Random Forest Twitter Bot Classifier

James Schnebly
Computer Science and Engineering
University of Nevada, Reno
Reno, Nevada 89557
Email: jschnebly@nevada.unr.edu

Shamik Sengupta
Computer Science and Engineering
University of Nevada, Reno
Reno, Nevada 89557
Email:ssengupta@unr.edu

***Abstract*—As social media usage continues to grow, so does the number of automated bot accounts that either spread malicious content or generate fraudulent popularity for political and social figures. Twitter, one of the more popular social media sites, has been plagued by bot armies and needs to find a way to rid itself of these infestations. Studies have shown that currently there are no accurate ways to detect Twitter bot accounts regularly. Using datasets from IIT (Institute of Informatics and Telematics), a feature set is created that allows a classifier to be both accurate and generalized. This feature set includes accessible features from the Twitter API as well as derivative ratio features that give a different perspective of the account. We then decide upon the Random Forest machine learning algorithm because of its ability to prevent most overfitting as well as create a generalized model that can be deployed for accurate use directly after training. In this paper, we propose a set of attributes for a Random Forest classifier that results in high accuracy (90.25%) and generalizability. To prove our derived feature set outperforms basic feature sets and grants valuable insight, we test our derived features against the most important of the basic features. Our derived ratio features outperform these basic account features.**

***Index Terms*—Twitter bot, machine learning, classifier, Random Forest, Scikit-learn.**

I. Introduction

Social media usage is growing faster than ever in today's world, according to PEW Research Center, 73% of U.S. adults use YouTube, 68% use Facebook and 35% use Instagram. Shortly behind these social media "staples", Twitter has quickly risen to the point at which one in every four U.S. adults uses its platform [1]. Twitter launched in 2006 and has not stopped growing since it went public. On average, around 6,000 tweets are posted every second of the day, adding up to roughly 360,000 tweets per minute [2].

With the huge growth of social media, it is not a stretch to see why many businesses spend time creating and maintaining various social media accounts. Digital marketing has allowed companies to harness the power of social media which, in turn, brings more customers to their businesses. Another key component of these corporate social media accounts is customer relations. When customers have a problem with a product or service they often go to social media to post about it. These customer relation accounts can seek out any posts about issues and attempt to resolve the matter through refund or product replacement.

A growing issue with Twitter is the amount of fake or "bot" accounts either sharing malicious content or being used to enhance the metaphorical reach of genuine accounts. These Twitter bots are easily programmed due to the Twitter API (Application Programming Interface) and can be created within seconds. This means that just about anybody can create bots to form the illusion of popularity around hot topics or political figures. To make matters worse, Twitter does not use a "captcha" when users are creating an account, therefore it is very simple to have an automated service pump out tons of bots that are then run through the open Twitter API [4]. A captcha is a program used to verify that a human, rather than a computer, is entering data on an online form [5].

Twitter bots have the ability to alter trending topics by the sheer volume of tweets they can produce. They can make false stories appear to have more views by promoting or tweeting about a certain topic along with the associated hashtag. These bots also spell disaster for corporations and their social media presence. One person with a massive army of bot accounts could have the bots all tweet negative sentiment about a company's product or service and tag the company in the tweet. Now anyone searching this product, service or company will be fed false information due to the shear volume of tweets sent out by the attacking bots.

Now more than ever we need a way to get rid of existing bots to protect people from being fed false information. In this paper, we present a simple yet effective classifier model that has the ability to detect existing Twitter bot accounts using only data that is easily attainable through the Twitter API as well as attributes that are ratios of the previously mentioned data. Our contributions in this paper are:

- A generalized machine learning model that can detect existing Twitter bot accounts with 90.25% accuracy
- A feature set that allows for high accuracy that includes both basic and derivative attributes

We start by obtaining data and importing it into two MySQL databases. Then, we preprocess the data by querying the MySQL databases and begin our feature selection and feature creation processes. The preprocessing python program outputs CSV (Comma-Separated Values) files which hold the input data and labels that are to be fed into our machine learning model. Lastly, we feed the CSVs into our python program that trains and tests a Random Forest classifier. This workflow can be seen in Figure 1.

