Twitter Bot Detection and Classification

Using A Lexicon Based Approach

By

Krithika Saravanan, 100970975

July 2019

Supervised by Dr. Anthony White

School of Computer Science

Honours Project (COMP 4905)

Carleton University

Abstract

In this paper, we present a bot detection algorithm to identify Twitter bot accounts and to classify them as positive, negative or neutral bots. It's no surprise that bots are ubiquitous in social media. However, they can be problematic if they intentionally aim to manipulate valid information and spread misinformation, which can negatively affect users' opinion of various topics. The complexity of the problem lies in the fact that bots are becoming increasingly similar to humans, in order to avoid detection. We present an unsupervised lexicon based approach to determine the polarity of tweets.

The main objectives of this paper are to verify Botometer's ability to detect bots from a small set of Twitter accounts with accuracy. In addition to using a lexicon based approach to analyze the sentiment of tweets.

Acknowledgements

A special thanks to Dr. Tony White for supervising this project. I would also like to thank Andrew Pullin, Andrew Miles, Edina Storfer, and Mark Lanthier for their assistance.

Table of Contents

List of Figures

List of Tables

1 Introduction

Artificial intelligence can take on various definitions in various contexts, but the most concise definition by *Britannica,* describes artificial intelligence as 'the ability of computers to perform tasks commonly associated with humans'. In today's society, it seems as though AI is everywhere, from the more obvious application of self-driving cars to more subtle ones such as recommender systems found in popular platforms like Netflix and Amazon.

Machine learning is a subset of AI that consists of any computer program that is able to make predictions without human intervention. They are able to adjust themselves in response to the data they're exposed to, much like a human child. The learning aspect of ML refers to the fact that these algorithms attempt to optimize their results, either by minimizing error or maximizing the likelihood of their predictions being true.

The focus of this project will be on sentiment analysis, also known as opinion mining, a field within natural language processing, a subset of machine learning. The purpose of sentiment analysis is to identify and extract opinions within text and determine its polarity. It allows us to make sense of large amounts of textual data and provide key insights for businesses.

2 Background

Social media bots have been around since the inception of social media itself. There is generally a negative attitude towards them, since many of them are created to endanger democracy, cause panic during emergencies, expose private information, affect the stock market, and wreak havoc in general. However, bots can also serve a useful purpose like encouraging users to get their flu shot, provide earthquake alerts, health tips, share automated artwork, etc.

Identifying negative bots may help us decipher their patterns of behaviour and determine which features of sentiment make them stand out the most as bots. Furthermore, by readily identifying Twitter accounts as bots, the public can be educated to not fall prey to bot messages on Twitter. Moreover, the earlier that bots are discovered, the sooner their tweets can be prevented from spreading on the platform.

2.1 Bot Detection

Bot detection is the process of using a variety of tools and methods to identify bots in a collection of entities. The complexity of this varies depending on the type of bot and the set of attributes it contains. The goal here is to minimize the number of false positives (bots that are actually humans) and false negatives (humans that are actually bots).

Twitter

Twitter is a microblogging (condensed blogging) social media platform that launched in 2006. Users communicate via tweets, which are limited to 280 characters in length, to get their message across and efficiently and succinctly. Communication may come in the form of tweeting (creating messages), replying (responding to messages), and direct messaging (private chat). Users are able to reach out to others by tagging them with their handle, the '@' symbol followed by the target account's username. Users can also interact with others by using the hashtag symbol '#' to discuss specific topics, and categorize tweets for easy search and retrieval.

Additionally, users can choose the content they want to see on their timeline by following certain accounts. With the rise in Twitter's popularity over the years, companies, schools, celebrities, and even politicians have a Twitter account. The platform also provides the option to 'like' and 'retweet' users' tweets. Retweeted tweets are added to the user's timeline, which is a collection of posts that are created by or mention the user. Followers of an account are able to see all content on their timeline. [2]

Ever since Twitter became an effective tool to spread political messages in the presidential elections, the number of monthly active accounts in the US jumped from 45 million during the 2012 US presidential election to 67 million during the 2016 US presidential election.

Internet Bots

An (internet) bot is any automated software program that can repeatedly execute a variety of tasks. The use of bots on the web is so common that they currently make up 40% of all online traffic. [6] Tasks generally performed by these bots are commercial crawling, feed fetching, monitoring, and ticket touting.

Twitter Bots

A Twitter bot is a bot that operates on the Twitter platform from an account. Certain tasks can be automated from a Twitter bot like writing tweets, retweeting, and liking. There are no restrictions against creating a Twitter bot account as long as they do not break their Terms of Service via tweeting automated spam messages or misleading links.

Twitter bots, similar to general bots, are versatile enough to serve a variety of purposes ranging from basic tasks like following a user to more complex ones like carrying out a conversation with other users. Social bots are a type of bot that interacts with users, whose goal is to generate content that promotes a specific point of view. An estimated 9 to 15 percent of Twitter accounts are bots. [15]

Botometer (BotOrNot)

Botometer, formerly known as BotOrNot is a supervised machine learning API that checks the activity of a Twitter account and gives it a score based on how likely the account is to be a bot, where higher scores are more bot-like. This classifier is trained to classify accounts as bot or human based on tens of thousands of labeled examples. [14]

VADER

VADER stands for Valence Aware Dictionary and Sentiment Reasoner. It is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media, and works well on texts from other domains. [12]

Tweepy

Tweepy is an open source Python package that provides a convenient way to access the Twitter API with Python. Tweepy includes a set of classes and methods that represent Twitter's models and API endpoints, and handles implementation details such as rate limits, streams, OAuth authentication, results pagination, and more. [13]

2.3 Bot Classification

For our purposes, bot classification is the process of labelling already identified bots as positive, negative or neutral based on a variety of characteristics like text sentiment. Here it is more crucial that we are able to distinguish negative from positive and neutral tweets.

