ratio.

*2.2.3.6 At Following Threshold Today*

An account created today and set to follow as many accounts as possible (so as to increase the potential size of its network) would hit Twitter’s daily 1,000 follow limit very quickly. An account at 1000 followers would therefore be highly suspect.

We isolate the following count using the triple split method and test to see if it equals 1,000.

*2.2.3.7 At Overall Following Threshold*

Twitter has two overall following limits- 5,000 or 5,000 + (the amount of followers over 5,000) \* .10. The latter limit is typically reserved for celebrity accounts. However, some bots are complex enough to garner user interaction. That bot accounts use attractive females as their profile pictures to attract male attention is widely accounted for online.

An account at this overall following limit would be exceptional absolutely. It either is a highly overactive user or has followed as many people as possible as quickly as possible so as to extend its influence as far as possible. We therefore check to see if the following count matches either of these values.

*2.2.3.8 Username Not in Screen Name*

When bots are created in mass, the designer would not be checking to ensure that all screen names match the user names. Screen names may be randomly generated alphanumeric characters or another account’s screen name with a number appended to it with little regard to endure the two match up. We therefore check to see if the two are widely dissimilar.

We split a username up both by spaces and by underscores (Twitter allows either in a user name), converted all split strings to lower case, and check to see if any of these split strings are contained in a lowercase version of the screen name.

*2.2.4 Duplicates*

Often, an account will tweet twice throughout the duration of our stream and our warehouse may have two or more of the same screen name. This is why prior to running each of these aforementioned tests we remove duplicate screen names with Remove().

Remove() accepts as input a list, creates an empty list, and iteratively adds elements of the input list to the empty list if it is not already found within the created empty list. Upon completion, it returns the recreated non-duplicate list.

*2.2.5 master.csv*

Our final csv takes account of every single test failed for a given screen name in columns and has the final column be the number of tests failed.

**2.3 Analysis of Suspect Accounts’ Tweet Activity**

*2.3.1 check\_Twts.py*

With the streamed tweets’ accounts analyzed for baseline suspect attributes using Preprocess\_twt.py, we now have the master csv with record of the number of tests each account triggers. We analyzed accounts that trigger a high number of tests using calls to Twitter through tweepy to assess the degree to which they display bot-like characteristics. The most number of tests triggered was 6 of the 8, so we chose this as our threshold to analyze accounts as the analysis of each account takes quite a while.

check\_Twts.py handles this more in-depth analysis. To begin with, we import its dependencies: tweepy, dateitme, SequenceMatcher from diffLib, Article from newspaper, time, and csv. newspaper is not native to python and may be obtained through installation of the newspaper4k package.

*2.3.1.1 Philosophy of Assessing Bot-like Activity*

Automated accounts are dumb. To elaborate, they will do a set number of operations on others’ tweets and their own repetitively without supervision. With this in mind, we develop 4 criteria to test suspect accounts further to verify whether or not they are extremely bot-like.

*2.3.1.2 Shared Article with Article Headline as Tweet Text*

Many accounts are created with the simple aim of amplifying subject matter. In our investigation, we saw that many articles from overtly left-leaning or right-leaning outlets with semi-inflammatory headlines would be shared by accounts with the headline of the article as the text of the tweet.

This is highly suspicious for anyone trained in Computer Science. Copying the text of an article to be posted and including it as the tweet text would not be too difficult to implement. Additionally, if a natural user were responsible for this activity, one would expect some typos at the very least. Finally, an automated account created with the goal of amplifying divisive discourse (a claim purported by the CIA as Russia’s goal behind operating in the social media landscape) would find no easier means to do so than to simply repeatedly share articles from biased news outlets.