The remainder of this paper is organized as follows: Section II will dive into the process and methodology of building

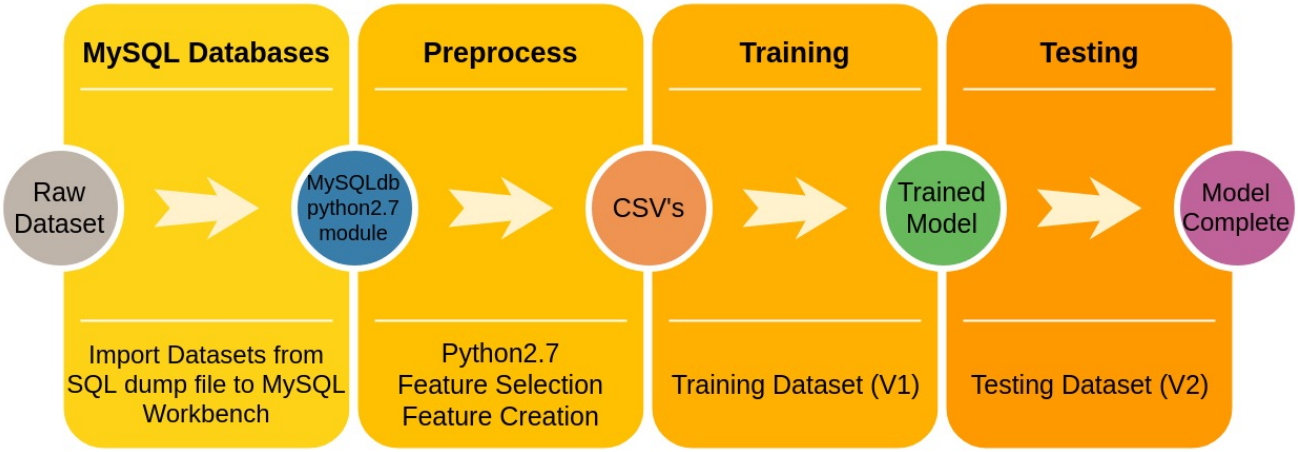


Fig. 1: Workflow of creating the Random Forest model 3.0

and training the classifier, Section III will cover the results and the testing of the previously mentioned classifier and Section IV will review and conclude the paper.

II. Methodology

A. Data Gathering

To train a Twitter bot classifier, we first need access to Twitter account data for both genuine accounts and bot accounts. Additionally, we also need access to as many tweets per account as possible. This task proves to be tricky as there are limitations to the free Twitter API [6]. Furthermore, there is no way to label an account as genuine or bot if we are just querying the API for data. Two datasets from the MIB (My Information Bubble) project [7] [8] hosted at IIT in Italy are used for our research purposes.

The first dataset was gathered in 2015 and contains verified human accounts as well as known bot accounts used to falsely inflate the number of followers for a genuine account. The second dataset was gathered in 2017 with new known bot accounts and verified genuine accounts.

The two datasets contain basic account information such as, but not limited to: number of followers, number of tweets, when the account was created and if the account was genuine or a bot. Along with the account information there are also hundreds, sometimes thousands, of tweets that belong to each account.

For our research purposes we use a subset of each dataset as the data is somewhat repetitive. Dataset V1 contains 445 real accounts and 3202 bot accounts from the original 2017 dataset and V2 which contains 1946 real accounts and 3457 bot accounts from the original 2015 dataset. These datasets V1 and V2 are then imported into a MySQL [9] database to make the preprocessing stage smoother. The data is imported into two databases, V1 and V2 where each database has two tables, namely Users and Tweets. The primary key shared between these two tables is the user_id which is a unique identifier given to every Twitter account.

B. Dataset Preprocessing

We select basic account attributes from the database table Users that could have an impact on the account being genuine or fake. These attributes are length of description (bio), age of account, number of followers, number of tweets, number of people the account is following (friends), number of likes and the follower to friend ratio. These attributes are chosen because they explain the account usage at a fundamental level. The basic feature subset can be seen in Table I.

TABLE I: Basic Feature Set

Length of Bio
Followers
Age of account
Following
Number of tweets
Number of likes
Follower_ratio

After the basic attributes are chosen, we create derived attributes by computing ratios. These derived attributes allow us to gain more insight on the activity of the account. These are derived features because they are not directly available through the Twitter API.

