**Twitter User Classification**

**Eldin Sahbaz Raymond Hu** **Josiah MacDonald** **Omer Winrauke**

Syracuse University

{esahbaz, rahu, jhmacdon, owinrauk}@syr.edu

**0 Abstract**

Our goal with this study is to create machine learning models that are successful in determining whether a Twitter profile is run by a human or a bot. We were mostly interested in bots that were more difficult to classify than spambots. Being able to classify mid-spectrum bots is the first step in our goal of generalizing to bots used in large numbers to potentially influence political events. We had success with a number of machine learning models and achieved accuracy rates of above 97%.

**1 Introduction**

40 million tweets were posted by 10 pm on the 8th of November, the 2016 Election Day.[[1]](#footnote-1) Unsurprisingly, social media plays a crucial role throughout the election process – maximizing the diffusion of ideas. Seeing as social media is intended for individuals to share their thoughts with others, it is quite surprising to learn that Twitter bots are

responsible for producing 24% of tweets on Twitter.[[2]](#footnote-2) What’s even more surprising is that a study showed that, on Facebook, 80% of the bots are undetectable.[[3]](#footnote-3) Knowing this information, it may now be unsurprising that an estimated 400,000 bots were responsible for about 20% of election-related posts. The reason that this is worth mentioning is because these bot accounts become influential, in that these accounts aggregate large numbers of followers and are being retweeted by others.[[4]](#footnote-4) 4

Our ultimate goal is to successfully classify a Twitter profile as a bot or a human.

When defining our goals for this study, we decided to focus our work on more sophisticated bots than are generally studied. Spambots are typically easier to classify, posting large volumes of the same content, with lexical diversity often being the only necessary attribute to differentiate between human and bot. However, seen as this problem has already been solved, we wanted to focus on a more challenging subset of the bot population.



More specifically, we were interested in the bots currently being used in large numbers to influence public opinion, even having an effect on politics and elections.

These profiles are typically very difficult to identify by eye and even harder to classify. For starters, they have a lexicographical diversity that is much closer to a human's account. We also found that these bots appear to follow each other, resulting in more natural friend to follower ratios. Imbalanced friend to follow ratios are typically strong indicators that an account is a spambots. Furthermore, top tier bot accounts also have long lifespans with respect to the lifespans of spambots, since spambots typically get reported and shut down.

The two main contributions from this paper:

1. The bot/human classification algorithm we developed
2. The analysis and rationale for the features we use

**2 Previous Work**

Previous research in identifying Twitter bot accounts used similar approaches of classification through Twitter attributes but with different datasets. Other research also used datasets comprised of verified human accounts and fake Twitter bot profiles bought from a bot selling website while we focused on a harder dataset of sophisticated bots that emulated humans more than just a typical spam bot and unverified real human profiles. They narrowed the feature space either by assigning weights to each feature through the GAIN measure or through pattern analyzation from combinations of attributes that would best identify a core group of bots [1]. The research we looked into did not do any text processing of actual tweets which meant they lost out on a good chunk of data but used tweets indirectly through number of tweets .We used intuition to determine seven attributes that held the most potential for difference between bots and users such as lexical diversity. One interesting set of attributes came from “Fake Twitter Accounts: Profile characteristics obtained using an activity-based pattern detection approach,” by Supraja focused on more nominal attributes, we thought wouldn’t work as well since both users and bots used random location and names which makes it harder to differentiate the two [4].

One approach we did not use was graph-based detection used by Boshmaf in “Integro: Leveraging Victim Prediction for Robust Fake Account Detection in OSN,” where they considered the different online social media platforms used by people with Twitter being one of them to detect fake accounts through the number of social relationships between users as a graph [2]. We could have used other social media platforms by which the user is connected to on Twitter as another attribute to look into. Classification methods used by researchers included SVM, Random forest ,Decision Tree, Naive Bayes, and Neural Networks which got precision rates over 98% without bagging which is significantly higher than our rates given the types of bots they used over ours [1]. However, the results do match where random forest performs the best out of all the classifiers.

