2020

Automatic Distinction Between Twitter Bots and Humans

Jeremiah Stubbs
Minnesota State University, Mankato

Automatic Distinction Between Twitter Bots and Humans

by

Jeremiah Stubbs

Thesis Submitted for Partial Fulfillment

of Bachelor of Science Degree

in Cognitive Science

with Emphasis in Computer Science

Minnesota State University, Mankato

May 2020

Abstract

Weak artificial intelligence uses encoded functions of rules to process information. This kind of intelligence is competent, but lacks consciousness, and therefore cannot comprehend what it is doing. In another view, strong artificial intelligence has a mind of its own that resembles a human mind. Many of the bots on Twitter are only following a set of encoded rules. Previous studies have created machine learning algorithms to determine whether a Twitter account was being run by a human or a bot. Twitter bots are improving and some are even fooling humans. Creating a machine learning algorithm that differentiates a bot from a human gets harder as bots behave more like humans. The goal of the present study focuses on the interaction between humans and computers, and how some Twitter bots seem to trick people into thinking they are human. This thesis evaluates and compares decision trees, random forests of decision trees, and naïve Bayes classifiers to determine which of these machine learning approaches yields the best performance. Each algorithm uses features available to users on Twitter. Sentiment analysis of a tweet is used determine whether or not bots are displaying any emotion, which is used as a feature in each algorithm. Results show some of the most useful features for classifying bots versus humans are the total number of favorites a tweet had, total number of tweets from the account, and whether or not the tweet contained a link. Random forests are able to detect bots 100% of the time on one of the datasets used in this work. Random forest gave the best overall performance of the machine learning approaches used; user-based features were identified as being the most important; and bots did not display high intensities of emotions.

Keywords: decision tree, random forest, naïve Bayes, Twitter, bot, detection

Table of Contents

1. Introduction

Twitter is a micro-blogging social network that allows users to post, share, and reply to messages posted by other Twitter users; a hashtag can be used in a tweet to search for different topics (Appling and Briscoe, 2017). An example of a hashtag is “#COVID19”. Twitter can be an effective tool to get a message across to a large population—such as during presidential elections (Efthimion, Payne, and Proferes, 2018). Unfortunately, this tool can also be abused by creating bots that retweet ideas, news, or opinions more frequently than individuals could, creating the illusion that an opinion is widely shared (Varol, Ferrara, Davis, Menczer, and Flammini, 2017). An example is with the 2016 presidential election where millions of Russian bots, possibly, had some kind of impact on the results of the election (Roeder, 2018). Having one bot posting tweets at a higher frequency than humans is a problem, but it is an even bigger problem if there are millions of bots doing the same thing, especially if the bots are used in other malicious ways such as spreading propaganda, identity theft, or directing others to a malicious website (Battur and Yaligar, 2019). While humans may do all of these things, a further problem on Twitter is the use of bots to spread this information more broadly.

An estimated 9-15% of Twitter accounts are bots (Varol et al, 2017). There may be more bots during a special event, such as a political election, or during a global crisis. With 90-95% of Twitter accounts being humans, this does not seem like much of a problem. In order to understand why Twitter accounts that are run by bots are a problem, it is important to know the definition of a Twitter bot; it is a program that posts messages, autonomously, and supports information dissemination (Varol et al., 2017). If it is used to spread propaganda, or to get people to click on a malicious link, then it would be important to remove these spam messages, and the account, run by a bot.

After finding out what a Twitter bot is, and how it can be used in a negative way, a driving question is whether an automatic classifier of bots is possible using artificial intelligence (AI) that incorporates information available to humans. As the field of AI improves, AI will be able to copy human behavior and model human cognition. By addressing the driving question, the current study examines how good bots are at modeling human behavior. If bots are still a form of classical, rule-based AI, then building an automatic classifier should not be hard, but if bots are doing more than just following the rules of a program, a successful automatic classifier may not be possible.

There is one widely known test for AI generating human-like behavior. Instead of trying to determine whether machines could think, Turing (1950) proposed an imitation game—also known as the Turing test. One room consists of a human of either sex, and a machine. Another room consists of an interrogator who asks a series of questions. The goal of the interrogator is to successfully identify which subject is human (Turing, 1950). If the interrogator can successfully classify the human from the machine, then the machine fails the Turing test. If the interrogator mistakes the machine to be human, then the machine passes the Turing test. Any machine that passes the Turing test is capable of producing intelligent behavior similar to the behavior of humans. In order to correctly classify the human from the machine, the interrogator must know how humans normally behave. In order to build a successful automatic classifier of bots, it is important to know how humans normally behave on Twitter.

Before deciding which features to use to classify a bot from a human, this study looked at which machine learning approach is the most effective at classifying bots and humans. The classification algorithms used are decision trees, random forests of decision trees, and naïve Bayes. Choosing the right algorithm to detect bots is one thing, but choosing the right features to feed the algorithm is equally important. Several studies have shown that user-based features, along with features describing the content, give the best performance. User-based features are easily accessible by Twitter users. Easily accessible means the features are easily found by looking at a Twitter profile or tweet. Some examples include screen name length, profile description, number of friends, followers, and number of retweets (Varol et al., 2017). Content features include linguistic cues such as parts of speech. The second question addresses the features used by the algorithms to determine which ones are the most important, and whether user-based features or content features are better for classification. The final question addresses the emotional content of tweets using sentiment analysis, which indicates whether tweets posted by bots are displaying high intensities of negative emotion to attract attention. If a bot is capable of displaying high intensities of negative or positive emotion, then it is a step closer to modeling human behavior; otherwise, it is just another version of classical rule-based AI.

2. Background

Battur and Yaligar (2019) conducted a study in which an analysis was done on different machine learning algorithms to determine which algorithm is best for classifying tweets as written by bots or humans. These algorithms included decision trees, multinomial naïve Bayes, random forest, and “bag of words”. Features used in their study include “number of followers, friends, screen name, description, location, and verified” (Battur and Yaligar, 2019). After they split 70% of the data into training and 30% for testing, Battur and Yaligar found that the bag of words algorithm had the highest accuracy for classification on both training (97.07%) and test data (95.24%). The bag of words algorithm was later used on real-time Twitter data, and it successfully identified the Twitter bots (Battur & Yaligar, 2019). Battur and Yaligar also found that the decision tree algorithm had an accuracy of 87.85%, and the naïve Bayes algorithm had an accuracy of 69.76 percent. Even though these two algorithms performed worse than the bag of words algorithm, the current study uses these algorithms to confirm the results and to explore the values of different features. Decision trees and naïve Bayes were also chosen over bag of words because these two algorithms were easily available through open source software. This is in part because bag of words matches word frequency patterns and similar information can be included through other features. The current study also wanted to see whether naïve Bayes is the worst to choose in detecting bots, or if it did not receive enough features to do a good job in predicting correct classes.

When analyzing the tweets of a Twitter account, there are many features available for a machine learning algorithm. Davis et al. (2019) used 1,200 features to determine whether an account is a bot or not. Not all of these features are available to the user in real-time. Varol et al. (2017) conducted a study in which twitter data was gathered using 1,150 features in 6 different classes: user-based, friends, network, temporal, content and language, and sentiment features. Models were trained using the different classes of features and Varol et al. (2017) found that user-based features and content features gave the best performance. In order to keep improving the bots, they can be programmed to better resemble the behavior of humans—which can be done by emulating user-based features. As an example, if bots are able to produce tweets at a similar frequency to humans, then tweet count would not be a good feature to use to distinguish a bot from a human. The closer AI resembles the behavior of humans, the more difficult it is to detect whether it is human or a bot.

Efthimion et al. (2018) used a logistic regression and a support vector machine on a bot detection model using several user-based features including profile picture, screen name, followers, friends, description, and tweet descriptions. Their model made predictions about whether an account was a social or traditional spam bot, fake follower, or Russian bot, and achieved an accuracy of 97.75%, a 2.25% misclassification rate, and 98.89% true positive rate (Efthimion et al., 2018). Geo-location was the best feature used for detecting social spam bots, but temporal data and geo-location are not easily accessible by the user when browsing tweets. A Twitter account having less than 30 followers was found to be the best indicator among the features that were easily accessible by browsing Twitter (Efthimion et al., 2018). Due to Russian bots not being very sophisticated, Efthimion et al. (2018) stated that future studies need to include more user-based features, as well as more content features, to pick up more sophisticated bots. The dataset also has to be large enough to avoid underfitting, or overfitting the model to the data.