1) Likes_age_ratio: First we create the likes to age ratio, where we divide the number of likes the account has given by the number of days since the account's creation. This feature tells us how actively the account likes another account's tweet.

2) Tweet_age_ratio: The tweets to age ratio is created by dividing the number of tweets by the days since the account has been created. This attribute tells us how often the account tweets. We now analyze the account's tweets by performing an inner join on the user_id between the User table and Tweets table.

3) Hashtag_tweet_ratio: The hashtag to tweet ratio is computed by counting all tweets in our sample that contain at least one hashtag and dividing it by the total number of tweets in our given sample. The total number of tweets in the sample is given by using the SQL aggregate COUNT() to count the

number of tweets associated with a given user_id. This new ratio attribute gives insight on how often the account uses hashtags in their tweets.

4) URL/Pics_tweet_ratio: The url to tweet ratio is computed similarly. We use COUNT() to tally up all the tweets in the sample that contain urls and divide that integer by the total number of tweets in the sample for that particular user_id. Interestingly enough, when accounts post pictures on twitter the pictures are represented as urls, therefore this url to tweet attribute accounts for links to websites as well as pictures. This ratio tells us how often the account posts urls or pictures when they tweet.

5) Reply_tweet_ratio: The reply to tweet ratio is computed the same way using the COUNT() aggregate, and it tells us the approximate percentage of tweets posted that are replys to other accounts. This percentage is only based on the sample of tweets we have available, but provides valuable insight on the account's overall tendencies. The derived feature subset can be seen in Table II.

TABLE II: Derived Feature Set

Hashtag_tweet_ratio
URL/Pics_tweet_ratio
Reply_tweet_ratio
Tweet_age_ratio
Likes_age_ratio

The full proposed feature set can be seen in Table III along with a description of each attribute.

TABLE III: Proposed Feature Set

Length of Bio	Number of characters in bio
Hashtag_tweet_ratio	Ratio of tweets with hashtags to total tweets
Followers	Number of accounts following
URL/Pics_tweet_ratio	Ratio of tweets with URL's to total tweets
Age of account	Number of days since account creation
Reply_tweet_ratio	Ratio of replies to total tweets
Following	Number of accounts the account is following
Number of tweets	Number of tweets on the account
Tweet_age_ratio	Ratio of tweets to days since creation
Likes_age_ratio	Ratio of likes to days since creation
Number of likes	Number of likes on the account
Follower_ratio	Ratio of followers to following

Once all of these attributes are queried from the SQL database or created by the above methods, we create a CSV file that serves as input for our machine learning classifier. One CSV is created for genuine accounts and another is created for known bot accounts since the information is held in separate databases.

C. Random Forest Model Background

Before going into model implementation, a brief background of supervised learning and the Random Forest algorithm is needed.

1) Supervised Learning: Supervised learning is a subsection of machine learning where the user feeds the model training data which contains labels. The model learns to classify or predict the labels based on the training data that is associated with it. Once training is complete, the model uses its prior knowledge of the training data to predict labels on new data or test data. We use supervised learning, as opposed to unsupervised learning, because we want to train a classifier on existing labeled examples. Unsupervised learning makes decisions based on the relationship between the data instance at hand and the rest of the dataset. The unsupervised learning algorithm does not have any prior information about the correct class the data instance should belong to. Since we have data to shape our classifier, we chose supervised learning. There are many supervised machine learning algorithms, but for this particular classification problem we use the Random Forest algorithm.

2) Random Forest: A Random Forest is a meta estimator that fits a number of Decision Tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting [10]. The Random Forest algorithm is an ensemble learning algorithm, meaning the algorithm is actually made up of many other basic machine learning algorithms. To predict a class, each basic machine algorithm votes for a class and after all of the basic algorithms have voted, the class with the most votes is the class the ensemble algorithm predicts. With Random Forests, the underlying algorithm used is the Decision Tree classifier, hence the "Forest" in Random Forest. The Random Forest algorithm brings randomness into the model when it is growing the Decision Trees. Instead of searching for the best attribute while splitting a node, or decision in the tree, it searches for the best attribute among a random subset of attributes. This process creates diversity among trees and allows for each tree to be built upon different attributes, which generally results in a better model [11]. We choose the Random Forest algorithm because of its general effectiveness when classifying numerical inputs.

