Twitter Bot Detection

Colin McCahill

cmccahil@oberlin.edu

Shomya Mitra  
smitra@oberlin.edu

*Abstract*—This report takes a deeper look into how Twitter bots can be identified. It is no secret that they are a huge problem for Twitter, and there are several machine learning methods that Twitter uses to detect these bots. In this project we attempt to create our own solution for classifying Twitter data from bots. We use three different supervised learning algorithms and compare the results, after giving them datasets of bot and genuine account information. We evaluate the accuracy of the algorithms and explain the reasons behind certain results. We also compare these algorithms on one dataset to see which one is the most effective at detecting Twitter bots and compare these observations to results of past implementations and past projects.

# Introduction

Twitter and other social media platforms are littered with bots. They can be used to perform simple and repetitive tasks that otherwise would be time-consuming, dull, or impossible for actual humans. Sometimes it is repetitive to post content across social media, and it is easier to give the scheduling and posting to a bot. The Netflix bot, for example, automatically tweets when new content is added to the streaming service. The NFL 4th down bot posts live NFL analysis to every 4th down play in the NFL. Both of these bots are useful for analytic and metric reports.

Most bots, however, are fraudulent accounts that exist to spread spam and political agenda. If you look at the twitter account of any major political leader, especially the United States President, the tweet replies are often flooded with thousands of twitter bots tweeting redundant material. Replies and tweets are amplified with endless streams of tweets and retweets, often 1000 per day. Content is often not original material but retweeted and quote links. For some people it is easy to recognize a bot. Many people, however, are easily persuaded by bots that are tweeting political discourse. Traffic to news websites, sports websites, and most notably adult content websites can be drastically increased with bots repeatedly tweeting and retweeting the link to the website in the replies of major accounts. We can’t deny that bots have an enormous impact on Twitter, whether we think we’ve been personally affected by them or not. The goal of our project, and the goal of Twitter and other third parties is to figure out the best way to detect these fraudulent accounts so that they can be studied, and eventually eliminated.

In our solution, we evaluated a few popular supervised machine learning approaches. We use a Python’s scikit-learn Decision Tree and a Tensorflow Neural Network on a dataset filled with genuine accounts and bot accounts. The data set is actually a combination of three data sets: genuine\_accounts.csv, social\_spambots.csv, and traditional\_spambots.csv. We apply each algorithm to the data set, and after running the program with every attribute, alter the attributes fed into the algorithms to enhance the performance.

After running these two algorithms, the Decision Tree actually does significantly better than the Neural Network. The accuracy is almost 3 percent better, and due to the large dataset, the confidence intervals do not overlap. There are a few possible reasons for this. Overfitting, non optimal hyperparameters, and irrelevant attributes can negatively impact the accuracy of the neural network. We will investigate these reasons and the performance differences in the algorithms over the course of this report.

As a secondary goal of our solution, we also looked at classifying individual tweets as originating from bots or genuine accounts. For this solution, we used a data set consisting of 6.4 million tweets gathered by the same study we used for our Twitter account data. We used the tweet data to train a scikit-learn Naive Bayes algorithm to determine whether each tweet was generated by a bot or genuine account. This solution actually performed much better than we expected, considering the relatively primitive approach that Naive Bayes uses. The performance is also presented later in this report.

# Background & Related Work

To understand the problem, we must look at the political impact that bots can have. A Vanity Fair article, “Twitter’s bot problem still seems bad as ever”, gives us several examples where Bots can actually sway public sentiment. The article gives an example of racist rhetoric towards the caravan of Central American migrants, and that 60 percent of the conversation was driven by bots. Tweets from bot accounts can get picked up and rebroadcast by real people, including the president himself. From there, they make the jump from cable news, which is broadcast to millions of people. Russian troll tweets were cited in more than 100 U.K news articles [1].

The problem of identifying bots has been solved by supervised learning by only a few researchers. Based on an article from Fast Company, “These Students Built the Anti-Bot Algorithm Twitter Desperately Needs”, we observed tendencies of bot accounts. The students at UC Berkeley through their research were able to find similarities between bot accounts: “Most bots share certain tendencies: They tend to tweet regularly every few minutes, retweet fake news, obtain a large number of followers very quickly, and retweet similar accounts” [2]. From this statement alone we were able to figure out how to write our own algorithm on how to detect bots. We were going to look account attributes like number of followers, number of tweets, etc. The Berkeley students mentioned in the article, Bhat and Phadte, claim that bot-behaving accounts do respond and comment like human accounts and therefore looking at tweets was probably not the best way to approach this.