Appling and Briscoe (2017) conducted a study where they also looked at different features in the automatic detection of bots. Before using a machine learning algorithm to determine the best features, Appling and Briscoe had participants look at several Twitter accounts to determine whether it was a bot or not. Participants said that accounts were run by humans when it was actually a bot 22% of the time (Appling and Briscoe, 2017). Afterwards, machine algorithms including linear support vector machines, random forests, K-nearest neighbors, and extremely randomized trees were used to determine the best features and algorithms for detecting bots. Extremely randomized trees had the best performance (Appling and Briscoe, 2017). The features that were found to be the most important include the presence of a profile URL, tweet count, and linguistic cues from tweets such as words in third person plural, personal pronouns, and verb count (Appling and Briscoe, 2017).

In the early days of AI, it was once thought that machines were not capable of displaying emotions, and were not capable of perceiving the world the way humans perceive the world. AI may display certain emotions, such as happiness or sadness, but would not know what it is like to experience these emotions. With advancements in technology, an empathetic machine, with perceptions of the world, is possible. Kušen and Strembeck (2020) conducted a study where they looked at interactions between humans, and interactions between a human and a bot during specific riot events on Twitter. A riot is a "series of violent acts against public property, private property, and/or other people" (Kušen & Strembeck, 2020). They found that bots can turn any Twitter event into a "riot" by displaying negative emotions in their tweets. Some examples of events that bots can turn into riots include Thanksgiving, responses of racial prejudice by policemen, or promoting a political candidate. They showed bots ended up presenting more negative emotions compared to humans during an event. They also showed bots sent more messages relating to anticipation, fear, and joy, and received more messages from humans relating to anger, fear, joy, surprise, and trust. The bots produced high intensities of negative emotions, and this finding seems to be related to the bot's goal. Usually, the bot's goal is to attract attention and possibly deceive, but Kušen and Strembeck (2020) also found that bots who showed fear received more attention. The following results seem to show how human's think. If a tweet displayed fear, some would respond in a way to make the situation seem less scary, but other responses would try to make the user more scared than they previously were.

Because these prior results show that classification is possible, this work will evaluate and confirm the results using decision trees, random forests of decision trees, and naïve Bayes classifiers. The features used in each algorithm describes the user's Twitter account activity in addition to the content of the tweet. Features that describe the account activity include favorites count and number of tweets, and features that describe the content of the tweet include the frequency of parts of speech. Geurts (2006) found that extremely randomized trees work by reducing variance and increasing bias, which can be a problem if there are a lot of noisy features. The random forest algorithm was picked over extremely randomized trees to avoid this problem.

3. Methodology

This section describes the resources and tools; the two datasets used in this work: gender-classifier dataset (Figure Eight, 2016) and the mid-term 2108 dataset (Yang et al., 2019); the experimental procedures; and the evaluation metrics.

3.1 Resources and Tools

Data cleaning, manipulation, text mining, data visualization, and sentiment analysis were done using R[1] (version 3.6.3) in RStudio[2]. R is a programming language used for mathematical and statistical analyses. Training and testing the machine learning models was also done using R. Part-of-speech tagging was done using the "rJava" (Urbanek, 2019), "NLP" (Hornik, 2018), and "openNLP" (Hornik, 2019) libraries. openNLP determines the part of speech tag by calculating the maximum entropy of the token, and then uses a probability model to predict the correct part of speech tag out of a tag set. The English part of speech model used a Penn Treebank[3] tree set. Sentiment analysis was done using the "sentimentr" library (Rinker & Spinu, 2016). The "rpart" library was used to construct decision trees (Therneau & Atkinson, 2019). The "randomForest", library was used to create several forests containing randomized decision trees (Liaw & Weiner,

[1] https://www.r-project.org/
[2] https://rstudio.com/products/rstudio/
[3] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

2002). The “e1071” library was used to construct naïve Bayes models (Meyer et al., 2019).

A tweet having a sentiment of 0.0 means that the tweet was displaying no emotions or a combination of positive and negative emotions. This value is referred to as the “sentiment score”. Using the sentimentr package, a sentiment score was calculated from each tweet. An example tweet might say, “I like pizza. I hate the snow.” Sentimentr calculates the sentiment of each sentence in this tweet. The sentiment value of the sentence would be positive if the sentence contained positive words such as like, love, and enjoy. The sentiment value of the sentence would be negative if the sentence contained negative words such as hate, dislike, and terrible. If it contained a mixture of positive and negative words, the sentiment score of the sentence would not necessarily be zero. Profanity words are intense negative words. If profanity was used in a sentence with the word “like”, the sentence would likely be more negative because of the intensity of profanity words. With the example tweet, the first sentence may have a sentiment of 0.5, and the second has a sentiment of -0.5. The first is positive because it contains a positive word, and the second is negative because it contains a negative word. If a tweet had more than one sentence, the average sentiment was taken across all the sentences used in the tweet. In the example tweet, the average sentiment would be neutral (0.0).

3.2 Datasets

After cleaning the data by removing tweets with missing feature values, the gender-classifier dataset had 8,411 Twitter accounts. One tweet was pulled from each account. There were 2,321 labeled as written by bots and 6,090 by humans. The mid-term dataset contains several accounts that posted a tweet, or responded to a tweet during the 2018 mid-term election. The mid-term dataset did not have any tweets, and consisted mainly of user-based features. The mid-term dataset contains 7,104 Twitter accounts with 566 labeled bots and 6,538 labeled humans.

The gender-classifier dataset features include number of favorites, tweet count, and other features that were extracted from the tweet including number of pronouns, nouns, verbs, length of the tweet, word count, length of Twitter name, whether or not a Tweet contained a link, sentiment of the Tweet (sentiment score), total negative and positive words, average counts in tweets, as well as the number of hashtags in the Tweet. The mid-term dataset contained features including the time stamp, user_id, screen name, their name, account description, when the account was created, the URL that was in the tweet (if any), language, whether the account was protected, verified, had geo-location enabled, used a default profile background image, whether or not the account had a default profile, listed count, as well as the total number of followers, friends, favorites, and statuses. The features that were selected to use for training and testing include all except the time stamp, user_id, screen name, their actual name, description, when the account was created, and the URL that was in the tweet.

3.3 Procedure

In order to use both datasets, it was necessary to clean the data and address missing values. Instances with missing values were excluded. In the gender-classifier dataset, the original labels were brand, male, and female. The label brand was changed to bot, and labels male and female were changed to human for this work, where gender is not the question being addressed.

Additional processing was done on each tweet to create summative features that provide content information. For each tweet, word count, number of hashtags, and whether there was a link in the tweet were extracted. Part-of-speech tagging and sentiment analysis were done on the tweet, in addition to gathering relative frequencies of personal pronouns, possessive pronouns, nouns, verbs, negative words, and positive words. Counting variables kept track of personal pronouns, possessive pronouns, nouns, verbs, and sentence (tweet) length. Refer to appendix A for the R code used for part-of-speech tagging and sentiment analysis. After processing each tweet, the data was ready to be used for training and testing each model of each machine learning algorithm.

The datasets are not formally divided into training and test sets. The gender-classifier data was split into various training and test sets with different random seeds to determine which split gave the best performance. Splits of 70% training and 30% testing, 75% training and 25% testing, and 65% training and 35% testing were made. Because the gender-classifier data set is 72.4% humans, which can affect model training, accounts written by humans were randomly selected to equal the number accounts written by bots (2,321). This was split into 70% training and 30% testing, which was based on the results from the full dataset. Similarly, the mid-term dataset was randomly split into 70% train and 30% testing. Experimental results are in the next section.

Machine learning models were built for naïve Bayes, decision trees, and random forest using the available features. Parameters that were modified to improve performance included the number of trees in the forest and the number of features available at each split. For random forest experiments, the minimum size of the terminal nodes was two. Refer to appendix B for the R code for building decision trees, random forest, and naïve Bayes models and predicting classes on test sets.