D. Random Forest Implementation

To implement the Random Forest classifier, we use python3.5 as well as the libraries Scikit-Learn, pandas, and numpy [12]–[14]. First, we read the CSV's for the genuine accounts and the bot accounts into pandas dataframes. Then, using numpy, we add the label column to each dataframe, namely "Fake_Or_Real". The genuine accounts get a '0' in this column while the bot accounts receive a '1'. Next, we combine the two dataframes into one that is then split it into the input attributes, X, and the output labels, y. Before training the model, we must create a train set and test set.

Since we have two completely separate datasets as discussed in II-A, we create three different models each using the data differently. Model 1.0 uses V1 for both the training and testing. The data is randomly split with 75% going to training while the remaining 25% is kept for testing. Model 2.0 combines both V1 and V2 and then does the same 75%-25% split for training and testing. Model 3.0 uses the entire V1 dataset for training and then tests on the entire V2.

We use the Scikit-learn RandomForestClassifier class and specify the forest to be made up of 20 Decision Trees [12].

20 Decision Trees is decided upon after tests of 10, 20, and 50 trees. The accuracy increase from 10 to 20 was significant while the accuracy increase from 20 to 50 was very minimal, thus leaving 20 as the optimal amount. We also specify that we want the trees to make decisions based on entropy by making the nodes, or decisions, yield more binary splits. The more binary the split, the higher the entropy. To train the models, we feed the twelve attributes X, discussed in Section II-B, as well as the output label y, into the model as training input and associated output.

III. Results

A. Experiment Environment

The training and testing of the models presented in this paper were trained and tested on a machine with the specifications from Table IV.

TABLE IV: Machine Specifications

Processor:	Intel Core i7-7700 CPU @ 3.60GHz x 8
RAM:	16 GiB
OS:	Ubuntu 16.04 LTS
Environment:	python3.5 in Spyder3

B. Model Results

Model 1.0 is tested with 25% of V1 and averaged 99% accuracy over the course of 10 runs. Model 2.0 is tested with 25% of V2 and averaged 98% accuracy over the course of 10 runs. Because of the high accuracy, we look to the decision trees produced in the random forest to see if there is any potential over-fitting in these two models. The decision trees give a visualization of each decision node and the size of the tree. The larger the tree, the more fit to the training data the model is. The goal is to have trees that can accurately classify instances with as few decision nodes as possible. The tree needs to be smaller, or generalized, in order to be useful in a real world situation. The larger, less-generalized trees may perform well on the training data but poorly on real world data because it is too tightly fit to the training data.

As seen in Figure 2a, the tree from model 1.0 is smaller and cleaner than the tree from model 2.0, seen in Figure 3. The tree from model 1.0 has less decision nodes and the splits are more binary, which means higher entropy. Since model 1.0 is more suitable for generalized usage we use its foundation to create model 3.0, previously discussed in section II-D. Model 3.0 is trained and tested on completely different datasets therefore giving a more clear picture of how this model would work if deployed onto Twitter right away. Over the course of 10 runs the model produced an average accuracy of 90.25%. Figure 2b shows a Decision Tree from model 3.0; note the similarity between the tree from model 1.0 and model 3.0. Both models are more general than model 2.0. The purpose of Figures 2 and 3 are to display the overall generalizability of each model. It should also be noted that the tree numbers were chosen arbitrarily and that any other tree could have been chosen to display the generalizability of each model. Models 1.0 and 3.0 are generalized compared to the overfit tree from model 2.0. This similarity between models 1.0 and 3.0 makes sense because they are trained on the same dataset. Model 1.0 is trained on a subset of V1 and tested on a different subset of V1 while model 3.0 is trained on all of V1 and tested on V2. Model 3.0 shows better real world feasibility than model 1.0 because it achieved the 90.25% accuracy on data from a different dataset. Model 1.0 was tested on a subset of data from the same dataset as its training data, therefore its accuracy cannot be transferred to a real world situation.