**3 DataSets**

In order to train our machine learning models, we had to gather a large collection of bot profiles and a large collection of human profiles.

**3.1 Human Profiles**

Gathering human profile was simpler than gathering bot profiles since each individual on the team simply collected all Twitter profiles from their personal networks. Initially, we believed we could use verified accounts to model real people, at least to some degree, but this ended up leading to some problems. We surprisingly found verified Twitter accounts that were bots. Typically verified bots are run by big companies (e.g. The New Yorker’s Poetry Bot). But possibly a more pressing issue is that there are trends among verified Twitter accounts that do not accurately represent the majority of human profiles. For example, when comparing to the majority of human Twitter profiles, the friend to follower ratio and tweet frequency are typically extraordinarily high. This disparity between the majority human Twitter population and the verified Twitter accounts would cause the data to be a misleading sample. Because of this, we resorted to manually collecting human profiles by collecting friends and friends of friends accounts’.

**3.2 Bot Profiles**

Gathering bot profiles was more complicated than collecting human Twitter profiles since the bots we were targeting would appear to be more human. Bulk gathering these profiles is not possible, given that there’s no real way to know with 100% certainty that a group of profiles belongs to bots. Our initial approach was to purchase Twitter bots and use them to train our model. However, this had a few issues. The bots we would purchase would typically resemble spambots, making it difficult for us to generalize to our target bots. Luckily, we came across lists of bots people maintain, allowing us to easily determine which accounts we wanted to train with:

* Botwiki lists and categorizes bot accounts
* Lists of bots on Twitter (e.g. botALLY), aggregating thousands of accounts

**4 Proposed Approach**

Our approach to differentiating between bot and human Twitter accounts can effectively be split into three stages:

1. Sourcing data from Twitter API
2. Cleaning data and extracting relevant features
3. Classification

**4.1 Data Collection**

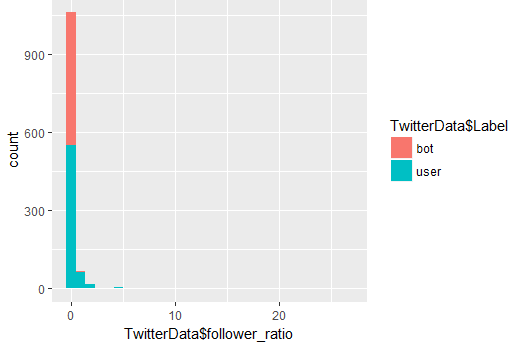
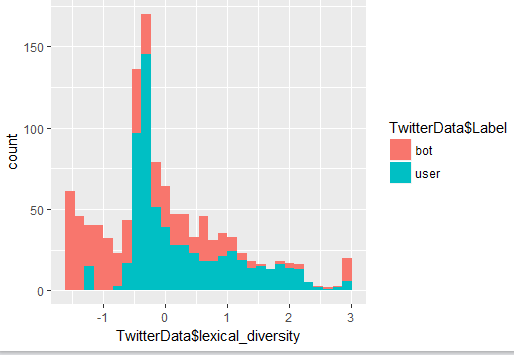
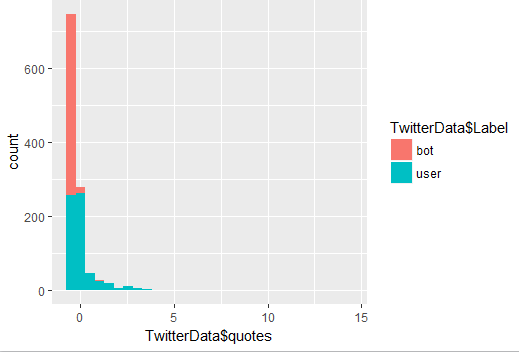
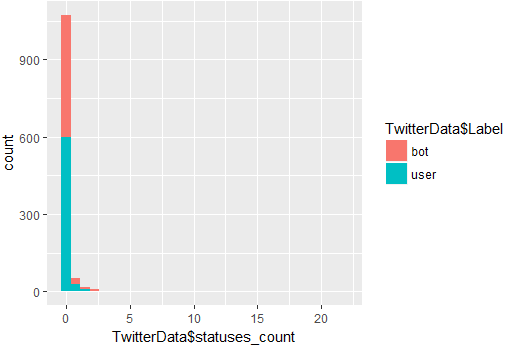
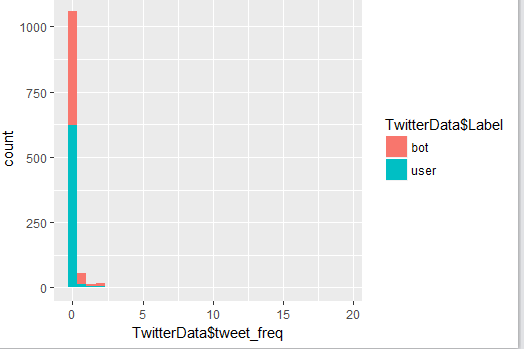
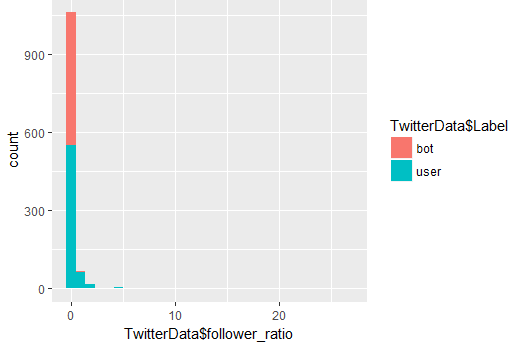
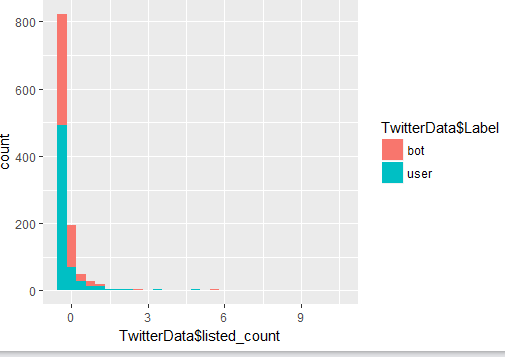
Given a set of user names, we iteratively visit each username and make two requests to the Twitter API to retrieve both the user’s profile and complete timeline. The profile contains information such as the screen name, number of tweets, location, profile picture, background, etc. The user timeline contains all the Tweets and their corresponding meta-data. Since our algorithm accepts a set of user names, it can also accept a single user name (important for how we hosted the script on our web application - section 6).

**4.2 Feature Extraction**

Now that we’ve sourced all the data pertaining to the user(s) we will be evaluating, the next step is to filter the data and distill it down to the features we will be using for classification. While extracting relevant features, we wanted to find the minimum feature set that would allow us to accurately differentiate between bots and humans while maintaining the expressive power to generalize to the top tier bots as well.

To be able to accurately distill the most important features from over 40 possible features, we first had to do some processing on the text data. The reason we need to do text filtering prior to any analysis is because Twitter allows users to include Emoticons in screen names, tweets, etc. Since using Emoticons requires extra text processing, we decided to remove them altogether in addition to removing all symbols - retaining only alphanumeric characters.

Now that we’ve cleaned the text data, we began the experimental process of finding the best features. However, in the process of searching for features that do work, we were able to identify several features that didn’t seem to work too well with our analysis. One of the features that we had trouble with was screen names. When we inspected the data of all the humans and all the bots in our dataset, we realized that screen names were just too unique to accurately quantify a difference between bots and humans. We faced a similar challenge with URLs.