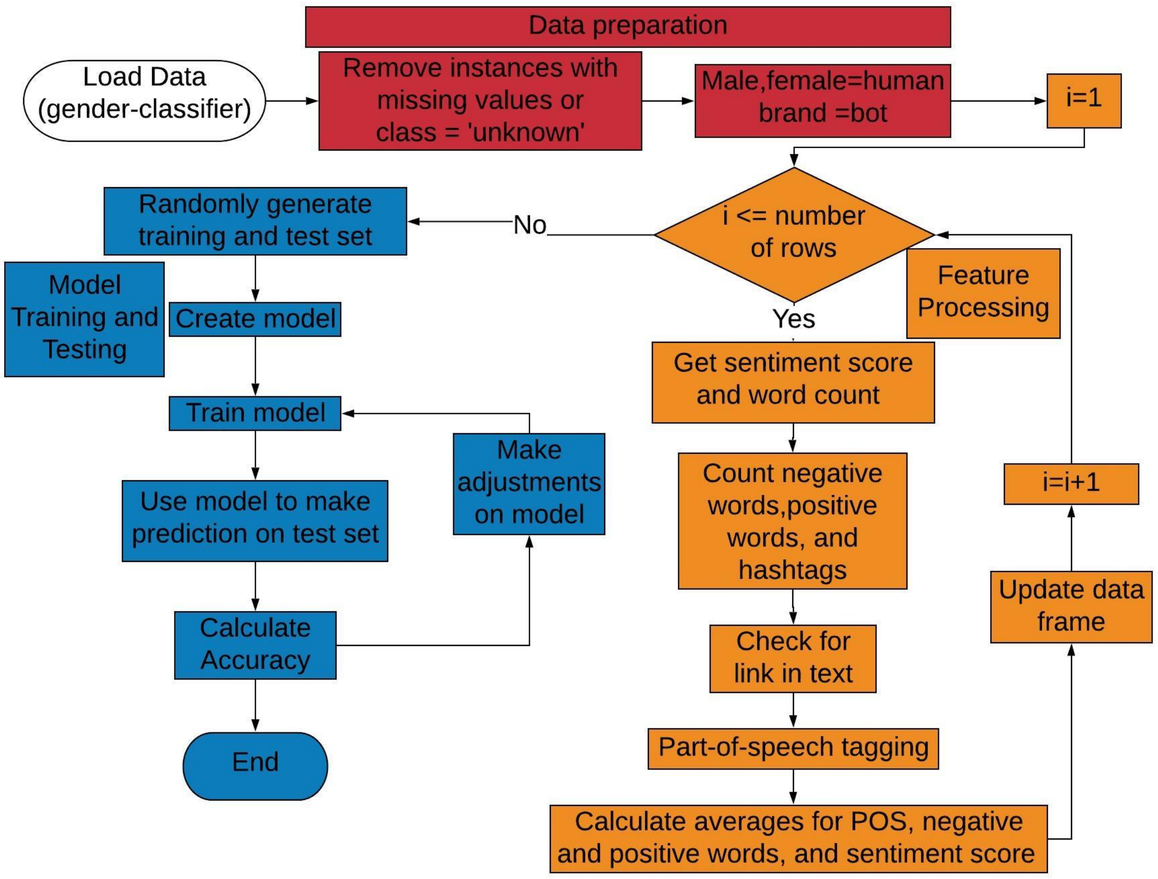
Data preparation
Load Data (gender-classifier)
Remove instances with missing values or class = 'unknown'
Male,female=human brand =bot
i=1
i <= number of rows
No
Yes
Feature Processing
Get sentiment score and word count
Count negative words,positive words, and hashtags
Check for link in text
Part-of-speech tagging
Calculate averages for POS, negative and positive words, and sentiment score
Update data frame
i=i+1
Randomly generate training and test set
Model Training and Testing
Create model
Train model
Use model to make prediction on test set
Make adjustments on model
Calculate Accuracy
End

Figure 1: Flowchart describing a general overview of data cleaning, feature processing, and model training and testing. Feature processing took through about 28 minutes for the gender-classifier dataset on a 7th generation AMD A9-9425 processor devoted to the task.

3.4 Evaluation Metrics

Evaluation of useful features and algorithm performance was done with two approaches. The top five features used in each random forest was recorded. Classification accuracy was used to compare performance across algorithms.

$$Accuracy \ = \frac{\#\ bots\ correctly\ classified + \#\ humans\ correctly\ classified}{total\ number\ of\ bots\ and\ humans}$$

4. Results

This section presents the results that answer the driving question of whether an automatic classifier of Twitter bots is possible. Statistical tests were used to compare several features between humans and bots. The following sections present results that help answer the three other research questions including which machine learning algorithm is the best to use, which features are most important to distinguish a Twitter bot from a human, and whether the bots in the gender-classifier dataset are displaying any emotion.

4.1 Comparing Features Between Humans and Bots

Different statistical tests were done on the gender-classifier dataset to determine whether there was a significant difference of several features between bots and humans. An independent 2-group t-test revealed a significant difference between sentiment score and classification of account (t = 4.3085, df = 4678.9, p < 0.01). Another independent t-test showed a significant difference between tweet count and classification (t = 5.74, df = 3009.3, p < 0.01). One last independent t-test showed a significant difference between number of favorites and class (t = 014.04, df = 6100.3, p < 0.01). Figure 2 shows the difference of the number of favorites and the average sentiment of a tweet between bots and humans in the gender-classifier dataset. Figure 3 shows the relationship between the average sentiment of a tweet and the tweet count. Figure 4 uses the mid-term dataset, and looks at the total number of friends and favorites that each human had. Figure 5 is the same as Figure 4, except it shows the total number of friends and favorites that each bot had.

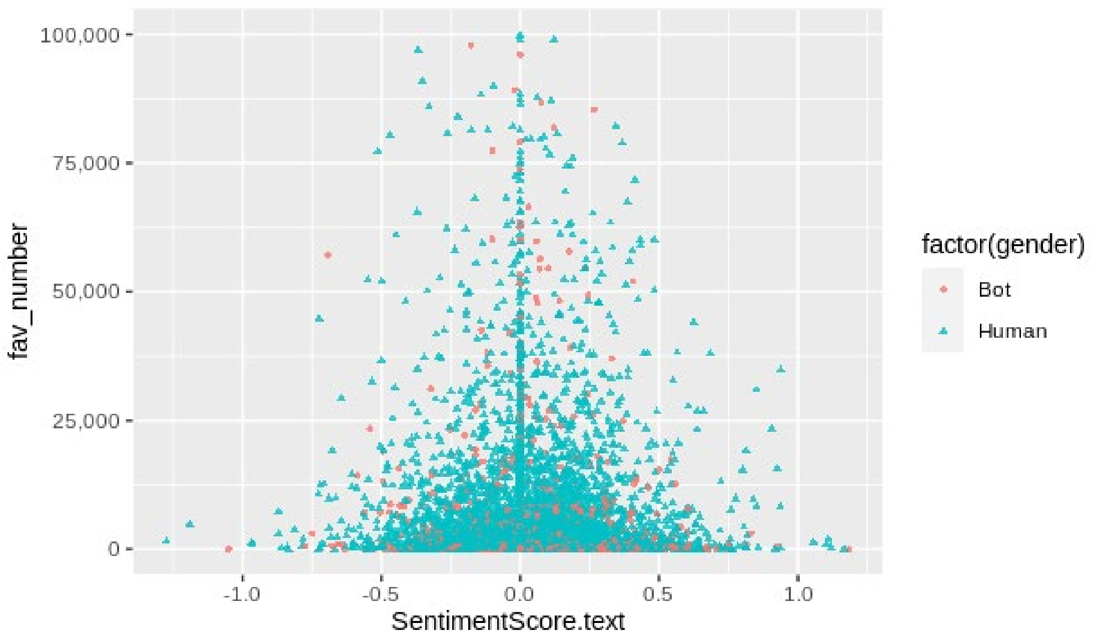

Figure 2: Relationship between sentiment score and number of favorites that resembles a normal distribution.