# Problem

In our experiment, we used a data set with labeled accounts and tweets of the following three types. Their distributions are illustrated in Table 1.

* Genuine Accounts: Verified, human-operated accounts.
* Social Spambots: Bots that tweeted political spam or promoted a product/company. Examples of these bots included retweeters of an Italian political candidate, spammers of a paid mobile device app, and accounts promoting products sold on Amazon. These accounts are designed to be similar to genuine accounts, since their goals are to influence real Twitter users’ behaviors instead of getting them to click on spam links.
* Traditional Spambots - Bots tweeting more malicious links than social spambots. Examples of these include simplistic bots that repeatedly mention other users in tweets containing scam URLs, or bots that repeatedly tweet about open job positions and job offers. These bots are easier to notice at a glance because they behave more like the usual bots that we’re used to.

Table 1: Distribution of Twitter Accounts Data Set

# Solution

## Neural Network

In order to classify Twitter accounts, we used a data set including metadata about each account, including Twitter ID, screen name, number of followers and following, cosmetics like background colors, time zone, and more. The first step to processing the data was to remove all attributes with a unique value for each account, like screen name, ID, profile image, and banner image. Once this information was removed, we could use it to train our algorithms correctly.

    Our first algorithm was a traditional neural network using Python’s Tensorflow library. Neural networks are a set of algorithms modeled loosely after the human brain that are designed to recognize patterns. They interpret sensory data, and then cluster and classify. The Tensorflow neural network is built out of individual neurons, each of which takes the attributes multiplied by weights as input, runs an activation function on these inputs, and returns a probability between 0 and 1. If in the output layer that probability is above a threshold for a certain label, the prediction will be that label. The neural network will then use Backpropagation to update the weights in the neurons depending if the prediction is right or wrong.

## Decision Tree

We also used the Python sklearn decision tree algorithm to classify accounts. The decision tree algorithm recursively trains by looking at all of the training data. If all of the attributes have the same label it returns a leaf node with that label. Otherwise it chooses the attribute that will best predict the label and creates a tree node for that attribute and subsequent tree nodes for each value of the attribute. The algorithm divides the training data into subsets depending on what value each instance has for the attribute. This is repeated until there are no attributes left, and a leaf node is available for all training instances. The training set will end up with 100% accuracy, and to create predictions in the test set, the algorithm simply follows the path down the tree corresponding to each attribute values of each test set instance.

## Naïve Bayes

To predict sources of individual tweets, we used a huge data set of labeled tweets, including metadata about them like when they were tweeted, like count, retweet count, and reply count, along with the actual text of the tweets. We were interested in seeing if we could predict a label based on only the content of /

|  |  |  |
| --- | --- | --- |
| Label | Proportion of Data Set | Count in Data Set |
| Genuine Account | 31.54% | 3474 |
| Social Spambot | 44.59% | 4912 |
| Traditional Spambot | 23.87% | 2630 |

?”> -/8each tweet. Thus, our first step was to omit all attributes other than the tweet content itself. The algorithm we decided to use to analyze the tweet content was Naive Bayes, since it is very effective at text classification. Naive Bayes works based on Bayes’ Theorem, which uses the prior probability of a label, the proportion of the label in the data set, along with a few more proportions found in a data set, to predict the posterior probability of the label, the likelihood of a label based on the data. This posterior probability is what we want to predict. In a nutshell, Naive Bayes uses the words in a text sample to calculate a probability for each possible label, and chooses the label with the highest probability as the prediction. What makes it “naive” is that to make the calculation practical, it assumes every attribute is independent of the others. This assumption is almost always incorrect but the algorithm still performs well regardless.

# Experimental Setup

In our account detection experiment, our performance measures were the overall accuracy, the recall for each label, and a confidence interval to compare which solutions were better. Our solution implementations came from Tensorflow for the Neural Network, and sklearn for the Decision Tree.