 We initially thought it may be a good idea to document all the URLs tweeted by an account since we were working under the assumption that certain sites will appear more often among humans (such as CNN, FOX, NYT, etc.). However, bots can easily be programmed to source data from these sites as well. Although there are many unusual URLs from bots, URLs are just too unique to accurately separate bots and humans from each other. We could have created an incidence matrix from all the URLS, but this would be a very sparse matrix. This problem reoccurred when we tried incorporating sources into our analysis. The issue with sources is that many of the bots had very unique sources. In fact, the list of sources from bots had much more variance than the list of sources from humans. This doesn’t mean that bots and humans didn’t have overlapping sources, but many of the sources coming from bots were just too unique. We ran into a this problem again when attempting to use hashtags as features. The fact of the matter is, if we plan to generalize our algorithm to accurately identify bots in the top tier bracket, using hashtags to differentiate bots from humans may not be a good metric. The general purpose of bots is to influence their networks to conform to their creator’s ideology. Therefore, these bots will likely be using the same hashtags as humans. Furthermore, there are many hashtags that humans make up that bots likely will not use. Following this idea, we thought we could possibly discern bots from humans using profile location. However, we could not quite incorporate this attribute into our final feature set. While investigating the data, we noticed that many bots and humans had fake locations set in their profiles. If the location was not fake, many locations were misspelled, leaving the remaining subset too small to accurately separate bots and humans. We tried to recover the true location from misspelled locations by using an auto correction engine, however, the auto correction engine was unsuccessful in recovering the true location. Finally, we tried to incorporate the actual text data from each user’s tweets. Initially, we used each tweet as a document when building the TF-IDF vector. However, since each tweet has a maximum limit of 140 characters, the samples are not large enough to make accurate statistical inferences. Especially since after filtering the text data, it’s very likely many tweets were completely discarded (since they were entirely Emoticons). As a result, using each tweet as an individual document when building the TF-IDF vector and using an SVM, our classifier yielded 0% accuracy.

Since we still wanted use the text data, we derived lexical diversity from the tweets. When testing our algorithm we simply calculate number of unique words over number of total words. However, when training our classifiers, we had access to lots of data for both bots and humans. Since we wanted to avoid having outliers in our data, we replaced all lexical diversity of value zero with the average of the non-zero lexical diversities of their corresponding classes (i.e. replace bot zeros with average bot lexical diversity and human zeros with average human lexical diversity). The second derived feature is the friend to follower ratio. Since we wanted to find the minimum feature set, we decided to combine two features (friend count and follower count) into one feature by taking their ratios. The third and final derived feature is the tweet frequency. Since bots are just lines of code being executed, there must be some event that sets off the process that allows bots to tweet. Taking advantage of this inherent attribute of bots, we compute tweet frequency by computing the average number of tweets per day.

Now that we isolated the features that we will not use and derived some useful features, we can easily do some data exploration to find the most important attributes from the remaining set. Upon exploration, we found that the most informative features are lexical diversity, friend to follower ratio, replies count, quote count, listed count, statuses count, and tweet frequency. Empirically, these attributes seemed to give us the greatest separability between bot and humans. This can be seen when analyzing the bar graphs depicting the distributions for each feature.

**4.3 Classification Methods**

Our classification approach was to use the variety of classification methods provided by scikit-learn library in python to determine the best classifier that would most accurately determine if a Twitter profile was a bot or user. We used Random Forest, K-nearest neighbor, Naive Bayes, Multi-Layer Perceptron, Logistic Regression and SVM to classify our dataset and incorporated bagging which improved accuracy of all classifiers with the exception of naive Bayes and random forest. We took 50% of the features and 50% of the training data to train each bagging classifier. This pushed the all the classifier's accuracy rates above 85% but without bagging gave accuracy rates in the 60’s and 70’s for KNN, SVM, and Naive Bayes. For most of the classification methods, we used the default parameters provided by the library with a bit of fine tuning. Finally, we incorporated K-fold cross validation with K equal to 10 to obtain the average accuracy of the classifiers.