Sentiment Analysis

Sentiment analysis also known as opinion mining, sentiment mining or polarity mining is a field within natural language processing (NLP) that predicts whether a piece of information (i.e. text, most commonly) indicates a positive, negative or neutral sentiment on the topic.

This process involves four key tasks including data preprocessing, class labeling, annotation granularity, and target identification. Data preprocessing is vital to the outcome of the analysis since text collected from social media sites are generally unstructured and riddled with spelling errors and grammatical peculiarities. Class labelling, which is the process of annotating text into labels or classes such as subjective or objective. This task is generally performed prior to polarity classification to improve results.

Natural Language Processing

Natural language processing (NLP) is a branch of artificial intelligence that helps computers understand, interpret and manipulate human language. NLP is an interdisciplinary area of study combining fields like computer science and computational linguistics, in order to fill the gap between human communication and computer understanding.

Machine Learning

Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

Supervised Learning

Supervised learning is the process of an algorithm learning from the training dataset to determine the mapping function from the input to the output. These problems can be further broken down into classification and regression.

Unsupervised Learning

Unsupervised learning is the process of an algorithm trying to model the underlying structure or distribution in the data in order to learn more about it. These problems are more complex than supervised learning and can be further grouped into clustering and association. Some popular examples are support vector machines, linear regression and random forest.

Classification

Classification is a technique for categorizing data into unique classes where we can assign a label to each class. Grouping data into two distinct classes is called binary classification (eg. male and female). Classification with more than two distinct classes is called multi-class classification (eg. types of crops).

Naive Bayes

Naive Bayes is a probabilistic classifier inspired by Bayes theorem, where it assumes that the attributes are conditionally independent. One of its major advantages include requiring only a small amount of training data to estimate the necessary parameters. Additionally, it is very fast compared to more sophisticated methods. However, on the downside it is a poor estimator.

To further break down the steps, a naive bayes classifier first calculates the prior probability for given class labels. It then finds the likelihood probability with each attribute for each class, puts these values in the Bayes formula and calculates posterior probability. Finally it determines which class has a higher probability, given the input belongs to the higher probability class.

The strengths of this type of classifier lie in its ability to handle noisy data. That is, null values are ignored and irrelevant features are uniformly distributed such that they do not have significant influence on the classification result.

Positive Bots

We identify positive bots as being those that are created for social good (eg. earthquake alerts, health tips) and refrain from using negative or inappropriate language.

Neutral Bots

We identify neutral bots as being those that don't meet the requirements of either a positive or negative bot. These are generally of the fake follower type, that exist to just make a user appear more popular or influential than they are.

Negative Bots

We identify negative bots as those that are created to spread negative, misleading or false information (eg. Russian bots tweeting about the US election), impersonate others, or post spam content.

Lexicon Based Approach

Lexicon based approach are widely used to classify unsupervised text sentiment. They work by detecting tweet polarity using a dictionary of words with semantic scores attached to them to calculate the polarity of a tweet and incorporates POS (part of speech) tagging. The main issue with this approach is that it doesn't provide domain or context dependent meanings. For example, the word 'awfully', can be used to convey both a positive or negative opinion depending on the context in which it is used. Consider the sentence, 'It was an awfully good performance' which is a positive opinion versus 'We played awfully' which clearly expresses a negative opinion.

NLTK

NLTK is a leading platform for building Python programs that is used with human language data for applying in statistical natural language processing (NLP). It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning. It also includes graphical demonstrations and sample data sets as well, which explains the principles behind the underlying language processing tasks that NLTK supports. [11]

3 Related Works

3.1 Pew Research Center

A few related works were examined prior to selecting this specific approach to detecting Twitter bots. The Pew Research Center study done in 2017 conducted work in determining the

role bots play in sharing links on Twitter. It involved examining a collection of 1.2 million tweeted links, collected over the span of several months to measure how many came from suspected bot accounts. They found that about two-thirds of these links were shared by automated accounts.

The researchers utilized an open source machine learning algorithm called Botometer, which has been widely used in the field of academia and independent research. This tool is trained to recognize bot behaviour based on patterns in a large dataset of accounts that were pre-labelled as either bots or non-bots. Botometer is able to read over a thousand various features for each account and assigns a score between 0 and 1. The likelihood that the account is automated is positively correlated with the score.

3.2 SMU Data Science Review

In another paper titled 'Supervised Machine Learning Bot Detection Techniques to Identify Social Twitter Bots', they discuss using a novel, complex machine learning algorithm that utilizes a range of features for bot detection. This technique using key attributes including the length of names, reposting rate, temporal patterns, sentiment expression, followers to friends ratio, and message variability claims to have 97.75% accuracy. Bot identification is done by first separating three main areas for analysis: profile, account activity, and text mining. They identify bots using a set of features like the lack of a bio or profile picture, follows less than 30 accounts, has over 1000 followers, less than 50 tweets.

4 The Twitter Dataset

4.1 Twitter Handle Data

I had originally planned on collecting a large number of twitter accounts however, given the time constraints and other technical difficulties I chose to limit the dataset to 645 accounts. The process involved with collecting these accounts went through several trial and error phases. I initially planned on using Twitter's search API to search for tweets containing links to the top 100 sites and save the corresponding accounts that the tweet originated from. As this was somewhat outside my area of expertise, I instead chose to collect a small, random subset of followers from a popular Twitter account. However, Botometer's accuracy proved to be unsuccessful when comparing automatically classified bots against a manual inspection. That is, from the 253 accounts that passed as bots under our custom Botometer threshold of 0.43, only 21 or 8.3% were most likely bots. The final method for acquiring Twitter handles consists of

combining a predefined list of bots with popular and regular non-bot accounts, giving us a total of 645 handles.