For training/test percentages, we used 60% for training, 20% for validation, and 20% for testing in the neural network. For us to be consistent, we decided to use 80% for the training and 20% for the testing in the decision tree. This way, the testing percentages were the same for both algorithms. The random seed was 12345 for both algorithms. The hyperparameters in the Neural Network were 10 hidden neurons, 1 hidden layer, and a 0.01 learning rate.

To classify individual tweets, our performance measures were similar to those of the account classification problem, though we were not aiming to compare different approaches. We decided on overall accuracy and recall of each label for our

Table 4: Attribute Changes After Preliminary Attempt

|  |  |
| --- | --- |
| Removed Attributes | Kept Attributes |
| * Name, screen name * Urls, image urls * Profile backgrounds, profile color info, profile text color * Timestamps of when the account was created and last updated | * Statuses count (total tweets) * Followers count * Following count * Favorites count * Notifications count * Language, time zone, location |

performance measures. For our experimental setup, we used the

scikit-learn Naive Bayes implementation, and Naive Bayes doesn’t have hyperparameters like those of neural networks. However, we did have to make decisions regarding our data preprocessing. We decided to remove punctuation and stem each word to ignore tenses and plurals. Since Naive Bayes looks at individual words and many tweets include links, we decided to replace all links with the word “link” so every individual link wouldn’t be treated like a unique word. We also decided to use tf-idf to convert text into usable data. Tf-idf stands for “term frequency-inverse document frequency,” and it takes into account a word’s frequency in a tweet in addition to the proportion of tweets containing that word in the entire data set.

# Results

## Account Classification

Table 2: Results with all 42 attributes used

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Accuracy (%) | Recall “Genuine account” (%) | Recall “Social Spambot” (%) | Recall “Traditional Spambot” (%) | Accuracy 95% Confidence Interval |
| Neural Network | 45.28 | 0.35 | 100 | 0 | [0.4342, 0.4714] |
| Decision Tree | N/A | N/A | N/A | N/A | N/A |

Table 3: Results with only 16 attributes used

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Accuracy (%) | Recall “Genuine account” (%) | Recall “Social Spambot” (%) | Recall “Traditional Spambot” (%) | Accuracy 95% Confidence Interval |
| Neural Network | 95.42 | 94.95 | 95.66 | 95.63 | [0.9455, 0.9629] |
| Decision Tree | 98.18 | 98.40% | 98.68 | 97.00 | [0.9762, 0.9874] |

When we began this project, we struggled to get both algorithms to work with all of the attributes. Initially, we tried giving every attribute to the neural network to see how it would perform. However, as shown in Table 2, it performed terribly. This was because many of the attributes had values unique to every account, and were nonsensically interpreted as categorical, which meant that one-hot encoding them gave us tens of thousands of attributes. After recognizing this issue, we selectively removed attributes from our data set. These attributes are outlined in Table 4.

Once the attribute problem was fixed, both algorithms ran smoothly. Table 3 shows their results after we filtered the

attributes, which were substantially improved. The recalls for each label were similar throughout the algorithms, and the

accuracy of the decision tree was in fact better than the accuracy of the neural network. The non-overlapping confidence intervals show that this difference in accuracy is statistically significant.

The reason for the two algorithms performing better after many of the attributes were removed is that many of the attributes were unique to each account, making them irrelevant to predicting whether an account was a bot. These attributes greatly slowed down the preprocessing without adding any effectiveness. In preprocessing, every categorical attribute needed to be one-hot encoded, and if an attribute had a different value for every single instance, it was a major problem. Further, the decision tree implementation did not accept missing values. Filling in values for every instance would have been time-consuming, and many of the columns that were missing values were irrelevant to whether an account was a bot. Instead of adding values, we decided to remove attributes that tended to be missing values while also not appearing to be helpful.

Why the decision tree outperformed the neural network is unclear. One possible reason is the neural network showed signs of overfitting. The accuracy after epoch 350 in the training set was 96.13%, compared to 95.45% after epoch 500. Another possible reason is non-optimal hyperparameters. The neural network may have done better with more or less neurons, a different threshold, more hidden layers, or a different learning rate. The last reason we believe the decision tree did better was the attributes. Irrelevant attributes could have had more of an impact on the neural network than it did the decision tree.