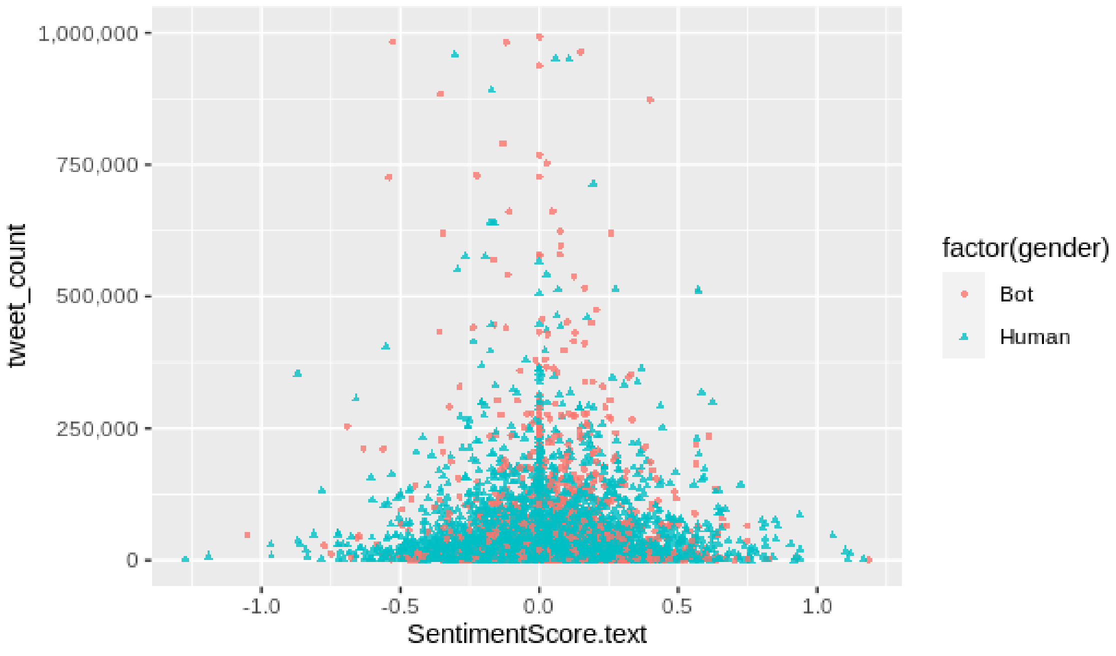

Figure 3: Relationship between sentiment score and tweet count that resembles a normal distribution.

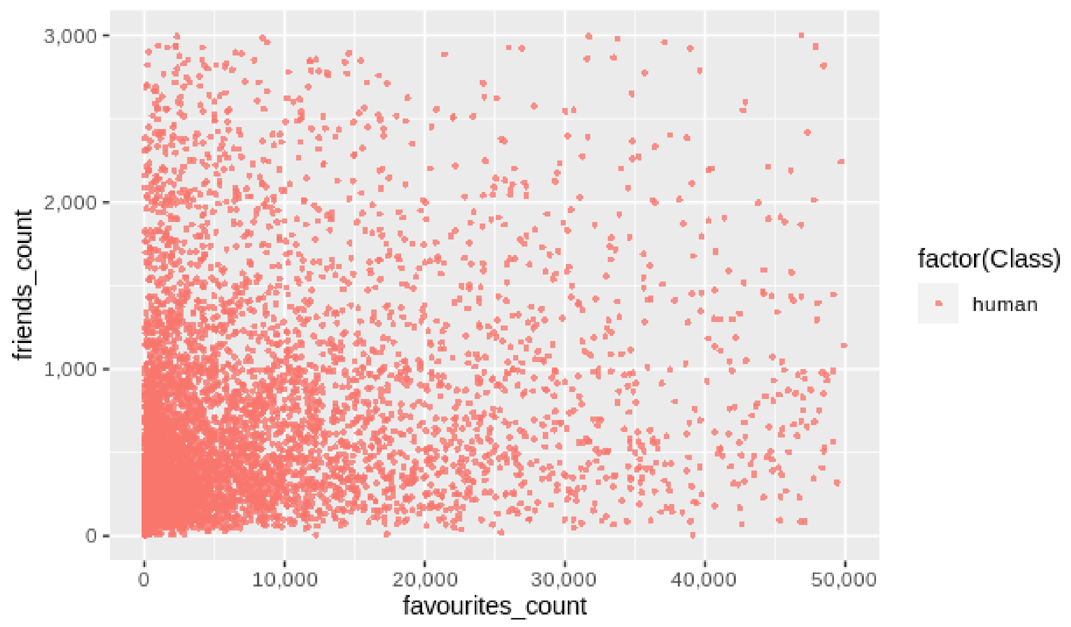

Figure 4: Number of friends and favorites a Twitter account, run by a human, had in the mid-term 2018 dataset (median friends = 561, μ_{friends} = 1551, median favorites = 4720, $\mu_{\text{favorites}}$ = 15262).

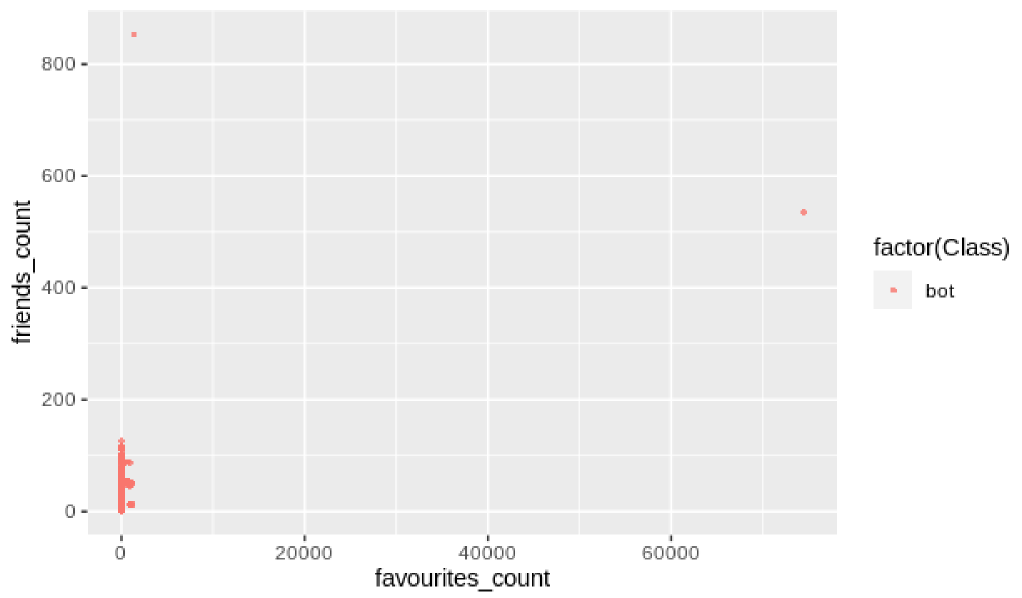

Figure 5: Number of friends and favorites a Twitter account, run by a bot, had in the mid-term 2018 dataset (median friends = 55, μ_{friends} = 60.25, median favorites = 1.0, $\mu_{\text{favorites}}$ = 235.1)

4.2 Performance of Decision Trees, Random Forest, and Naïve Bayes

Table 1 shows the various training and testing splits of the gender-classifier dataset, along with the accuracies on the test set between decision trees, random forest, and naïve Bayes. Figure 7 is the decision tree that was constructed using 70% of the data for training, 30% for testing, and a random seed of 101. Table 2 shows the results of random forest using this same split. Table 3 shows the random forest results on the gender-classifier dataset with a 75/25 training and testing split, and a random seed of 500. For the decision tree with 75% of the data in training, the features at each split were the same as those in Figure 7. Table 4 shows the performance of decision trees and naïve Bayes using the gender-classifier dataset and a 70/30 training and testing split with an equal proportion of humans and bots in the training set. Table 5 shows results for random forest on the same training and test sets. Figure 8 shows the decision tree that was constructed.

Table 6 shows the performance of decision trees and naïve Bayes using the mid-term dataset with a 70/30 training and testing split. Figure 9 is the decision tree that was constructed. Table 7 shows the results of random forest on the mid-term dataset using the same training and test sets. All user-based features were made available for use by the model.

Table 1: Accuracy (%) on test set using different training/test splits on gender-classifier dataset for decision trees (DT), random forest (RF), and naïve Bayes (NB).

train/test/seed	DT	RF	NB	Chance in train	Chance in test
75/25/15	77.41	78.03	71.71	72.37	72.52
75/25/101	76.03	76.80	70.71	72.67	71.61
75/25/500	76.37	78.55	70.85	72.43	72.33
75/25/1000	76.60	78.31	70.76	72.15	73.18
70/30/15	77.02	77.61	71.75	72.46	72.27
70/30/101	75.16	76.27	70.40	72.82	71.43
70/30/500	76.35	78.05	71.51	72.33	72.58
70/30/1000	75.87	78.44	71.35	71.94	73.49
65/35/15	77.38	77.79	71.20	72.34	72.52
65/35/101	75.88	76.49	70.82	72.87	71.53
65/35/500	76.66	77.82	70.72	72.42	72.38
65/35/1000	75.44	77.68	71.09	72.14	72.89

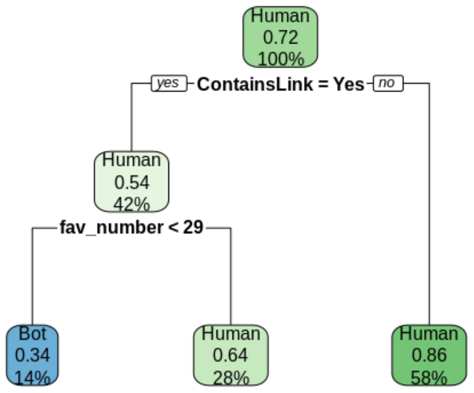

Figure 7: Decision tree with 70/30 split of gender-classifier and 75.16% accuracy on test data.