* Random forest classifier used 100 decision trees that used the gini index to determine the measure of impurity of a split.
* Decision tree classifier also used gini index to determine the measure of impurity of a split to determine the best attribute to split at. For example, in our decision tree model, we found that the best split was the “replies” attribute (Visualizations).
* K Nearest Neighbor classifier with K equal to 4 in conjunction with Euclidean metric to calculate distance and bagging to improve accuracy .
* The default Bernoulli Naive Bayes was used with Laplace smoothing to avoid the Zero probability problem and without bagging. The Naive Bayes method binarized the values of each feature provided since it only handles binary-valued feature vectors. Since we determined the features we are using by intuition , Naive Bayes is a method that would handle the irrelevant attributes that we had chosen if there was one. We also wanted a classifier which used independence assumption to assume all our features are independent from each other which mean each attribute is estimated in one dimensional space to see how well the classifier does compared to other classifiers [7]. This is a pro and a con since it avoids the curse of dimensionality that other classifiers might run into but it doesn’t consider the correlation between the attributes such as the correlation between tweet frequency and lexical diversity.
* MLP classifier using L-BFGS as the solver for weight optimization with bagging to improve accuracy. We set the seed for the random number generator to be 1 and (1e - 5) for the “L2 penalty (regularization term) parameter,” [5] . Lastly the parameters for hidden\_layer\_sizes we set as (7,4) to fit our 7 features and to have a 2-hidden layer neural network.
* Default linear regression with 1e5 for the inverse of the regularization strength with bagging to improve accuracy.
* Linear support vector classifier that used square of the hinge loss as the loss function and bagging to improve accuracy. We used a linear version because it works better with a larger sample size as compared to the regular support vector machine.

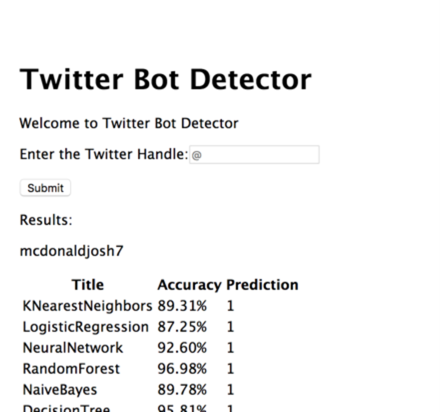
**5 Results & Findings**

We began with over 40 possible features, we found that the most informative features are lexical diversity, friend to follower ratio, replies count, quote count, statuses count, and tweet frequency. The fact that these attributes are the most informative was expected.

We expected lexical diversity to be the among the most informative features because there are generally differences between bot tweets and human tweets that can be recognized when inspecting lexical diversity. Many bots will tweet the same string of text with minor variations or they will simply tweet URLs. This sets an upper bound on the lexical diversity for these bots, however there are exceptions to this. There are generative bots that will create poems or create images. The bots that generate images generally fall under the first case we discussed since they typically do not generate much text. Conversely, bots that generate poems will have higher lexical diversity. But, the fact of the matter is that bots with higher lexical diversity are less frequently seen than humans with higher lexical diversity. We can easily observe this from the bar graph of bot and user lexical diversities. Likewise, the friend to follower ratio, replies count, quote count, and listed count were all very informative. These attributes quantify how likely these accounts are to interact with their respective networks. We can easily see from the replies count and quotes count graphs that it’s more likely that a human will engage in transactions with their networks. We expected that statuses count and tweet frequency would also quantify correspondence between an account and its network, but it turns out that these two phenomena are independent. Statuses count and tweet frequency merely quantify how active an account is.