4.2 Tweet Data

In preparation for the sentiment analysis segment of the project, I collected a large dataset of tweets for the identified bots from phase 1. The general process consists of iterating through the list of bots, extracting the first x tweets from the timeline that meet a certain criteria. Specifically, the language must be English or undefined, cannot contain possibly sensitive content. Additionally, text preprocessing is a crucial step that involves cleaning the text to ensure accuracy of the sentiment analysis. That is, I removed words that contain numbers, monetary values and numbers containing decimals, extra letters added to words for emphasis, unwanted characters that do not help to express sentiment, and duplicate tweets. I trained the machine learning model using a set of about 1.6 million tweets from the Sentiment140 dataset.

5 Overview of ML Classifiers for Sentiment Analysis

This section aims to provide a summary of what is explained in depth in Ismail et al.'s paper on this subject matter. There are several ways to deal with NLP problems including Naive Bayes, Support Vector Machine (SVM), and neural networks. Naive Bayes is often an ideal choice due to their simple design, speed, reliability, and accuracy in various applications.

5.1 Multinomial Naive Bayes Classifiers

Multinomial Naive Bayes is a specific instance of Naive Bayes classifier which uses a multinomial distribution for each of the features. It explicitly models the word counts and adjusts the underlying calculations to account for repeated words. In addition, it lets us know that each probability of feature i given some class c is a multinomial distribution rather than some other distribution. This works well for data which can easily be turned to counts, such as word counts in text documents. Contrast this to simple Naive Bayes which models a document as the presence and absence of particular words.

The probability of a document d being in a class c is computed as the following, where P(tk|c) is the conditional probability of term tk occurring in a document of class c. We interpret P(tk|c) as, 'with what level of certainty does tk contribute to ensuring that c is the correct class?'. P(c) is the prior probability of a document occuring in class c. If a document's terms do not

provide clear evidence for one class versus another, we choose the one that has a higher prior probability.

Symbols		
$\prod$	Capital pi	Product of all values in range of series
$\propto$	Proportional to	Proportional to
$P(A \mid B)$	Conditional probability function	Probability of event A given event B occurs
$\sum$	Summation	Sum of all values in range of series
$\|\| x \|\|$	Double vertical bars	Norm
$\langle x, y \rangle$	Inner product	A scalar function of two vectors
λ	Lambda	-
δ	Delta function	-

TABLE 5.1

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

FIGURE 5.1 a

5.2 Bernoulli Naive Bayes Classifiers

Bernoulli Naive Bayes is a great alternative to the multinomial model. It also has the same time complexity as the multinomial model. When classifying a document, the Bernoulli model uses binary occurence and ignores the number of occurrences. Thus, the Bernoulli model

generally makes many errors when classifying longer documents. To exemplify, it may erroneously assign an entire book to the class ‘India’ due to a single occurrence of the term ‘India’. Furthermore, in the Bernoulli model the probability of nonoccurrence is factored into the final probability. The reason for this is that only this model, models the absence of terms explicitly.

The decision rule for Bernoulli Naive Bayes is based on the formula

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

FIGURE 5.2 a

5.3 Support Vector Machine Classifiers

A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be utilized for both regression and classification purposes. SVMs are commonly used in classification problems. The underlying concept of SVMs are based on finding a hyperplane that best divides a dataset into two separate classes. The support vectors themselves are the data points closest to the hyperplane. Furthermore, they are the points of a data set that if removed would alter the position of the dividing hyperplane. For this reason, they are considered to be critical elements of a data set. A hyperplane is a line that linearly separates and classifies a set of data. The further from the hyperplane our data points are, the more confident we can be that they have been correctly classified. Thus, our data points should be as far away from the hyperplane as possible and on the correct side.

As new testing data is added, whichever side of the hyperplane it lands will decide the class that it belongs to. The pros of an SVM are that it is fairly accurate, works well on smaller cleaner datasets, and it can be more efficient since it uses a subset of training points. However, the cons are that it is not suitable for larger datasets since the training time with SVMs can be quite long and it is less effective on noisy datasets with overlapping classes.

The loss function helps maximize the margin between the data points and the hyperplane and is shown below.

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

FIGURE 5.3 a

Using the loss function, we are able to take partial derivatives with respect to the weights to find the gradients and update our weights.

$$\frac{\delta}{\delta w_k} \lambda \parallel w \parallel^2 = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k} \left(1 - y_i \langle x_i, w \rangle\right)_+ = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

FIGURE 5.3 b

There are two possible cases to consider when choosing the appropriate gradient update function.

Case 1 (no misclassification) : The model is able to correctly predict the class of our data point, so we are only required to update the gradient from the regularization parameter.

$$w = w - \alpha \cdot (2\lambda w)$$

FIGURE 5.3 c

Case 2 (misclassification) : The model makes an error predicting the class of our data point, so we include the loss along with the regularization parameter to perform gradient update.

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

FIGURE 5.3 d

5.4 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are popular models that are widely used in many NLP problems, and use sequential information such as text. These neural networks have an edge over traditional feedforward ones since they perform the same task for every element of a sequence, where the output depends on the previous computations. In other words, RNNs have a memory that allow it to capture information about what has been calculated thus far. The pros of neural networks are that they are flexible and can be used for both regression and classification problems, they are good to model with nonlinear data with large number of inputs, predictions are fast, and they can be trained with any number of inputs and layers. However, some of the cons are that they we do not know how they work (ie. black boxes), they are computationally expensive and time consuming to train with traditional CPUs, and they depend greatly on training data. The last point leads to problems with overfitting and generalization.

6 Methods

6.1 Phase I : Bot Detection

Libraries used

- Botometer

For this segment of the project, I collected a small sample of known Twitter bots, unpopular non-bot accounts, and popular bot accounts. I then combined them into a single list of 645 accounts for Botometer to classify as bots or non-bots.

get_bom_scores()

The first step is to iterate through the list and discard the accounts that do not meet certain criteria. We must check that they are English based accounts. The Tweepy API does not provide a straightforward way to accomplish this, so instead we will check if the first tweet on their timeline is either English or undefined. If there is any error or it does not meet the aforementioned criterion we skip the account and continue to the next one. Otherwise, we get Botometer to compute their English score, and write the results as a map, where the handle is the key and the score is the value, to a text file called scores.