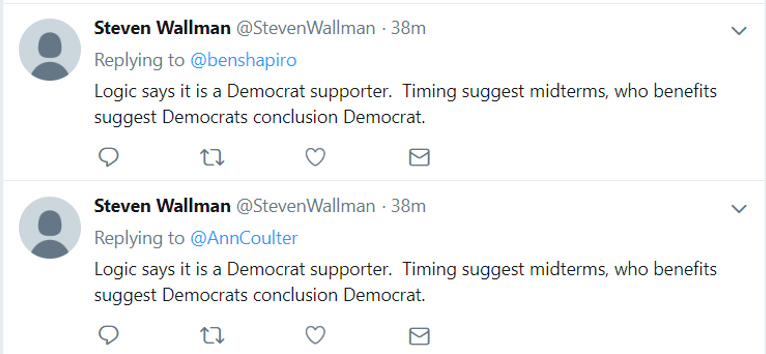
**Figure 2: An Example of a Headline Bot**

With this in mind, for an account in question, we implement this test by pulling 100 tweets from a given screen name and assess whether the “url” object internal to the “entities” object internal to Statis objects pulled using tweepy’s api.user\_timeline() method contains a relevant link. An attempt is then made to download and parse out the headline using Article’s provided (self-explanatory) methods. The headline is then compared for exact match with the tweet text.

If the two are precisely the same, we say this test is triggered and we are looking at a bot.

*2.3.1.3 Reposts*

The most obvious indicator of a bot would be reposting the same exact tweet. A designer aiming to automate an account may have a dictionary of tweets that an account may cycle through to promote one message or another.



**Figure 3: An Example of a Repost Bot**

100 tweets from a suspect account are pulled using the prior method (the same instance of it for efficiency) and the list is checked for duplicates. If there are any, we are clearly dealing with a repost bot.

*2.3.1.4 Mostly Reposts*

During our investigation, following the initial analysis, we physically combed through suspect accounts to identify relevant bot-like features. Often times, we saw an account would repost the same tweet with some punctuation added to it. For this reason, we apply the same methodology to this third test as the prior one only instead of a precise match, we look for 75% similarity using the SequenceMatcher().ratio() method from the diffLib package which assesses how similar two tweets are.

To get some idea for the relevance of a 75% threshold, in comparing 2 string: “Noah wants to know if this is similar.” and “Noah would like to know if this is very similar.”, we get a value of 79% similarity.

Accounts that have tweets that are 75% similar to one another (in addition to triggering several prior tests) are said to be extremely bot-like.

*2.3.1.5 Use of 3 or More Languages*

Oddly enough, some sources have suggested that accounts tweeting in 3 or more languages are seen as automated in that most people are not casually trilingual. We apply the same tweet-check methods and assess whether 3 languages are being used by recording the language of each tweet pulled in a list (the Status object has this value native to it), converting it to a set- thereby eliminating duplicates- and comparing the length of the set to see if it’s greater than 2.

*2.3.1.6 High-level Overview of check\_Twts.py*

This program opens master.csv, composes a list of screen names that have a value for tests triggered as 6, and then iteratively checks each screen name in the list by pulling 100 of its tweets to a variable, breaking those tweets down into a list of a list of tweet text and a list of tweet language and running the text and languages through the above methods.

The screen names are then written to our final output csv, botlike.csv, with values 1 or 0 depending on whether or not they trigger a test and a final value indicating whether any at all of the bot-like methods were triggered.

*2.3.1.7 Limitations*

Given that for every screen name that triggers 6 tests from the former analysis we are pulling 100 of its tweets, downloading any links contained in those tweets, and running a nested for loop (99^2 iterations) twice to assess repost or majority-reposts, each screen name check takes approximately 40 seconds on my host machine.

This means overall program runtime is dependent upon how many screen names were found to have triggered 6 of the 8 prior tests.

*2.3.2 Confirming Our Suspicions*

In the interval between executing Preprocess\_twts.py and check\_twts.py, we found that some screen names we attempted to pull tweets from no longer existed (verified by manually checking Twitter). This would imply that the account was very likely identified by Twitter itself as a bot and thusly removed. We therefore have an exception handled in this last program that details that the account is deleted if this be the case.