Some important findings regarding the classifiers we used also surfaced throughout our experiments. We noticed that naive bayes and random forest perform worse with bagging than they do without bagging. Bagging increases the predictive error for naive bayes and it doesn’t do much to reduce variance for naive bayes because it is already a low variance classifier [3]. Furthermore, since bagging only takes a sample of the training set to train each naive bayes classifier this contributes to predictive error since the classifier doesn’t have all the prior knowledge it needs of actual probabilities per attribute to determine if a profile is a bot or human. Random forest also performed worse with bagging because bagging it allows different classifiers to have the same subsets of features which removes the key trait of independence from Random forest. This reduces the predictive performance slightly. Support vector machine performed the worst because it had difficulty finding the best hyperplane to separate the data points given there were seven attributes and data points might have become sparse when finding such a cut in higher dimensions.

**6 Web Service**

In order to run our service and open it to the public, we turned to Amazon Web Services (AWS). Even though the bulk of the project (including Twitter API interaction and model training / classification) was completed in python, we decided to use Node to host the site due to our familiarity with the framework and past experience with it. 

Luckily, the bridge from the Javascript in Node to python was not too complicated. We utilized a shell npm package that allowed us to run custom shell commands directly from Javascript. Since the python script accepted Twitter handles through the command line arguments, it was as simple as running the command:

In order to get the relevant data back to the Node environment, we output the relevant data through the stdout as a JSON string.

For the sake of simplicity, we utilized Pug (formally Jade) for the frontend, which allowed us to create the entire front page (albeit a very simple one) in relatively few lines. It consisted of 2 parts: an input section at the top, where the user could input a Twitter handle, and a table at the bottom that showed the prediction and accuracy for different classification models for the handle. To the right you’ll find a screenshot of our website with the output as correctly classifying @mcdonaldjosh7 as human.

**7 Future Work**

There is still much work that could be done to further improve this algorithm. Since this project can effectively be split into three sub-steps, it follows naturally to analyze the improvements that can be achieved in each of these sub-steps.

The first step is sourcing training/testing data. Since we would like to see how our algorithm behaves in a multitude of environments, we intend to collect more and different data. In the introduction we presented a scale depicted how bots have various complexities. As we already stated, for this study we decided to focus on middle tier bots. However, since we did not train our classifiers on data containing top tier and low tier bots, we cannot accurately classify those accounts. Collecting low tier bots is fairly simple, since these bots may easily be purchased on the internet for a small fee. The real difficulty in improving our data will lie in collecting top tier bots. These bots are purposely disguised to behave like humans, since their objective is generally to influence some cause. Since bots are generally going to be focused on some theme, we plan to perform our own Turing tests on tweets coming from accounts tweeting about trending topics. However, this only pertains to finding accounts and collecting their screen names. The other aspect of data collection is actually sourcing the data for each of these users. As mentioned in the previous sections, we collect data from Twitter via the Twitter API. However, when there is a large volume of data to source from Twitter, data collection can take a lot of time. To make our algorithm more efficient at data collection, we plan on taking advantage of the multiple cores on our systems by multi-threading the process.

The second phase is to perform feature extraction on the data we previously collected. This is where we plan on investing majority of our time. As discussed in the previous sections, we attempted to include text data into our classification process, but were not successful in obtaining accurate results. Moving forward, rather than using each tweet as a document for the TF-IDF vector, since 140 characters is too small a sample, we plan on grouping tweets over some specified period of time and using those grouped tweets as documents for the TF-IDF vector. Similarly, we intend to make use of sources and hashtags. For both sources and hashtags, we can create a new binary variable for each of the nominal states.