Here is a sample screenshot of the results.

```
arctoplabs 0.9632891569033074
topnotifier 0.7866262980972412
interlinklabs 0.2691022464970497
trista_dzuba 0.07343326191269153
account bretones skipped
marcuskehrle 0.06783456974919343
TeamQuest_Corp 0.6698271269837389
Error: account NewWorkshop skipped
paul1978a 0.19374947883382485
Error: account DiegoRBaquero skipped
```

FIGURE 6.1 a

get_bot_handles()

The next step is to iterate through the scores and replace all the unwanted characters (eg. {, }, \") with empty spaces and split on the commas to create a list of handle, score pairs separated by colons. We then iterate through this list to read in the handles and scores by splitting on the colon character. The final step in parsing the data into a proper format is to create a new map called scores with the same key value pairs, with all empty spaces stripped from the score values.

Next, we set the threshold for detecting bots to 0.43, as it has been found to maximize accuracy, according to the Pew Research Centre [6]. Now we can add these identified bots to an array and write them to a text file called bots.

get_bot_all_scores()

We also create a function to collect all Botometer scores as well, so that we can compare the English scores to the universal scores. As we can see from the diagram, both scores are quite different given that the universal score omits sentiment and content features, both of which are English-specific, whereas the English score uses all six categories of features.


```
 'scores': {'english': 0.9402123048447205, 'universal': 0.9500427150733608}, 'use
 'scores': {'english': 0.8005982245473489, 'universal': 0.6729387518017023}, 'use
.0}, 'scores': {'english': 0.7565854729457974, 'universal': 0.6729387518017023}, '
: 2.5}, 'scores': {'english': 0.5905129273413294, 'universal': 0.6545596461071505}
?}, 'scores': {'english': 0.7719542327261165, 'universal': 0.49505756400804624}, '
1.9}, 'scores': {'english': 0.7719542327261165, 'universal': 0.49505756400804624}
: 2.2}, 'scores': {'english': 0.5905129273413294, 'universal': 0.7409598586102422}
```


FIGURE 6.1 b

6.2 Phase II : Bot Classification

Libraries used

- Nltk
- Tweepy
- Preprocessor
- VaderSentiment

6.2.1 Unsupervised lexicon based approach

For this segment of the project, I utilized three methods, one to collect the tweets for each bot, another to handle the sentiment analysis, and another to handle the bot classification.

get_tweets()

The first step is to initialize the tweet preprocessor and set the options to remove URLs, handles, and reserved words from tweets. Then I initialize a list of unwanted characters called ‘bad_chars’ and set it to the set of punctuation from the string library. I modified the list by

removing the characters '!, ., ?' since these account for sentiment and adding the characters '-, ..., ..' since they do not add significant meaning.

Next I iterate through the list of bot handles and remove any extra whitespace before grabbing a list of tweets from the account's timeline using Tweepy. For the parameters, I set the count to 100 since it seems to return less than the specified amount. I also set it to only include original tweets since retweets would not reveal much about the account itself.

In addition, I make sure the tweet's language is either English or undefined and does not contain sensitive material before separating the tweet's text by whitespace into an array. I iterate through this to perform some text preprocessing steps.

- Removing words where letters and numbers are mixed (eg. LX23)
- Removing monetary values and numbers with decimals in the middle (eg. $325, 4.25)
- Removing extra letters added to words for emphasis (eg. THIIIIIS becomes THIS)
- Removing unwanted characters from the 'bad_chars' list

clean_tweets_dict(this_map)

I remove bots that have less than 50 tweets in their timeline. This value was arbitrary chosen after attempting to do some research on this failed to return useful results. Furthermore, sentiment analysis on individual accounts is not something that is commonly done and therefore I must rely on my own intuition. I then write the results to a pickle file for later use.

sentiment_analysis(sentiment_values, num_tweets)

The first step in conducting sentiment analysis is to open the pickle file containing the tweets and load the results. Then I create an instance of the SentimentIntensityAnalyzer from the VADER library. Iterate through the tweets array to determine the compound VADER scores and accumulate it to the user_sentiment variable, which would allow us to average scores over all tweets for each user.

The compound VADER score is computed by summing the valence scores of each word in the lexicon, adjusted according to the rules, and then normalized to be between -1 (very negative) and +1 (very positive). Furthermore, it is the most useful metric for determining the sentiment for a given sentence. It can be thought of as a normalized weighted composite score.

The number of tweets VADER had available to analyze would affect the final score, so I tested the method with values of 20, 30, 40, and 50 tweets. The method returns a map of key,

value pairs where keys are handles and values are the compound VADER scores rounded to 4 decimal places.

```
By the time they realize it's actually an ending story we'll have their money. 0.0
I have seen the future.---------------------------------------- 0.0
Inside every teen is a wolf just waiting to play basketball.----- 0.34
Deep within all of us theres a baby on board.-------------------- 0.0
The fact that you dont know where Carmen San Diego is only makes you want her more. 0.0772
I think I might win another Clio for this one-------------------- 0.5859
Find out what you cant do on television and well do that.-------- 0.2732
If America is the new Rome don't we deserve our own gladiators?-- 0.0
Every moment of our lives is a flight into unknown loves and pains so youd better be wearing
The Super Bowl is Americas truest holiday a pageant of our greatness and way of life. It dema
The sun crests on a new decade. A decade of hope and possibility. A decade that doesnt ask wh
We all have a vault deep within our souls where we keep our secrets and pain. A vault that nc
The more you build yourself up the bigger your inevitable fall. Thats Jenga. 0.0
```

FIGURE 6.2 a

```
AVERAGE SENTIMENT FOR USER:     hotteststartups     0.0931
```

FIGURE 6.2 b

classify_bots(sentiment_values, pos)

The first step in classifying bots is to determine the threshold for separating neutral from positive and negative tweets. I set this value to the standard .05, mentioned on the official VADER github page. The pos parameter refers to the value in the dictionary that determines the maximum number of tweets we are using for the classification. The result is a breakdown of the number of positive, negative, and neutral bots.