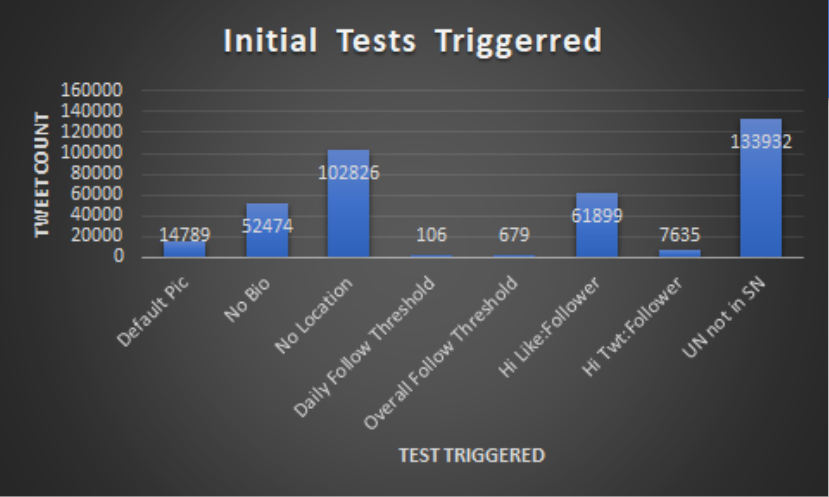
# Results and Discussion

## The Extent of our Analysis

We streamed 408,859 to text warehouses of a total size of 3.28 GB.

## The Fruits of Our Labor

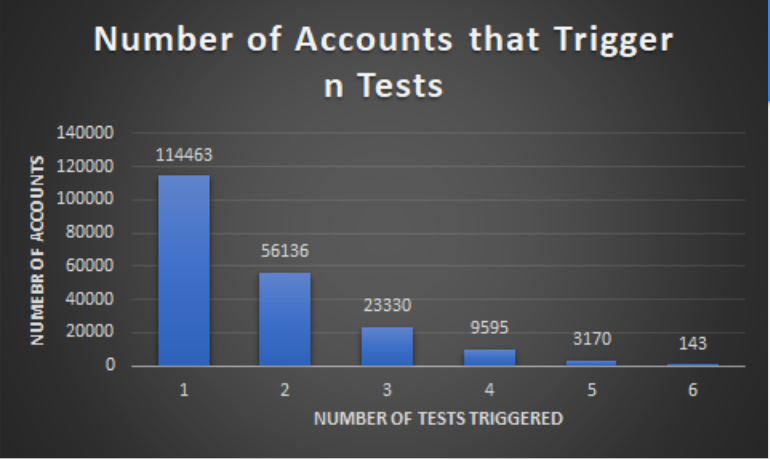
Following this, the results of the output of Preprocess\_Twts.py may be described in the following figure:



**Figure 4: Initial Analysis Output**

As is apparent, the method that was triggered the least was checking to see if the account was at the daily follow threshold (which makes sense as this is highly specific). Overall Follow Threshold closely followed. High tweet to follower or favorite to follower then followed that in terms of increasing count.

The number of accounts that trigger n tests can be described in the following figure:



**Figure 5: Number of Tests Triggered**

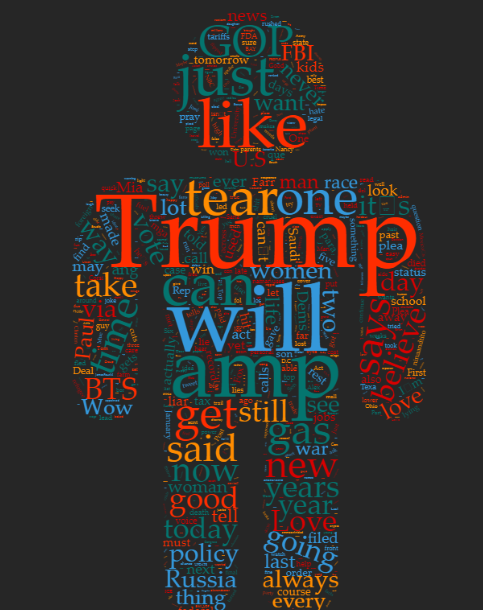
Given that only 143/400000+ accounts triggered 6 tests, we assume it is fair to say that 6 tests is a conservative enough threshold to proceed with check\_Twts.py and its additional 4 extensive tests. While we are not reporting the results of running our third program on the accounts that trigger 5 of the 8 tests (as testing 3170 accounts for extreme bot-like activity would take 3170 \* 40= 126800 seconds, divided by 3600 = over 35 hours) we assume from an incomplete analysis of accounts that trigger 5 of 8 initial tests that there would still be bots to be found at this less conservative threshold.