This allows us to define the above distance metric that more accurately depicts how distinct/similar bots and humans in their use of sources/hashtags. Also, in an effort to improve our ability to generalize to different bots, we plan to incorporate media data into our algorithm. More specifically, we would like to utilize the neural network framework to convert image data to text data. Researchers at Stanford University have made use of the CNN and RNN framework to caption images[[5]](#footnote-5). With this text data, we can process the captions similarly to how we plan to process tweets. The next feature we would like to incorporate into our algorithm is Tweet randomness. We plan to build a probability distribution of how likely bots and humans are to tweet given date and time stamps. This will allow us better quantify tweeting patterns. This Tweet randomness attribute will likely replace the current Tweet frequency feature. The last feature we would like to include in our algorithm is node centrality. As we discussed before, bots generally operate to influence their networks in some way. Therefore, we would like to analyze each account’s network of friends and followers to retrieve their corresponding node centrality values. This social media mining technique will assess if a bot or human is truly trying to influence its network. However, not all of these features should be equally weighted. Some features may be more influential in determining if an account is a human or a bot. Therefore, we plan on incorporating information gain/PCA into our algorithm to assign appropriate weights to each feature.

The third step in our algorithm is to perform classification on features we extract in step two. As we discussed in previous sections, we used the Random Forest, Decision Tree, Logistic Regression, Multilayer Perceptron, Support Vector Machine, K Nearest Neighbor, and NaÏve Bayes classifiers. However, we would also like to include a deep neural network since the Multilayer Perceptron used in our algorithm only has four layers. We believe that the added layers will help us differentiate between bots and humans.

**8 Conclusion**

In this paper, we proposed an algorithm to identify which Twitter accounts belong to bots and which Twitter accounts belong to humans. The attributes used in our study have been collected via empirical studies. We have conducted experiments to reach the minimum set of attributes to extract the best results from our classifiers. From a set of more than 40 possible attributes, we were able to reduce the feature space down to just seven attributes. Although these attributes are effective when it comes to classifying mid-tier bots, we plan to extend our feature set to allow our classifier to capture more complex bot accounts. To do this, we also need to prepare a dataset that better represents the bot population on Twitter - incorporating bots from each of the three tiers. The two main contributions from this paper. The first contribution is the bot/human classification algorithm we developed. The second contribution from this paper is the analysis and rationale for the features we use. Extending this point, we also contribute some possible features that we suspect will increase accuracy and better handle the bot population as a whole. All things considered, our algorithm’s performance meets the expectations we had at the set out of this project.

**9 References**

[1] Ahmed, Elaza, Mahmood A. Mahmood, and Hefny A. Hesham. "Fake Account Detection in Twitter Based on Minimum Weighted Feature set." ResearchGate. January 2016. <https://www.researchgate.net/publication/304569053_Fake_Account_Detection_in_Twitter_Based_on_Minimum_Weighted_Feature_set>.

[2] Boshmaf, Yazan, Dionysios Logothetis, Georgos Siganos, Jorge Leria, Jose Lorenzo, Matei Ripeanu, and Konstantin Beznosov. "Integro: Leveraging Victim Prediction for Robust Fake Account Detection in OSNs." *Proceedings 2015 Network and Distributed System Security Symposium*, 2015.

[3] Davidson, Ian. "An Ensemble Technique for Stable Learners with Performance Bounds." http://web.cs.ucdavis.edu/~davidson/Publications/aaai2004.pdf.