Table 2: Random Forest accuracy on the gender-classifier dataset with a 70/30/101 split, the number of trees in the forest, features available at each split, and minimum size of terminal nodes set to two. Chance error rate is 71.4%.

Number of trees	Features available at each split	Classification error (%) of bots on training	Classification error (%) of humans on training	Accuracy (%) on training	Accuracy (%) on test
100	2	57.13	9.38	77.64	77.18
100	16	54.19	10.68	77.49	76.23
100	20	55.44	10.75	77.10	76.27
500	2	57.63	8.79	77.93	76.47
500	16	55.06	10.14	77.65	76.26
500	20	54.19	10.17	77.87	76.27
1000	2	57.81	8.80	77.88	76.98
1000	16	54.06	9.89	78.10	76.90
1000	20	54.38	9.84	78.05	76.27

Table 3: Random Forest accuracy on the gender-classifier dataset with a 75/25/500 split, the number of trees in the forest, features available at each split, and minimum size of terminal nodes set to two. Chance error rate is 72.3%.

Number of trees	Features available at each split	Classification error (%) of bots on training	Classification error (%) of humans on training	Accuracy (%) on training	Accuracy (%) on test
100	2	61.18	7.79	77.49	78.03
100	16	56.64	10.02	77.12	78.51
100	20	56.12	10.30	77.06	78.93
500	2	62.91	7.35	77.33	77.22
500	16	56.93	9.56	77.38	78.93
500	20	57.90	9.85	76..90	78.31
1000	2	61.59	7.00	77.95	77.89
1000	16	57.50	9.35	77.38	78.70
1000	20	57.39	9.39	77.38	78.55

Table 4: Accuracy on train and test set using decision tree and Naive Bayes with 70/30/101 training/testing split for the gender-classifier dataset. All features were available for use. Chance error rate is 50%.

DT test accuracy	NB accuracy (%) on train	NB accuracy (%) on test
72.34	69.52	69.52

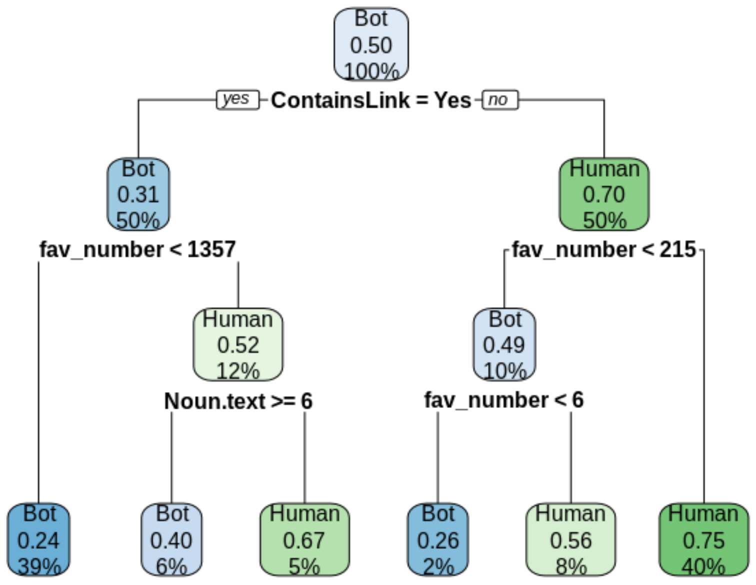

Figure 8: Trained decision tree for the gender-classifier dataset with chance rate of 50% and the root node split on the feature “contains link”.

Table 5: Random Forest accuracy of gender-classifier dataset with a 70/30/101 split of training/test data, the number of trees in the forest, features available at each split, and minimum size of terminal nodes set to two. Chance error rate is 50% on training and test.

Number of trees	Features available at each split	Bot Classification error on training	Human classification error on training	Accuracy on training	Accuracy on test
100	2	28.48	27.36	72.08	96.77
100	16	30.07	27.62	71.15	99.85
100	20	30.20	26.76	71.52	99.83
500	2	28.48	25.64	72.94	97.13
500	16	29.51	25.64	72.43	99.87
500	20	29.94	25.59	72.23	99.83
1000	2	28.22	25.94	72.92	97.16
1000	16	29.34	25.80	72.43	99.87
1000	20	29.60	25.98	72.21	99.87

Table 6: Accuracy on train and test set using decision trees and Naive Bayes on 70/30/101 training/testing split in mid-term 2018 dataset. Chance is 92.0%.

Features used	DT accuracy (%) on test	NB accuracy (%) on train	NB accuracy (%) on test
Language, protected, verified, geo_enabled, background profile picture, default profile, friends count, followers count, friends count, listed count, favorites count, statuses count	95.65	86.13	86.13

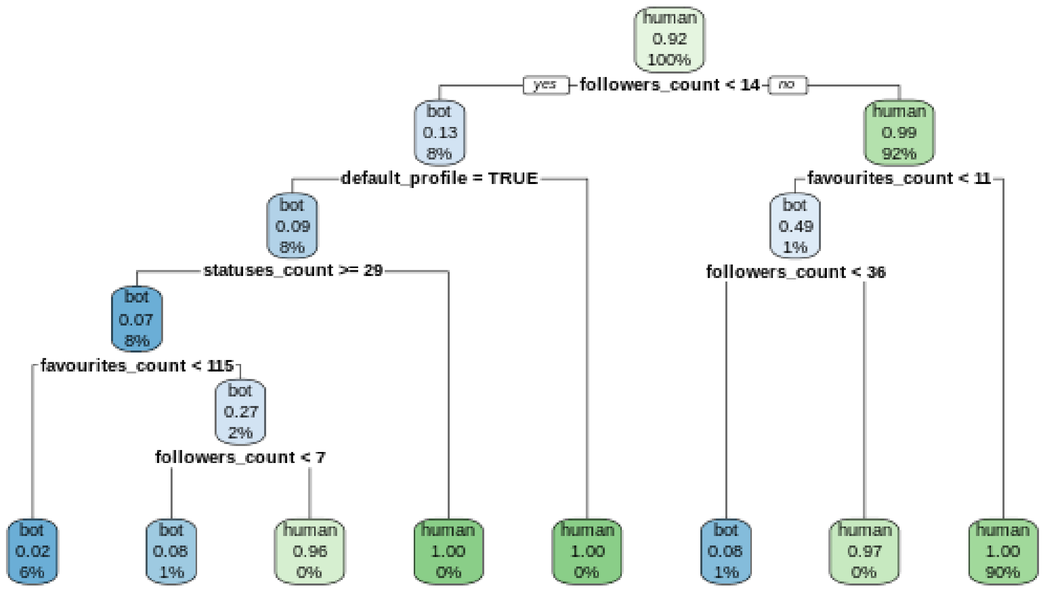

Figure 9: Trained decision tree for the mid-term 2018 dataset

Table 7: Random forest accuracy of mid-term 2018 dataset with a 70/30/101 split of training/test data, the number of trees in the forest, features available at each split, and minimum size of terminal nodes set to two. Chance error rate is 92.0%.

Number of trees	Features available at each split	Bot Classification error on training	Human classification error on training	Accuracy on training	Accuracy on test
100	5	1.24	0.18	99.73	100
100	9	0.88	0.15	99.79	100
500	5	1.24	0.18	99.73	100
500	9	1.06	0.14	99.79	100
1000	5	1.24	0.17	99.75	100
1000	9	0.88	0.17	99.77	100

4.3 Features Selected for Use by the Random Forest Algorithm

This section presents the top 5 features used in each random forest model. Tables 8, 9, and 10 used various training and testing splits of the gender-classifier dataset. Table 8 uses a random seed of 101 and a 70/30 training and testing split. All of the experiments using the 70/30 split from Table 8 were repeated using the 75/25 training and testing split with a seed of 500; the results were the same for both experiments. Table 9 only shows differences in importance of features used in the models that had a 75/25 training and testing split. Table 10 is the same as Table 9 except the model used a 70/30 training and testing split with a random seed of 101, and an equal proportion of bots and humans in the training and test sets. If an experiment is listed in Tables 9 or 10, then different features were used or had a different order than Table 8. If an experiment is not listed in Tables 9 or 10, refer to Table 8 to see the most important features used in the model containing a 75/25/500 training and testing split, and the model containing a

70/30/101 split with a 50% chance error rate.