6.2.2 Supervised naive bayes approach

Initially, this approach aimed to classify bots by training a Naive Bayes machine learning model on a large collection of tweets, approximately 1.6 million, taken from the Sentiment140 dataset. The tweets have been annotated as 0 is negative and 4 is positive. However, this did not work effectively since the model was trained on positive and negative tweets only and the test set consisted of neutral tweets as well. The solution to this would be to either remove the neutral tweets, which would be very time consuming and nearly impossible, retrain the model with 533333 neutral, 533333 negative, and 533333 positive tweets which is also time consuming or

find a way to determine the optimal boundaries between positive and negative tweets. For the final option, with simple phrases, we can set specific words as markers for neutrality. But with more complex cases like “I love apples but hate bananas”, this becomes confusing because we need to decide if the individual phrases cancel each other out. What about sentences with multiple phrases, like “I love apples, hate bananas, and like oranges”? We can see how this becomes increasingly complex as the length of the sentence increases.

The training_data.csv dataset contains the following fields:

- **ID** : the unique identifier of the tweet (1960146170)
- **Sentiment** : the sentiment of the tweet (positive/negative)
- **Label** : the topic label of the tweet (tech)
- **Text** : the content of the tweet (Linux is cool)

Since most of the source code was taken from a tutorial, I will not be going into much detail. However, the general steps are as follows: getting the authentication credentials, authenticating our Python script, creating the function to build the test set, preparing the training set, pre-processing tweets in the data sets, building the vocabulary, matching tweets against our vocabulary, building our feature vector, training the classifier, and testing the model. [4]

7 Results and Discussion

7.1 Phase I : Bot Detection

Although all the accounts that Botometer identified as bots were in fact bots, it did miss quite a few accounts from the original bot list, that could have been detected with a lower threshold, I did not want to increase the risk of false negatives (ie. non-bots that are classified as bots). Since this is not the focus of the project, we will not be going into further detail.

7.2 Phase II : Bot Classification

7.2.1 Unsupervised lexicon based approach

We can see from this screencap that classifying bots on varying number of tweets gives us different results, however the closest results are obtained with 40 and 50 tweets. The more tweets VADER has available, the more accurately it can detect polarity. Thus, we can say that there are 298 positive bots, 35 negative bots, and 47 neutral bots. It would be time consuming to

verify the accuracy of the results for each bot since we would have to examine all its tweets and make a decision about the type. So I decided to analyze the tweets of an account from each category.

For example, if we choose a positive bot like tinycarebot and observe its tweets, we can see that it is clearly positive. Here is a sample tweet: 'here's a personal reminder, go outside if you can please! Ily!'. Furthermore, here is a negative bot called TwoHeadlines and one of its tweets, 'Lethal Weapons old presidential campaign is still sitting on illegal campaign contributions'. Since news accounts generally report negative news headlines to get readers' attention, this makes sense. Moreover, here is a neutral bot called libraryofevery with the following tweet 'A collection of hurdlers every hurdles. every word hurdling.' This tweet has neither a negative nor positive sentiment, so it has been classified as neutral.

```
CLASSIFY:   20 tweets
POSITIVE BOTS:  72
NEGATIVE BOTS:  8
NEUTRAL BOTS:   15

CLASSIFY:   30 tweets
POSITIVE BOTS:  147
NEGATIVE BOTS:  17
NEUTRAL BOTS:   26

CLASSIFY:   40 tweets
POSITIVE BOTS:  222
NEGATIVE BOTS:  26
NEUTRAL BOTS:   37

CLASSIFY:   50 tweets
POSITIVE BOTS:  298
NEGATIVE BOTS:  35
NEUTRAL BOTS:   47
```

FIGURE 7.2 a

From this, we can see that there is an overwhelming number of positive bots, but we can contribute this to the fact that our initial set of bots was hand picked, which would skew the results. However, using a lexicon based approach to classify bots is still very effective and would work well for a larger dataset also.

7.2.2 Supervised naive bayes approach

I was unable to test it on a larger dataset by using a test split since it would inevitably contain neutral tweets, and therefore would not be useful to evaluating the accuracy. Instead,

running the trained model on a sample set of ten tweets, we get an accuracy score of 6/10, or 60%. Unfortunately, this is not as good as I had expected considering the model was trained on a dataset of 1.6 million tweets. We could extend this to classify bots by manually labelling all tweets in a subset of our dataset (ideally 25% of our original dataset) as positive or negative and evaluating the accuracy of the model.

Testing		
Tweet	**Actual Sentiment**	**Predicted Sentiment**
'Suicide Bomber Kills Officials in Mayor's Office in Somalia's CapitalSuicide Bomber Kills Officials in Mayor's Office in Somalia's Capital'	Negative	Negative
'Trump Says Mueller Was 'Horrible' and Republicans 'Had a Good Day''	Negative	Negative
'DO This gives a very good view of the water. Small excursions to close by islands might be possible.'	Positive	Positive
'please eat something healthy'	Positive	Negative
'You have a great sense of humor.'	Positive	Negative
'Germany In Political Turmoil As Coalition Talks Fail lmao'	Negative	Negative
'Confidence goes a long way. Even if it doesn't come naturally fake it 'til you make it!'	Positive	Negative
'MIDRANGE Try it good place hangout in eveinings and night. Chinese and Malaysian food.'	Positive	Positive
'As Trump Accuses Iran He Has One Problem His Own Credibility'	Negative	Negative
'please dont forget to listen to a song you enjoy'	Positive	Negative

TABLE 7.2

8 Ethics

In this modern age of big data, it is important that information is collected and used in an ethical manner and follows all of Twitter's guidelines and policies on fair use. These guidelines ensure that ethically sourced data can be used in research, but the guidelines are in constant flux. Initially, Twitter allowed any data collected on its platform to be shared as a complete data set. Twitter has now amended its policies to only allow tweet or account IDs to be publicly shared. Although this allows users to trust that their information is safe, since it is removed from Twitter and cannot appear in future data sets, it makes conducting research in this field more difficult.