To evaluate accuracy and generalization of the models, a ratio of average decision nodes per tree to test accuracy is computed. Since model 3.0 is essentially a more feasible version of model 1.0, a comparison of this ratio between model 2.0 and model 3.0 is shown in Figure 4. Model 2.0 has, on average, 156 decision nodes per tree with an accuracy of 98%. Model 3.0 averages 43 decision nodes per tree and achieves an accuracy of 90.25%. This gives model 2.0 a ratio of .628 and model 3.0 a ratio of 2.09. These ratios show that although model 2.0 has a higher accuracy, it is not nearly as generalized as model 3.0. Model 2.0 may have a higher accuracy by 7.25% but the average amount of nodes per tree is more than triple of the average nodes per tree in model 3.0.

C. Feature Results

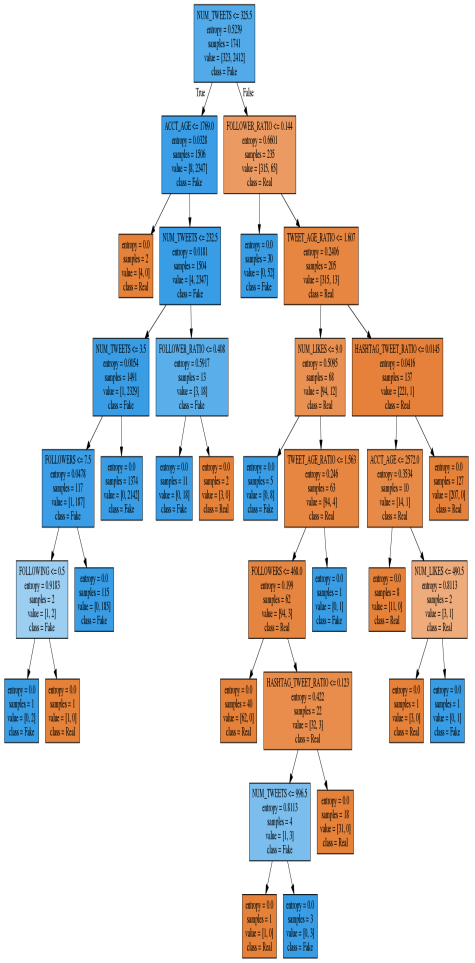
Since our contribution in this paper is not only the model, but also the features used to build it, we now look at feature importance to show our new features make a positive difference in classifier accuracy. All test runs to show feature importance are done with model 3.0 as it is the best, most generalized model of the three discussed in section III-B. Table V shows the feature importance for each attribute averaged over the same ten runs that model 3.0 achieved the 90.25% accuracy. Note the feature importance was calculated by the Scikit-learn [15] feature importance built-in method.

TABLE V: Feature Importance of Proposed Feature Set

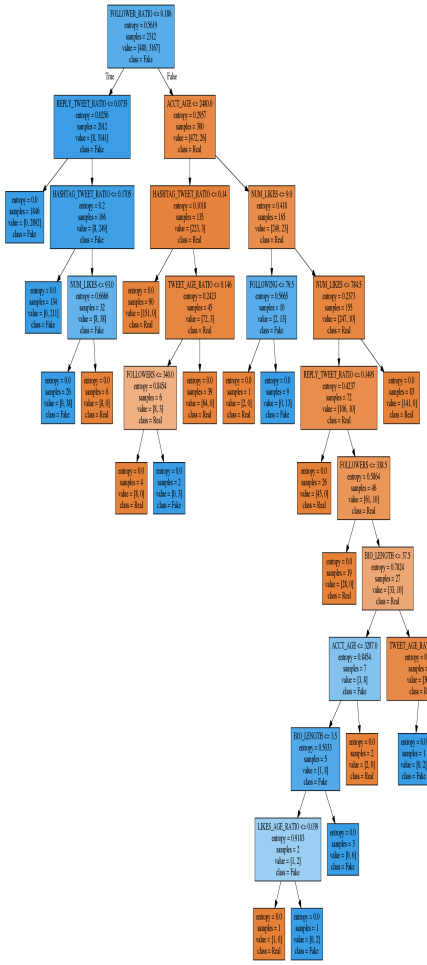
Length of Bio	.004
*Hashtag_tweet_ratio	.0054
Followers	.0145
*URL/Pics_tweet_ratio	.0174
Age of account	.0189
*Reply_tweet_ratio	.0304
Following	.0511
Number of likes	.1105
*Tweet_age_ratio	.1345
*Likes_age_ratio	.1654
Number of likes	.1834
Follower_ratio	.2647

The * next to the feature name in Table V represents a derived feature while the rest of the attributes are basic account elements obtained through the Twitter API. To show that our derived features make a positive difference in regards to accuracy, we run versions of model 3.0 with different attributes to see the differences in accuracy as we change the attributes the model is trained upon.