Table 11 shows the most important features used in random forest with a random seed of 101, 70% of the dataset in training, and 30% used for testing on the mid-term dataset. The importance of each feature was determined by using the Mean Decrease in Impurity (MDI) measurement. The MDI is calculated by taking the number of times a feature is used to split a node divided by the total number of samples it splits (Lee, 2019).

Table 8: Top 5 features used in random forest using a 70/30/101 split of the gender-classifier dataset with different numbers of trees and ranging numbers of features available at node splits.

Number of trees	Features available at each split	Top Five Features used in the decision trees in the forests
100	2	Favorite count, tweet count, sentence length, avgNoun, sentiment score
100	16	Favorite count, tweet count, contains link, sentiment score, sentence length
100	20	Favorite count, tweet count, contains link, sentiment score, sentence length
500	2	Favorite count, tweet count, sentence length, avgNoun, sentiment score
500	16	Favorite count, tweet count, contains link, sentiment score, sentence length
500	20	Favorite count, tweet count, contains link, sentiment score, sentence length
1000	2	Favorite count, tweet count, sentence length, avgNoun, sentiment score
1000	16	Favorite count, tweet count, contains link, sentiment score, sentence length
1000	20	Favorite count, tweet count, contains link, sentiment score, sentence length

Table 9: Changes in top 5 features used in random forest using 75/25/500 split of the gender-classifier dataset.

Number of trees	Features available at each split	Top Five Features used in the decision trees in the forests
100	2	Favorite count, tweet count, sentence length, avgNoun, avgVerb

Table 10: Changes in top 5 features used in random forest using 70/30/101 split of the gender-classifier dataset and 50% chance in training and test.

Number of trees	Features available at each split	Top Five Features used in the decision trees in the forests
100	2	Favorite count, sentence length, tweet count, avgNoun, sentiment score
100	16	Favorite count, contains link, tweet count, sentence length, sentiment score
100	20	Favorite count, contains link, tweet count, sentence length, sentiment score
500	2	Favorite count, tweet count, sentence length, contains link, avgNoun
500	16	Favorite count, contains link, tweet count, sentence length, sentiment score
1000	2	Favorite count, tweet count, sentence length, contains link, avgNoun
1000	16	Favorite count, contains link, tweet count, sentence length, sentiment score
1000	20	Favorite count, contains link, tweet count, sentence length, sentiment score

Table 11: Top 5 features used in random forest using 70/30/101 in mid-term 2018 dataset with different numbers of trees and ranging numbers of features available at node splits.

Number of trees	Features available at each split	Top Five Features used in the decision trees in the forests
100	5	Followers count, favorites count, friends count, statuses count, default profile
100	9	Followers count, favorites count, statuses count, default profile, friends count
500	5	Followers count, favorites count, friends count, statuses count, default profile
500	9	Followers count, favorites count, statuses count, default profile, friends count
1000	5	Followers count, favorites count, friends count, statuses count, default profile
1000	9	Followers count, favorites count, statuses count, default profile, friends count

4.4 Sentiment Analysis of Bots

Figure 6 shows all the instances of bots in the gender-classifier dataset, along with their favorite counts as well as the average sentiment of their tweet. The figure also addresses the question of whether bots are displaying any emotion, with the majority of scores between -0.5 and 0.5.

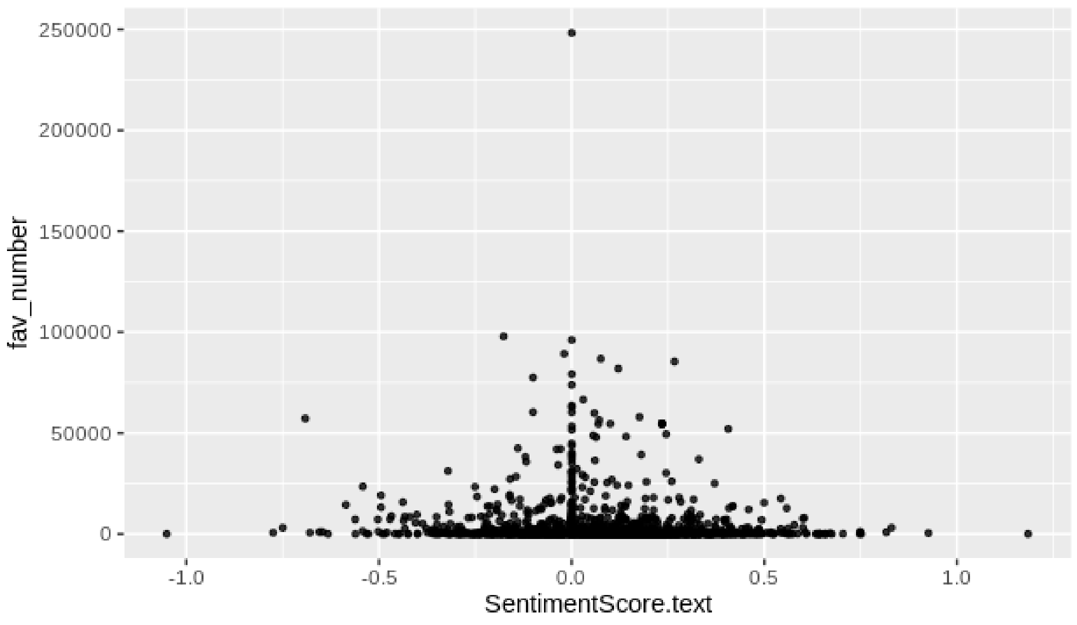

Figure 6: The average sentiment score for each bot tweet in the gender-classifier dataset, along with the favorite number that goes with that tweet (median = 0.03128, μ = 0.06179).

5. Discussion

Discussion of the results begins with looking at the differences in feature values between bots and humans, and why these differences are important in the construction of an automatic classifier. The second topic will address the question of which machine learning algorithm, between decision trees, naïve Bayes, and random forest, is the best to use in the classification of Twitter bots. The third topic discusses which features are the most important to use to make sure that the model has a good performance on the classification of bots and humans. The fourth question addresses the question of whether bots are capable of displaying any emotion. The last topic will mention what future studies can do to improve upon this study.

5.1 Importance of User-based Features in an Automatic Classifier of Bots

The significant difference between feature values for sentiment score, tweet count, and favorite count between humans and bots ($p<0.01$) suggests that Twitter bots do not get as many favorites on their tweets as humans do. The behavior of bots is different from the behavior of humans on Twitter. Bots have close to no friends or total number of favorites. Since the behavior of bots and humans are different, it was very easy for the random forest algorithm to achieve a perfect accuracy on the test set of the mid-term dataset. These Twitter bots would fail the Turing test. AI must resemble human behavior to pass the Turing test. Twitter bots must be able to behave like humans to avoid getting detected in an automatic classifier. Some examples of how it can behave like a human, bots would need to be able to get more favorites on their tweets, and adding other users as friends.

5.2 Decision Trees vs. Random Forest vs Naïve Bayes

The results from Table 1 reveal that naïve Bayes is the worst algorithm to use in the detection of Twitter bots because it consistently performed worse than chance on each test set. The results of naïve Bayes match those from Battur and Yaligar; in their study, naïve Bayes had an accuracy of 69 percent. From Table 1, the performance of naïve Bayes varied from 70-72 percent. At this point, not much work was done to try and improve the accuracy of naïve Bayes because decision trees and random performed better. Random forest had the best performance when 75% of the data was used for training and 25% was used for testing.

In an attempt to try to improve the random forest model, adjustments were made by changing the number of trees used in the forest, and the number of features available to be randomly sampled at each split. Using 100 trees and 20 available features, the model achieved an accuracy of 78.93% on the test set (Table 3). With 500 trees and 16 features available, the model achieved an accuracy of 78.31% on the test set (Table 3). These were the highest accuracies from Table 3. The model with 75% training in Table 1 achieved an accuracy of 78.55% on the test set. There was a slight improvement having fewer trees in the forest, and making all of the features available at each split. More trees are not always better. The random forest algorithm can still do well with fewer trees if it is given enough features. One last conclusion that can be drawn from Table 3 is that the algorithm did a better job at classifying humans, than bots, as noted by the classification error in the training set. This could be partly because the number of humans outweighed the number of bots in the training set. It may have over learned the majority, with less focus on learning the behavior of the bots.