A major ethical issue is informed consent. Participants must opt into the study in order for their work to be used. This ensures that subjects know the purpose of the study and what they are signing up for. However, it can be argued that since Twitter is a public social media forum, where publicly shared tweets can be read by anyone, consent is not necessary in this case. Furthermore, it is common practice to conceal personal information for a study (ie. not showing account names). However, it is possible that a user's identity could be revealed with the contents and timestamp of a tweet. Furthermore, the Twitter Search API could be used to identify a user by inputting the tweet text and the timestamp. Since the scale of this study is small, I am not concerned with publishing individual tweets. The purpose of collecting tweets for this study is to perform sentiment analysis on each tweet's content. I will aim to protect the anonymity of users in this study by not publishing personally identifying or account identifying information.

9 Future Work

If this project were to be continued in the future, I would most likely want to include a larger and more representative dataset for more comprehensive bot detection and classification. Furthermore, I believe it would be wise to incorporate semantic features in the training model in order to separate sentiment from semantic. I would also want to cross validate the results to ensure that they are accurate. I would also want to find a way to classify neutral tweets as well, having only trained the model on positive, negative, and neutral tweets. This would further allow me to classify bots as positive, negative, and neutral as well such that we can compare the machine learning model to the lexicon based model of classification.

Moreover, I would want to create another machine learning model that incorporates the VADER compound scores for each user as three new features in the feature vector (ie. positive, negative, and neutral). This would allow for more accurate bot classification since VADER is able to handle tweets containing negations, contractions, punctuation, degree-modifier, etc.

Finally, I would want to determine the accuracy of the machine learning model using a 75/25 train-test split so that I can predict how well it would be able to classify tweets in the real world.

10 Conclusion

Although this project turned into more of a case study than a full fledged research study due to the limited size of the data set, it shows the vast possibilities in the field of machine learning and sentiment analysis what can be accomplished. There is currently lots of research being done on improving methods to detect sentiment of social media content, especially using neural network and deep learning models. These methods have proven to be effective as they are vastly more complex than regular machine learning models and incorporate more data to understand edge cases. The hope is that in the future we can rely on hybrid models that combine the strengths of multiple methods to effectively detect bots to nearly perfect accuracy and remove them before they take over the majority of the content. This would in turn allow for a more open platform without having to worry about falsified or explicit content.

References

[1] Anon. 2019. A Brief History of AI. (January 2019). Retrieved January 2019 from https://aitopics.org/misc/brief-history.

[2] Phillip Efthimion, Scott Payne, and Nicholas Proferes. 2018. Supervised Machine Learning Bot Detection Techniques to Identify Social Twitter Bots. (February 2019). Retrieved February 2019 from https://scholar.smu.edu/cgi/viewcontent.cgi?article=1019&context=datasciencereview.

[3] Iliya Valchanov. 2018. False Positive and False Negative. (April 2018). Retrieved April 2018 from https://towardsdatascience.com/false-positive-and-false-negative-b29df2c60aca.

[4] Anas Al-Masri. 2019. Creating The Twitter Sentiment Analysis Program in Python with Naive Bayes Classification. (February 2019). Retrieved May 2019 from https://towardsdatascience.com/creating-the-twitter-sentiment-analysis-program-in-python-with-naive-bayes-classification-672e5589a7ed.

[5] Stefan Wojcik, Solomon Messing, Aaron Smith, Lee Rainie and Paul Hitlin. 2018. Bots in the Twittersphere. (April 2018). Retrieved January 2019 from https://www.pewinternet.org/2018/04/09/bots-in-the-twittersphere/

[6] Anon. 2019. Bots Drive Almost 40% of Internet Traffic. (April 2019). Retrieved June 2019 from https://wiredelta.com/bots-drive-almost-40-of-internet-traffic/.

[7] J. Clement. 2019. Number of monthly active Twitter users worldwide from 1st quarter 2010 to 1st quarter 2019 (in millions). (June 2019). Retrieved June 2019 from https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/.

[8] Heba M. Ismail, Saad Harous, Boumediene Belkhouche. 2016. A Comparative Analysis of Machine Learning Classifiers for Twitter Sentiment Analysis. (February 2016). Retrieved June 2019 from https://pdfs.semanticscholar.org/1167/a7841877d8ea2f988e73d066b95c72589515.pdf.

[9] Rohit Gandhi. 2019. Support Vector Machine — Introduction to Machine Learning Algorithms. (June 2018). Retrieved July 2019 from https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[10] Anon. 2009. Naive Bayes text classification. (April 2009). Retrieved July 2019 from https://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html.

[11] Anon. 2019. Natural Language Toolkit (NLTK). (June 2019). Retrieved July 2019 from https://www.techopedia.com/definition/30343/natural-language-toolkit-nltk.

[12] C.J. Hutto. 2018. VADER Sentiment Analysis. (December 2018). Retrieved May 2019 from https://github.com/cjhutto/vaderSentiment.

[13] Joshua Roesslein. 2009. Tweepy Documentation. (June 2009). Retrieved January 2019 from http://docs.tweepy.org/en/v3.5.0/.

[14] Anon. 2017. Botometer Python API. (April 2017). Retrieved January 2019 from https://github.com/IUNetSci/botometer-python.