In addition to running the neural network and decision tree with 16 attributes, we decided to see how effective a neural network could be if only given seemingly irrelevant data—namely the colors the user specifies for their profile page. To do this, we used only the following attributes: profile text color, profile background color, profile link color, and profile sidebar fill color. Incredibly, the neural network mustered an accuracy of 86% even though it was given only color information. This is because the bots displayed a preference for a few specific colors. For example, the colors #0000FF, #131516, and #9AE4E8 were some of the ones commonly found in bots. However, when we added this color data to our previous 16 attributes and re-ran the network, it did not do statistically significantly better. These findings show how complicated it can be to choose what data to use for machine learning.

Despite the algorithms having different accuracy, the key takeaway is that bots can be accurately detected by looking at certain account metadata, instead of the tweets themselves. Information like total number of tweets, number of followers, following, and favorites are indicative enough when searching for bots. They tend to have a large number of followers, most of which are other bots, and they follow a similar number of accounts. They also tweet at a very high rate in shorter periods of time. Supervised learning algorithms can even detect different types of bot accounts accurately, and this can be useful when separating the malicious and fraudulent bots from those that are useful and benevolent.

## Tweet Classification

Table 5: Naive Bayes Tweet Classification Confusion Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Predicted Label | | |  | |
| Genuine account | Social spambot | Traditional spambot |
| 274887 | 8770 | 359 | Genuine account | Actual Label |
| 38027 | 307299 | 199 | Social spambot |
| 6645 | 1437 | 6536 | Traditional spambot |

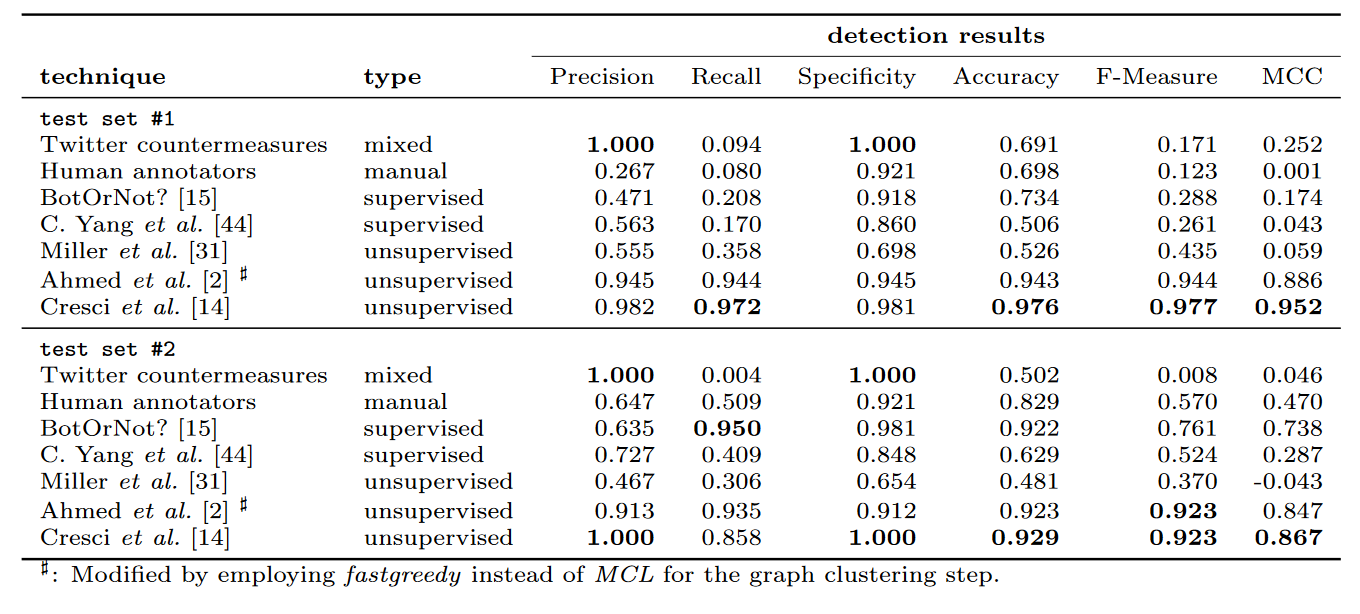
Table 6: General Results of Naive Bayes

Table 7: Performances of Existing Solutions on Our Dataset, from [3]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accuracy (%) | Recall “Genuine account” (%) | Recall “Social Spambot” (%) | Recall “Traditional Spambot” (%) | Accuracy 95% Confidence Interval |
| 91.394 | 96.786 | 88.937 | 44.712 | [0.9133, 0.9146] |