Reducing the number of humans in the training set gave the model a better chance at learning both humans and bots equally (Table 5). With an equal proportion of humans and bots in the training set, the model achieved an accuracy of 99.87% on the test set with 1000 trees in the forest, and 16 or 20 features available. The model also achieved the same accuracy when there were 500 trees in the forest and 16 available features. These results also show that more trees are not always better. Having an equal number of humans and bots in the training set resulted in fewer errors in classification for bots, but there were more errors in classifying humans. From Table 3, classification error for humans was around 10%, and it jumped up to 25-27% in Table 5. The results still show that random forest is the best classification algorithm to use over regular decision trees and naïve Bayes.

The mid-term dataset had more instances of humans, and fewer bots, but had a better performance using decision trees, random forest, and naïve Bayes, compared to the gender-classifier dataset. With a chance baseline of 92.0%, 70% of the data in training, and using 30% for testing, decision trees achieved an accuracy of 95.7% on the test set, and naïve Bayes still does worse than chance with an accuracy of 86.1% on the test set (Table 6). Random forest still out-performed decision trees and naïve Bayes by achieving 100% accuracy on the test set (Table 7). The number of trees in the forest and number of features available did not affect the performance at all due to the perfect accuracy. Similar to the gender-classifier dataset, random forest made fewer errors classifying bots than humans. The classification error is between 0-1.25% for humans and bots. These findings for the mid-term dataset support those by Efthimion et al. (2018) where it was suggested to use more user-based features and a large dataset. It is possible, though, that these bots are also not that sophisticated. The mid-term dataset bots had close to no friends or favorites compared to humans (Figures 4 & 5). The random forest algorithm could easily pick up on these differences, which resulted in the perfect accuracy.

Between decision trees, random forest, and naïve Bayes, the best algorithm is random forest, followed by decision trees, with naïve Bayes coming in last place. The results seem to suggest that having an equal proportion of humans to bots in the training set is not necessary. The model just needs enough user-based features and enough training data to successfully classify a bot from a human. Some features that were not in the gender-classifier dataset, but were in the mid-term dataset include whether the account had a background image on their profile, had a default profile, number of followers, and number of friends. All of these features are user-based features, and are used to determine whether or not a bot is behaving differently than a human on Twitter. When looking at these features, bots are behaving differently from humans as shown in Figures 2, 3, 4, and 5. As long as there is a difference in values of user-based features between bots and humans, it would not be hard to build an automatic classifier to detect these differences.

5.3 Which Features Are Most Important?

From the gender-classifier dataset, the top five features used in random forest were favorite count, tweet count, contains link, sentiment score, and sentence length. As mentioned previously in section 4.3, the Mean Decrease in Impurity (MDI) was the performance measure used to calculate the importance of a feature based on how good it was at classification when used at a particular split of the decision tree. Tables 8, 9, and 10 show the order of importance of the features, starting from most to least, in each random forest model. According to the MDI performance measure, the average number of nouns and verbs in the tweet also showed up in the top five features, but these two features were not consistently in the top five for every experiment. For the mid-term dataset, the top five features used were followers count, favorite count, statuses count, default profile, and friends count. Table 11 shows the order of importance, from most to least, of the features. Similar to the gender-classifier dataset, the order of importance depended on the number of features available at each split. All of the features were important with varying values for MDI, but user-based features typically had the highest MDI.

For both datasets, user-based features were found to be the most important in each model of decision trees and random forest. However, no content-based features were available in the mid-term dataset. Content features including contains link and sentence length were in the top 5 features used for the gender-classifier dataset. These findings support those of Varol et al. (2017) and Efthimion (2018). In order to build an automatic classifier that classifies humans and bots, it is important to look at how humans behave on Twitter, which can be done by looking at the user-based and content features of an account.

5.4 Are Bots Displaying Any Emotion?

A typical classical AI, or rule-following, machine would not be able to produce any emotions because such a machine is only following the rules programmed inside its hardware. As mentioned earlier, Kušen and Strembeck (2020) saw bots in their study displaying high intensities of negative emotion to attract attention. The finding by Kušen and Strembeck demonstrates that artificial intelligence can be more than just rule-following machines. It is possible to implement certain aspects of human cognition, such as emotion, into AI. The current findings seem to disagree with those of Kušen and Strembeck to a point.

Strong displays of emotion are rare in bots (Figure 6), as the majority of bot tweet sentiment scores range between -0.5 and 0.5. This suggests that bots are showing emotion, but do not have high intensities of emotion. An example of a tweet that had a high favorite count, and sentiment score close to zero, is "Throw me into a locker and shut the locker and bully me since im cute and also bad." The tweet had a sentiment score of -0.176 and 97,888 favorites. This bot is using negative emotions to attract attention, but the emotions are not intense. This bot also shows that bots can still elicit an emotional response in humans by displaying little emotion themselves. If Twitter bots can produce high intensities of emotion, and let these emotions guide their behavior, bots could behave similar to humans. An example of how this could be done is if a bot feels happy about a tweet, then it will favorite the tweet. If it does not like the tweet, then it ignores the Tweet to avoid attracting too much attention. The bots who are displaying some emotion show that it is possible for AI to be more than just physical symbol manipulating machines following the rules of a program. It is possible to make intelligent agents that are capable of displaying emotions.

5.5 Future Work

Future studies can do a comparison between random forest, bag of words and support vector machines to see which one yields the best performance. For Battur and Yaligar (2019) the bag of words algorithm gave the best performance. A future question is whether bag of words is a sufficient algorithm on its own to detect bots from humans, or whether it should be used to create a new feature, which would then be one of the features used to train models used in other algorithms. Efthimion et al. (2018) used a support vector machine and achieved a 97.75% accuracy on their test set. The current study found that the worst algorithms were naïve Bayes and decision trees. To determine the best algorithm overall, both decision trees and naïve Bayes can be excluded in future studies. A new question is whether a random forest will outperform a support vector machine.

One should also consider using a different programming language for faster processing of part of speech tagging and sentiment analysis. A language like C++ would speed up this process drastically. There will also be plenty of opportunities to gather data about bots and humans in the future. Work can be done by looking at COVID-19 Twitter bots spreading false information about the pandemic. Ferrara (2020) found that the main goal of COVID-19 bots was to promote political conspiracies. There were plenty of bots during the 2016 presidential election (Roeder, 2018). The 2020 presidential election likely will not be much different. One last thing that can be done in future studies is to get data about time between tweets. Bots typically produce a lot of messages in a short amount of time, but this study cannot make that kind of conclusion with no information about time between tweets. This would be an important feature because if bots have the same number of tweets as humans within the same amount of time, then tweet count would not be a useful feature for prediction.

6. Conclusion

Overall, the behavior of bots is distinguishable from humans, but automatic distinction is not perfect yet. Bots tend to have fewer friends and favorites compared to humans. Bots also produce no emotion or low intensities of positive and negative emotion. The accuracy of each machine learning approach will improve when there are more instances of training data, and when there are more user-based features. There is a difference in user-based feature values between bots and humans; random forest can easily detect these differences and does better than individual decision trees or naïve Bayes, which resulted in a higher accuracy on the test set for the random forest algorithm.

In order to develop an intelligent agent that resembles a human being, and is able to pass the Turing test, one has to look at the behavior of humans. In order to develop an intelligent agent that behaves like a human, it is important to implement human cognition into the agent; at least with social cognition, it is possible to do. An easy and inexpensive way to figure out how to do this is to look at bots on Twitter. Automatic distinction of bots can be perfect with enough user-based features and enough training data, but the perfection may not last long if bots continue to improve by copying human behavior and cognition.

7. References

Appling, Scott; Briscoe, Erica J. (2017): The Perception of Social Bots by Human and Machine. In: Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference, Association for the Advancement of Artificial Intelligence (www.aaai.org), pp. 20-25.

Battur, R., & Yaligar, N. (2019). Twitter Bot Detection using Machine Learning Algorithms. *International Journal of Science and Research, 8*(7). doi: 10.21275/ART20199245

Davis, C. A., Varol, O., Yang, K., Ferrara, E., Flammini, A., & Menczer, F. (2019). Botometer by OSoMe. Retrieved from https://botometer.iuni.iu.edu/#!/faq.

Efthimion, Phillip George; Payne, Scott; and Proferes, Nicholas (2018) "Supervised Machine Learning Bot Detection Techniques to Identify Social Twitter Bots," SMU Data Science Review: Vol. 1: No. 2, Article 5. Available at: https://scholar.smu.edu/datasciencereview/vol1/iss2/5

Ferrara, E. (2020). #COVID-19 on Twitter: Bots, Conspiracies, and Social Media Activism. *ArXiv, abs/2004.09531*.