[15] Zoey Chong. 2017. Up to 48 million Twitter accounts are bots, study says. (March 2017). Retrieved June 2019 from https://www.cnet.com/news/new-study-says-almost-15-percent-of-twitter-accounts-are-bots/

Appendix A

```
def get_bom_scores():
    accounts = []
    scores = {}
    # Read scores from file into list
    with open('accounts.txt', 'r') as fileobj:
        for row in fileobj:
            row = row.rstrip('\n')
            accounts.append(row)
    # Get botometer scores for a list of accounts
    for handle, result in botometer.check_accounts_in(accounts):
        try:
            tweet = api.user_timeline(screen_name=handle, count=1, include_rts=True)[0]
            # If user profile language is english print english score
            if (tweet.lang == 'en' or tweet.lang == 'und' and tweet.possibly_sensitive
== False):
                english_score = result['scores']['english']
                scores[handle] = english_score
                print(handle + ' ' + str(english_score))
            else:
                print('account ' + handle + ' skipped')
        # No tweets to analyze; skip and go to next handle
        except:
            print('Error: account ' + handle + ' skipped')
            continue
    with open('scores.txt', 'w') as file:
        file.write(json.dumps(scores))
    return scores

def get_bot_handles():
    # Read scores from file
    scores = {}
    bots = []
    with open('scores.txt', 'r') as file_obj:
        data = file_obj.read().replace('{', '').replace('}', '').replace('\"',
'').split(',')
        for line in data:
            handle, score = line.strip().split(':')
            scores[handle] = score.strip()
            if (Decimal(score) >= 0.43):
                bots.append(handle)
    outfile = open('bots.txt', 'w')
    for b in bots:
        # write line to output file
        outfile.write(b)
```

```
        outfile.write("\n")
    outfile.close()
    return bots

def get_bot_all_scores():
    accounts = []
    scores = {}
    # Read scores from file into list
    with open('bots.txt', 'r') as fileobj:
        for row in fileobj:
            row = row.rstrip('\n')
            accounts.append(row)

    # Get botometer scores for a list of accounts
    for handle, result in botometer.check_accounts_in(accounts):
        try:
            tweet = api.user_timeline(screen_name=handle, count=1, include_rts=True)[0]
            # If user profile language is english print english score
            if (tweet.lang == 'en' or tweet.lang == 'und' and tweet.possibly_sensitive
== False):
                score = result
                scores[handle] = score
                print(handle + ' ' + str(score))
            else:
                print('account ' + handle + ' skipped')
        # No tweets to analyze; skip and go to next handle
        except:
            print('Error: account ' + handle + ' skipped')
            continue
    with open('scores.txt', 'w') as file:
        file.write(json.dumps(scores))
    return scores

# Gets a set number of tweets for a list of bots
def get_tweets():
    # Removes URLs, handles, reserved words from tweets
    p.set_options(p.OPT.URL, p.OPT.MENTION, p.OPT.RESERVED)
    # Stores tweets and handles
    tweets_map = {}
    bad_chars = list(punctuation)
    bad_chars.remove('!')
    bad_chars.remove('.')
    bad_chars.remove('?')
    bad_chars.append('—')
    bad_chars.append('...')
    bad_chars.append('..')
    bad_chars.append('')

    with open('bots.txt', 'r') as file_obj:
        for row in file_obj:
```

```
            handle = row.strip()
            try:
                tweets = api.user_timeline(screen_name=handle, count=100,
tweet_mode='extended', include_rts=False)
                for tweet in tweets:
                    try:
                        if (tweet.lang == 'en' or tweet.lang == 'und' and
tweet.possibly_sensitve == False):
                            words = tweet.full_text.split(' ')
                            new_words = []
                            for wrd in words:
                                # Remove words where letters and numbers are mixed like
LX23: to do
                                # Remove monetary values $325 and numbers with decimals
in the middle
                                # Replaces THIIIIIIIIS with THIS
                                wrd = re.sub(r'\d+', '', wrd)
                                wrd = re.sub(re.compile(r'(.)\1{2,}'), r'\1', wrd)
                                new_words.append(wrd)

                            words = new_words
                            cleaned_tweet = ' '.join(words)

                            # Using replace() to remove bad_chars
                            cleaned_tweet = p.clean(cleaned_tweet)
                            for char in bad_chars:
                                cleaned_tweet = cleaned_tweet.replace(char, '')

                            if (cleaned_tweet != ""):
                                # Appending tweets to the empty array tmp
                                tweets_map.setdefault(handle, []).append(cleaned_tweet)
                                # Remove duplicate tweets
                                tweets_set = set(tweets_map[handle])
                                # Convert back to original format
                                tweets_map[handle] = list(tweets_set)

                                print(handle + " " + cleaned_tweet)
                        else:
                            continue
                    except tweepy.TweepError:

                        continue
            except:
                continue

    return tweets_map

# Removes bots that have less than 20 tweets in the dictionary
def clean_tweets_dict(this_map):
    for key in list(this_map):
```

```
        if (len(this_map[key]) < 50):
            this_map.pop(key)
    with open('tweets.pickle', 'wb') as f:
        pickle.dump(this_map, f)
    return this_map

this_map = clean_tweets_dict(tweets_map)

def sentiment_analysis(sentiment_values, num_tweets):
    tweet_count = 0
    print('SENTIMENT ANALYSIS: \t' + str(num_tweets) + ' tweets')
    # Load map from pickle file
    with open('tweets.pickle', 'rb') as f:
        result = pickle.load(f)

    print('num bots ' + str(len(result)))
    analyzer = SentimentIntensityAnalyzer()
    for key in result:
        print('num tweets ' + str(len(result[key])))
        tweet_count = tweet_count + len(result[key])
        user_sentiment = 0.0
        for tweet in itertools.islice(result[key], 0, num_tweets):
            vs = analyzer.polarity_scores(tweet)
            # Average scores over each user to find final score
            user_sentiment += vs['compound']
        avg_sentiment = round(user_sentiment / num_tweets, 4)
        # Add to map
        sentiment_values.setdefault(key, []).append(avg_sentiment)
    print('total tweets ' + str(tweet_count))
sentiment_values = {}
# Compute average sentiment over 20 tweets
sentiment_analysis(sentiment_values, 20)
print('sentiment values')
print(sentiment_values)
# Compute average sentiment over 30 tweets
sentiment_analysis(sentiment_values, 30)
# Compute average sentiment over 40 tweets
sentiment_analysis(sentiment_values, 40)
# Compute average sentiment over 50 tweets
sentiment_analysis(sentiment_values, 50)

positive_bots = []
negative_bots = []
neutral_bots = []
def classify_bots(sentiment_values, pos):
    if (pos == 0):
        value = 20
    elif (pos == 1):
        value = 30
    elif (pos == 2):
```

```
        value = 40
    else:
        value = 50
    print('\nCLASSIFY: \t' + str(value) + ' tweets')

    for handle, sent_arr in sentiment_values.items():
        # print(sent_arr)
        if (Decimal(sent_arr[pos]) >= 0.05):
            positive_bots.append(handle)
            print('positive' + handle)
        elif (Decimal(sent_arr[pos]) <= -0.05):
            negative_bots.append(handle)
            print('negative' + handle)
        else:
            print('neutral' + handle)
            neutral_bots.append(handle)

    print('POSITIVE BOTS: \t' + str(len(positive_bots)))
    print('NEGATIVE BOTS: \t' + str(len(negative_bots)))
    print('NEUTRAL BOTS: \t' + str(len(neutral_bots)))

# Classifies bots using the 1st value
classify_bots(sentiment_values, 0)
# Classifies bots using the 2nd value
classify_bots(sentiment_values, 1)
# Classifies bots using the 3rd value
classify_bots(sentiment_values, 2)
# Classifies bots using the 4th value
classify_bots(sentiment_values, 3)
```