Our results from Tables 5 and 6 show the Naive Bayes model performed relatively well. We expected the algorithms being given account data to fare much better than the more rudimentary Naive Bayes approach. Instead, this version was only 3 percentage points below the neural network confidence interval, and 5 points below that of the decision tree. Naive Bayes performed very well when presented with a genuine account, posting a near-97% recall for those accounts. The large sample size of tweets used also allowed us a very tight confidence interval for the overall accuracy. This approach did not do as well when presented with a bot tweet, especially one by a traditional spambot. Naive Bayes actually incorrectly predicted genuine account for traditional spambots more often than it labeled them correctly. However, when presented with a bot of either type, this algorithm correctly predicted that it was either of the two types of bots 87.14% of the time.

There are a few possible explanations for Naive Bayes’ impressive performance. First, almost all bot tweets contained a link to some other website, which meant that if this algorithm was given a tweet without a link, it could safely assume it was genuine. Also, fake accounts tended to follow one of a few archetypes, such as advertising jobs or sharing articles. The text of these tweets probably used similar words compared to real tweets, which had a much wider variety of subjects. Notably there was one major flaw in the data which actually lead to a better accuracy. Since our data was gathered by an Italian team of researchers, a significant amount of the social spambot data was tweeted in Italian. There were some Italian tweets in our genuine account and traditional spambot tweets, but in general, if a tweet contained Italian words, it was a safe bet to predict it was tweeted by a social spambot. Another issue was that very little of the data set was traditional spambot tweets, which made it hard for the Naive Bayes model to learn their behaviors. Interestingly though, the traditional spambots produce tweets that are more obviously spam to human eyes, with clickbait titles meant to induce clicks from readers. It’s possible that the reason Naive Bayes mixes them up with genuine tweets is that they only contain plain English words and few proper names and hashtags, and real tweets include real sentences with real words much more than social spambot tweets.

    The fact that the confidence interval of Naive Bayes does not overlap with those of neural networks or decision trees imply that account metadata is statistically significantly better at finding Twitter bots. As seen in Table 8, our solutions had accuracy that matched or exceeded that of many existing solutions. We trained our models on this data, giving them an advantage, but if we could perform this accurately then there is likely some real improvement to be found in more complex methods.

# Conclusion and Future Work

In this project, we set out to detect Twitter bots using machine learning algorithms like decision trees, neural networks and Naive Bayes. Our approaches performed very well, both in classifying bot accounts and in classifying bot tweets. Most interestingly, our decision tree performed statistically significantly better than our neural network, despite neural networks being a much more complex algorithm. Considering how Naive Bayes performs on individual tweets, one possibility for future work is to apply it to many tweets from one account simultaneously. The results could be combined to become a better predictor than account attributes, like what our other algorithms were considering. Another more complex approach that could build off our findings would be to run Naive Bayes on an account’s tweets, then use the result as an extra attribute in the account data to input to a neural network or decision tree, like we did. We are very satisfied with the effectiveness of our solutions, but social media bots are an ever-increasing problem and work like this could lead to very effective solutions in the future.

##### References

1. M. Kosoff, “Twitter's Bot Problem Still Seems as Bad as Ever,” *Vanity Fair*, 01-Nov-2018. [Online]. Available: https://www.vanityfair.com/news/2018/11/twitters-bot-problem-still-seems-as-bad-as-ever. [Accessed: 03-Nov-2019].
2. K. Schwab, “These Students Built The Anti-Bot Algorithm Twitter Desperately Needs,” *Fast Company*, 10-Jul-2018. [Online]. Available: https://www.fastcompany.com/90149576/these-students-built-the-anti-bot-algorithm-twitter-desperately-needs. [Accessed: 20-Dec-2019].
3. S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi, “The Paradigm-Shift of Social Spambots,” *Proceedings of the 26th International Conference on World Wide Web Companion - WWW 17 Companion*, Apr. 2017