First, we take the seven basic features, denoted in Table V by not having a *, and build model 3.0 on just those attributes.



(a) Model 1.0 Tree 15



(b) Model 3.0 Tree 0

Fig. 2: Decision Tree's for models 1.0 and 3.0

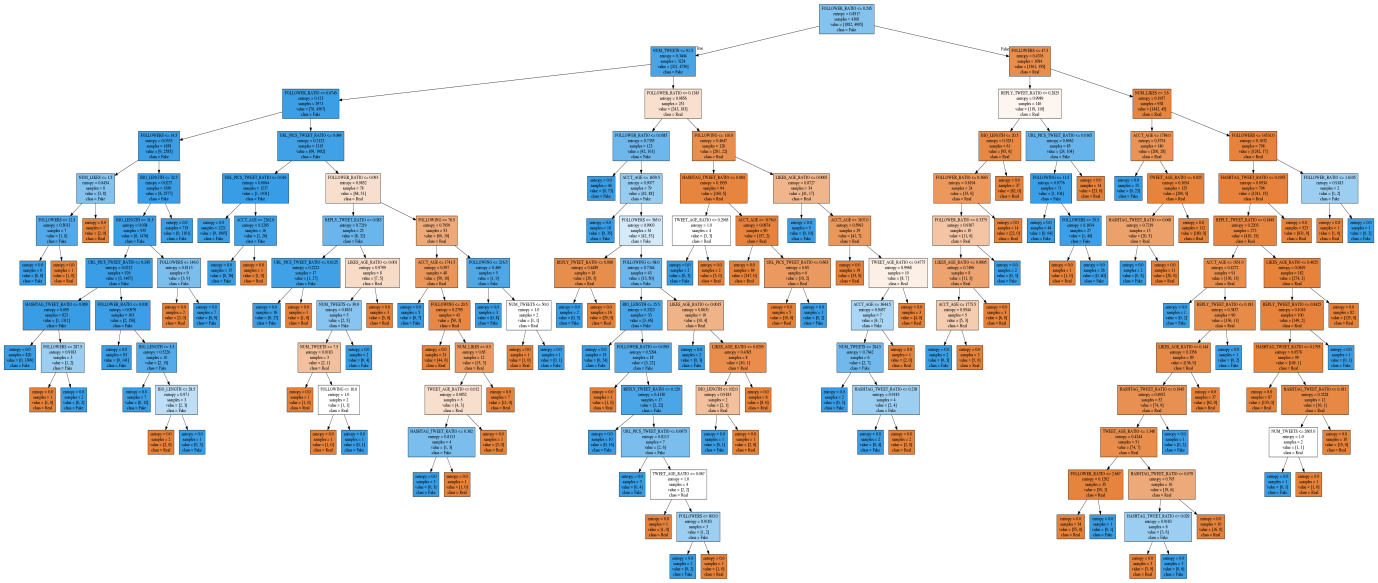


Fig. 3: Model 2.0 Tree 19

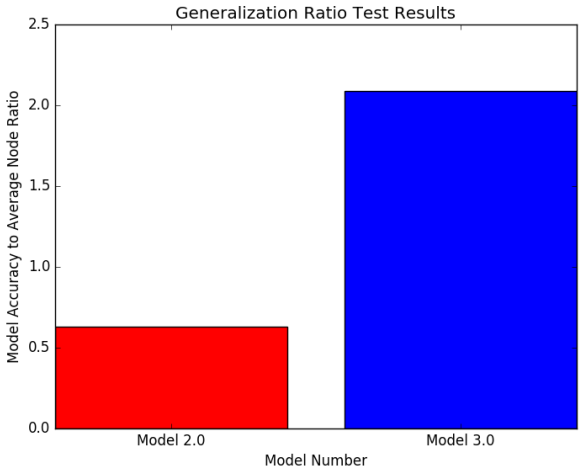


Fig. 4: Accuracy to Generalization Ratio Test

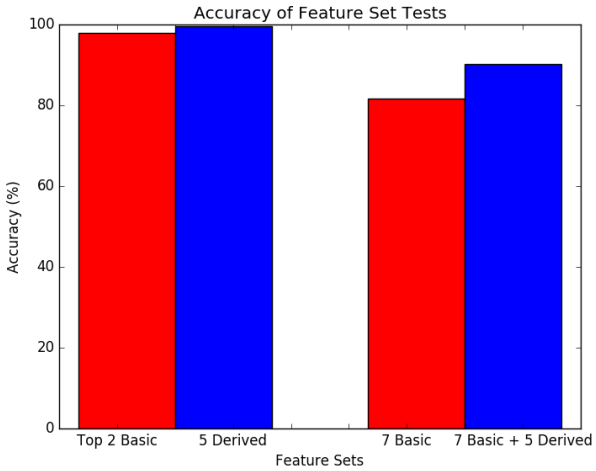


Fig. 5: Feature Set Tests

Over a period of 10 runs the accuracy of the Random Forest model was 81.7%. Therefore, by adding our derived ratio attributes to the model we increase its accuracy by just under 10%. From here, we look at the top two most important features from Table V and build a model on just those two features. When running the model on just Number of likes and Follower_ratio, the model achieved 98% accuracy on the test set. Although this accuracy was higher than the proposed model of all twelve attributes, it is not fit to be deployed as it lacks generalizability and could potentially be overfit; however, we can use this model as a baseline and see if a model using only our derived ratio attributes performs better. Over ten runs, the model trained on our five derived ratio attributes produced an accuracy of 99.6%. This test shows that these ratio attributes provide valuable insight on whether an account is genuine or fake. Using the derived features in conjunction with basic account features yields a generalized model that can classify Twitter accounts at an accuracy of 90.25%. The feature set test results can be seen in Figure 5. One should note that accuracy above 95% shows that the feature set provided valuable insight, but the model might not be generalized enough. Therefore, some tests show high accuracy, but have low feasibility for real world application which is why the proposed model has an accuracy of 90.25%, but is generalized.

IV. Conclusion

The unstoppable growth of social media in today's world also brings forth the problems of false information spreading, malicious content spreading and fake followers for popularity. These problems often involve the use of bot accounts or automated accounts. In this paper we have proposed a machine learning classifier along with a set of features