When the final analysis was conducted, we identified 82 extremely bot-like accounts.

This means that our rate of detection can be translated as: for n tweets scraped, we identify 0.0002n bots at the threshold of 6 tests triggered.

**3.3 Bot Content**

We generated a wordmap of the content of our detected bots’ last 25 tweets by using wordMap.py to construct a text file of their tweets and running that text file through <http://www.wordclouds.com> ‘s online resource. (Internal to wordMap.py is an unimplemented word map generator that is not as pretty as this website’s.)



**Figure 6: Word Map of Identified Bots’ Chatter**

As should be clear from the above figure, most of what these election-related bots were tweeting about had to do with POTUS 45.

**3.4 Some Curiosities for Later Investigation**

*3.4.1 K-Pop Stars*

It would seem that there are many Twitter accounts created with the aim of promoting K-Pop stars and persuading viewers to vote for various stars for local competitions. We inadvertently picked up these accounts as several of our stream terms (“vote”, for example) shared subject space with these accounts.

While it was not our goal to identify Twitter bots that have to do with other subject matters, that we ended up picking up these accounts- which often had extremely abnormal tweet count to follower or even following ratios- means that the methodology we used works.

*3.4.2 Semi-Autonomous Bots*

A few of the bots we manually investigated to verify that our methods were working appeared to churn out original account on a regular basis. Even with the original content, however, their numbers were extremely abnormal- very high tweet activity, like activity, very few followers, and Twitter screen names with several numbers following them (implying that the account simply selected the suggested screen name or that several accounts of a near-identical screen name with a single number difference have been created). An investigation into the networks of these bots may prove prudent as it is possible that bad actors are creating networks of bots and feeding them all the same content to tweet out. We have very little evidence from our investigation to support this claim, however.

*3.4.3 Concentrating on Specific Users*

Several of the bot-like accounts we looked into were retweeting or “trolling” high-profile political figures such as the president, Ann Coulter, Ben Shapiro, Cenk Uygur, or Bernie Sanders. This would suggest these accounts were made with the specific intent of following these users and engaging in politically-motivated amplification of divisive discourse.

**3.5 On the Efficacy of Our Results**

While our investigation has investigated accounts using 8 suspicion-garnering tests and 4 more tests that denote accounts as highly-suspicious, it is nearly impossible to actually verify that an account in question is autonomous of a natural user.

Because we have no true means of verifying these accounts as being 100% bots, we can only go by our suspicions.

Due to this, while the final result of our test yielded 82 high-probability results, we can only classify these accounts as “extremely bot-like”; we cannot go so far as to say that we found 82 bots when we have no means of verifying that with our limitations.

# REFERENCES

1. Hunt Allcott and Matthew Gentzkow. 2017. Social Media and Fake News in the 2016 Election. Journal of Economic Perspectives 31, 2 (2017), 211–236. DOI=http://dx.doi.org/10.3386/w23089
2. https://www.dni.gov/files/documents/ICA\_2017\_01.pdf
3. Noah H. Yasarturk. 2016. twitterdata.txt. (September 2018). DOI = https://github.com/Noah-Yasarturk/US\_Elections\_4\_Twits/blob/master/twitterdata.txt
4. About Twitter limits

<https://help.twitter.com/en/rules-and-policies/twitter-limit>

[5] David Remnick. 2018. The Midterm Elections Are a Referendum on Donald Trump. (October 2018). Retrieved November 29, 2018 DOI = https://www.newyorker.com/magazine/2018/11/05/the-midterm-elections-are-a-referendum-on-donald-trump

1. “Social Media and Fake News in the 2016 Election” references Facebook 33 times but Twitter only 3 times. [↑](#footnote-ref-1)
2. Taken from the CIA’s 2017 report “Background to “Assessing Russian Activities and Intentions in Recent US Elections”: The Analytic Process and Cyber Incident Attribution” which relays the scope and nature of cyber infiltrations preceding the U.S. 2016 Presidential Election. [↑](#footnote-ref-2)