Appendix B

```
def build_train_set(corpus_file):
   training_dataset = []
   with open(corpus_file, encoding='utf-8') as csvfile:
       line_reader = csv.reader(csvfile, delimiter=',')
       for row in line_reader:
           training_dataset.append({'tweet_id': row[0], 'label': row[1], 'topic':
row[2], 'text': row[3]})
   return training_dataset

def build_test_set(num_tweets):
   test_set = []
   with open('tweets.pickle', 'rb') as f:
       result = pickle.load(f)
   for key in result:
       user_sentiment = 0.0
       for tweet in itertools.islice(result[key], 0, num_tweets):
           tweet_dict = dict()
           tweet_dict['text'] = tweet
           tweet_dict['label'] = None
           tweet_dict['handle'] = key
           test_set.append(tweet_dict)
           if 'tweet_id' in tweet:
               print('tweet id')
               test_set.remove(tweet)

   return test_set

test_data_set = build_test_set(50)

class PreProcessTweets:
   def __init__(self):
       self._stopwords = set(stopwords.words('english') + list(punctuation) +
['AT_USER', 'URL'])

   def process_tweets(self, list_of_tweets):
       processed_tweets = []

       for tweet in list_of_tweets:
                           processed_tweets.append((self._process_tweet(tweet["text"]),
tweet["label"]))

       return processed_tweets

   def _process_tweet(self, tweet):
       tweet = tweet.lower()  # convert text to lower-case
       tweet = re.sub(' \d+', '', tweet)
```

```
        tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', 'URL', tweet)  # remove
URLs
        tweet = re.sub('@[^\s]+', 'AT_USER', tweet)  # remove usernames
        tweet = re.sub(r'#([^\s]+)', r'\1', tweet)  # remove the # in #hashtag
        tweet = word_tokenize(tweet)  # remove repeated characters (helloooooooo into
hello)
        return [word for word in tweet if word not in self._stopwords]

def build_vocabulary(preprocessed_training_data):
    print('build vocab')
    all_words = []
    for (words, sentiment) in preprocessed_training_data:
        all_words.extend(words)
    wordlist = nltk.FreqDist(all_words)
    word_feat = wordlist.keys()
    return word_feat

def extract_features(tweet):
    tweet_words = set(tweet)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in tweet_words)
    return features

def save_classifier(classifier):
    print('save classifier')
    file = open('sentiment_classifier.pickle', 'wb')
    pickle.dump(classifier, file, -1)
    file.close()

def load_classifier():
    print('load classifier')
    file = open('sentiment_classifier.pickle', 'rb')
    classifier = pickle.load(file)
    file.close()
    return classifier

tweetProcessor = PreProcessTweets()
preprocessedTestSet = tweetProcessor.process_tweets(test_data_set)
training_data = build_train_set('training_data.csv')
preprocessedTrainingSet = tweetProcessor.process_tweets(training_data)
word_features = build_vocabulary(preprocessedTrainingSet)

if os.path.isfile('sentiment_classifier.pickle'):
    print('pickle file exists')
    cl = load_classifier()
    for tweet in myTestSet:
        print(tweet)
    NBResultLabels = [cl.classify(extract_features(tweet)) for tweet in myTestSet]
```

```
   testFeatures = nltk.classify.apply_features(extract_features,
preprocessedTestSet[0:100])
   print(nltk.classify.accuracy(cl, testFeatures))
   cl.show_most_informative_features(5)
   print('classification done')
else:
   print('pickle file doesnt exist')
   trainingFeatures = nltk.classify.apply_features(extract_features,
preprocessedTrainingSet)
   NBayesClassifier = nltk.NaiveBayesClassifier.train(trainingFeatures)
   save_classifier(NBayesClassifier)
   NBResultLabels = [NBayesClassifier.classify(extract_features(tweet)) for tweet in
myTestSet]
```