that can accurately detect these automated accounts and label them as bots. To train and test this classifier we use datasets from different years that were gathered by the IIT (Institute of Informatics and Telematics) in Italy. We selected basic features that could be accessed from the Twitter API then created five ratio features that granted additional insight on how the account was generally operated. We then tested different models that used subsets of our proposed attributes in order to demonstrate that our derived ratio features played a significant role in creating a model that was general enough to be deployed onto Twitter, but could also maintain a 90% accuracy on new data.

Acknowledgment

We would like to thank Stefano Cresci and the Institute of Informatics and Telematics in Italy for allowing us to use their Twitter bot datasets.

References

[1] http://www.pewinternet.org/2018/03/01/social-media-use-in-2018/
[2] http://www.internetlivestats.com/twitter-statistics/#rate
[3] https://www.theatlantic.com/technology/archive/2016/11/election-bots/506072/
[4] https://qz.com/1108092/twitter-has-a-serious-bot-problem-and-wikipedia-might-have-the-solution/
[5] https://techterms.com/definition/captcha
[6] Makice, K. (2009). Twitter API: up and running. Sebastopol, CA: O'Reilly.
[7] The Paradigm-Shift of Social Spambots: Evidence, Theories, and Tools for the Arms Race, S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, M. Tesconi. WWW '17 Proceedings of the 26th International Conference on World Wide Web Companion, 963-972, 2017
[8] Fame for sale: efficient detection of fake Twitter followers, S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, M. Tesconi. arXiv:1509.04098 09/2015. Elsevier Decision Support Systems, Volume 80, December 2015, Pages 5671
[9] Widenius, M. and Axmark, D. (2002). MySQL reference manual. Beijing: O'Reilly.
[10] SK-learn RandomForestClassifier, https://tinyurl.com/SKRanFor
[11] https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd
[12] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
[13] Travis E, Oliphant. A guide to NumPy, USA: Trelgol Publishing, (2006).
[14] Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)
[15] http://scikit-learn.org/stable/modules/feature_selection.html