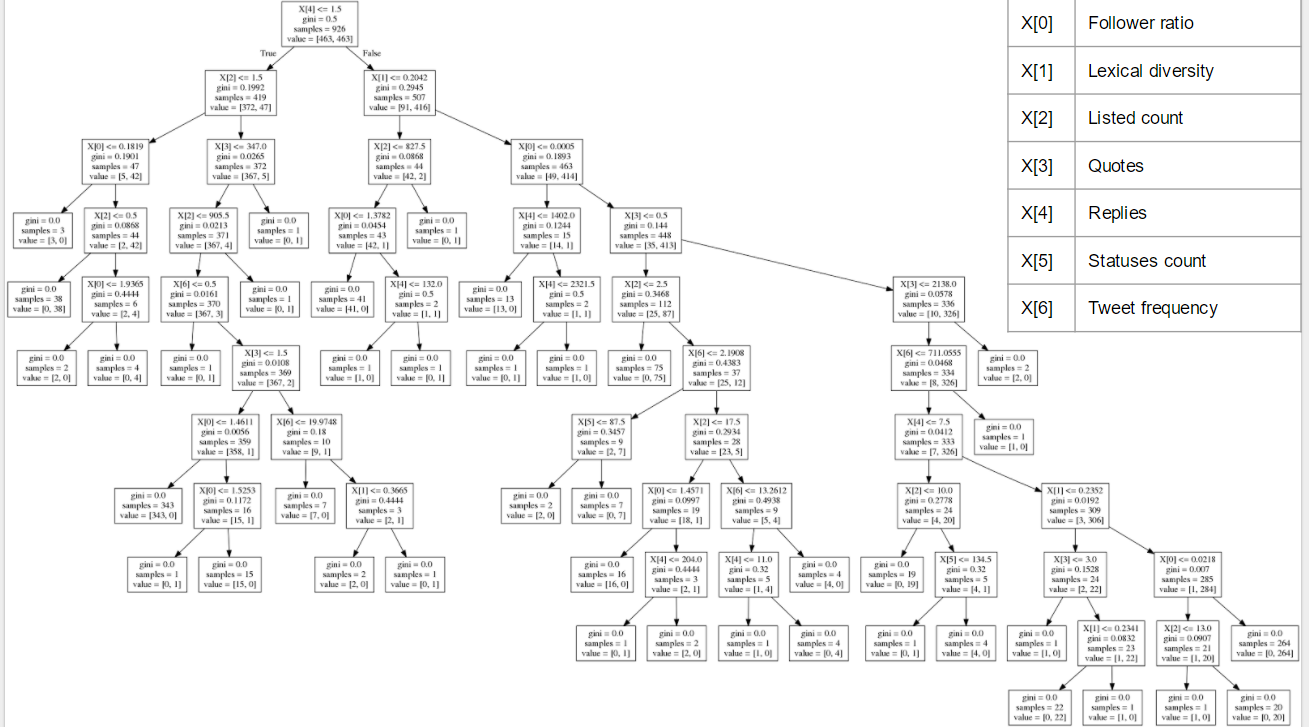
[4] Supraja , Gurajala, Joshua White, Jeanna Matthews, and Brian Hudson. "Fake Twitter accounts: Profile characteristics obtained using an activity-based pattern detection approach." ResearchGate. July 2015. https://www.researchgate.net/publication/280782550\_Fake\_Twitter\_accounts\_Profile\_characteristics\_obtained\_using\_an\_activity-based\_pattern\_detection\_approach.

[5] "Sklearn.neural\_network.MLPClassifier." Sklearn.neural\_network.MLPClassifier — scikit-learn 0.18.1 documentation. http://scikit-learn.org/stable/modules/generated/sklearn.neural\_network.MLPClassifier.html.

[6] "1.6. Nearest Neighbor." 1.6. Nearest Neighbors — scikit-learn 0.18.1 documentation. http://scikit-learn.org/stable/modules/neighbors.html#neighbors.

[7] "1.9. Naive Bayes" 1.9. Naive Bayes — scikit-learn 0.18.1 documentation. <http://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes>

**10 Visualizations**



1. https://www.nytimes.com/2016/11/09/technology/for-election-day-chatter-twitter-ruled-social-media.html?\_r=0 [↑](#footnote-ref-1)
2. http://www.public.asu.edu/~fmorstat/paperpdfs/asonam16.pdf [↑](#footnote-ref-2)
3. http://jpdickerson.com/pubs/dickerson14using.pdf [↑](#footnote-ref-3)
4. https://www.technologyreview.com/s/602817/how-the-bot-y-politic-influenced-this-election/ [↑](#footnote-ref-4)
5. http://cs.stanford.edu/people/karpathy/deepimagesent/ [↑](#footnote-ref-5)