Figure Eight. (2016). *Twitter User Gender Classification* (Version 1). Retrieved from http://data.world/data-society/twitter-user-data

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, *63*(1), 3–42. doi: 10.1007/s10994-006-6226-1

James, G., Witten, D, Hastie, T., & Tibshirani, R. (2017). ISLR: Data for an Introduction to Statistical Learning with Applications in R. R package version 1.2. https://CRAN.R-project.org/package=ISLR

Hornik,K. (2018). NLP: Natural Language Processing Infrastructure. R package version 0.2-0. https://CRAN.R-project.org/package=NLP

Hornik, K. (2019). openNLP: Apache OpenNLP Tools Interface. R package version 0.2-7. https://CRAN.R-project.org/package=openNLP

Kušen, E., & Strembeck, M. (2020). You talkin' to me? Exploring Human/Bot Communication Patterns during Riot Events. *Information Processing & Management*, *57*(1). doi: 10.1016/j.ipm.2019.102126

Lee, C. (2019). Feature Importance Measures for Tree Models - Part I. Retrieved May 11, 2020, from https://medium.com/the-artificial-impostor/feature-importance-measures-for-tree-models-part-i-47f187c1a2c3.

Liaw, A., & Wiener, M. (2002). Classification and Regression by randomForest. R News 2(3),18–22.

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., and Leisch, F. (2019). e1071:Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. R package version 1.7-3. https://CRAN.R-project.org/package=e1071

Milborrow, S. (2019). rpart.plot: Plot 'rpart' Models: An Enhanced Version of'plot.rpart'. R package version 3.0.8. https://CRAN.R-project.org/package=rpart.plot

R Core Team (2019). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Raja, A. M. (2018). Sentiment Analysis in R - Good vs Not Good - handling Negations. Retrieved February 10, 2020, from https://towardsdatascience.com/sentiment-analysis-in-r-good-vs-not-good-handling-negations-2404ec9ff2ae

Rinker, T., & Spinu, V. (2016). trinker/sentimentr: version 0.4.0 (Version v0.4.0). Zenodo.

http://doi.org/10.5281/zenodo.222103

Roeder, O. (2018). Why We’re Sharing 3 Million Russian Troll Tweets [Web log post]. Retrieved from https://fivethirtyeight.com/features/why-were-sharing-3-million-russian-troll-tweets/

Therneau, T., & Atkinson, B. (2019). rpart: Recursive Partitioning and Regression Trees. R package version 4.1-15. https://CRAN.R-project.org/package=rpart

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, *LIX*(236), 433–460. https://doi.org/10.1093/mind/lix.236.433

Urbanek, S. (2019). rJava: Low-Level R to Java Interface. R package version 0.9-11.https://CRAN.R-project.org/package=rJava

Varol, O., Ferrara, E., Davis, C. A., Menczer, F., & Flammini, A. (2017). Online Human-Bot Interactions: Detection, Estimation, and Characterization. arXiv:1703.03107v2 [cs.SI]

Wickham, H., Hester, J., & Chang, W. (2019). devtools: Tools to Make Developing R Packages Easier. R package version 2.2.1. https://CRAN.R-project.org/package=devtools

Yang, K., Varol, O., Pik-Mai, H., and Menczer, F. (2019). "Scalable and Generalizable Social Bot Detection through Data Selection." AAAI2020, arXiv preprint arXiv:1911.09179.

Appendix A: R Code for Data Preparation and Feature Processing

R Code for feature processing is presented in this section.

A.1 Part of Speech Tagging and Sentiment Analysis

This first section of code lists the libraries used for part-of-speech tagging and sentiment analysis. The data file is read into an R data frame and variables are defined to store feature values.

```r
library(rJava)
library(NLP)
library(openNLP)
library(sentimentr)
library(dplyr)

#library(devtools) ... used to download sentimentr from github
#install_github('trinker/sentimentr') ... sentiment analaysis package

####Cleaned version of gender-classifier dataset
df <- read.csv("BotorNot.csv",stringsAsFactors = F)

###variables to be used for part of speech tagging
sent.token <- Maxent_Sent_Token_Annotator()
word.token <- Maxent_Word_Token_Annotator()
pos.token <- Maxent_POS_Tag_Annotator()

###counting variables
negative <- 0
positive <- 0
PerP.count <- 0
PosP.count <- 0
noun.count <- 0
verb.count <- 0
Hashtag.count <- 0
numRow <- nrow(df)
```

A.2 R Code for Sentiment Analysis

Sentiment analysis and part-of-speech tagging is done inside a loop that cycles through each instance in the dataset. The loop begins by grabbing a tweet from the dataset. Sentiment is calculated using sentimentr library. Average sentiment is calculated along with the word count, total positive and negative words, and whether the tweet had a link in it.

```
for (i in 1:numRow)
{
    s <- df$text[i]
    twitter.name <- df$name[i]

    ####Sentiment analysis
    a <- extract_sentiment_terms(s)
    names <- names(a)
    a <- sentiment(s,by=NULL)
    avgSentiment <- mean(a$sentiment,na.rm = T)
    wordCount <- sum(a$word_count,na.rm=T)

    if ('positive' %in% names)
    {#count up the positive terms
      p <- unlist(a$positive)
      positive <- length(p)
    }

    ####Count up negative words
    if ('negative' %in% names)
    {#count up the negative terms
      n <- unlist(a$negative)
      negative <- length(n)
    }

    #####Does the tweet contain a link?
    if (grepl("https",s) == T){
      contains <- "Yes"
    } else {
      contains <- "No"
    }
```

A.3 R Code for Part-of-speech Tagging

After calculating the sentiment, part-of-speech tagging was done on each tweet. The part-of-speech tags and their relative frequencies were put in a table. If the tag is present in the table, then the variable gets assigned to the value that was in the table. All the different types of nouns were combined into one category—nouns. The same thing is done for all the different types of verbs. The specific types of nouns and verbs are commented in the code. After calculating the frequencies of the part-of-speech tags, the program would count up the number of hashtags that were present in the tweet. The different parts of speech extracted from each tweet include personal and possessive pronouns, all types of nouns, and all types of verbs.

```
#Part of speech tagging
 x <- annotate(s,list(sent.token,word.token,pos.token))
 y <- subset(x,type == 'word')
 tags <- sapply(y$features,'[[',"POS")
 table <- table(tags)

   if (length(table[names(table)=="PRP"]) > 0)
   {#personal pronouns count
     PerP.count <- table[names(table)=="PRP"]
   }

   if (length(table[names(table)=="PRP$"]) > 0)
   {#possessive pronouns count
     PosP.count <- table[names(table)=="PRP$"]
   }

   if (length(table[names(table)=="NN"]) > 0)
   {#number of nouns, singular or mass
     noun.count <- noun.count + table[names(table)=="NN"]
   }

   if (length(table[names(table)=="NNS"]) > 0)
   {#number of nouns, plural
     noun.count <- noun.count + table[names(table)=="NNS"]
   }

   if (length(table[names(table)=="NNP"]) > 0)
   {#number of proper nouns, singular
      noun.count <- noun.count + table[names(table)=="NNP"]
   }

   if (length(table[names(table)=="NNPS"]) > 0)
   {#number of proper nouns, plural
     noun.count <- noun.count + table[names(table)=="NNPS"]
   }
```

```
if (length(table[names(table)=="VB"]) > 0)
{#number of verbs, base form
  verb.count <- verb.count + table[names(table)=="VB"]
}
if (length(table[names(table)=="VBZ"]) > 0)
 {#number of verbs, 3rd person singular present
   verb.count <- verb.count + table[names(table)=="VBZ"]
 }
if (length(table[names(table)=="VBP"]) > 0)
{#number of verbs, non-third person singular present
  verb.count <- verb.count + table[names(table)=="VBP"]
}
if (length(table[names(table)=="VBD"]) > 0)
{#number of verbs, past tense
  verb.count <- verb.count + table[names(table)=="VBD"]
}

if  (length(table[names(table)=="VBN"]) > 0)
{#number of verbs, past participle
  verb.count <- verb.count + table[names(table)=="VBN"]
}
 if (length(table[names(table)=="VBG"]) > 0)
 {#number of verbs, present participle
   verb.count <- verb.count + table[names(table)=="VBG"]
 }
 if (length(table[names(table)=="#"]) > 0)
 {#number of hashtags in tweet
   Hashtag.count <- table[names(table)=="#"] }
```

A.4 Feature Processing Calculations

The final section of the loop takes the extracted feature information and puts it into a data frame where it can be used for training models and predicting classification results. The current row is represented by the variable "i". Each feature value, whether a frequency or calculation, are put into a particular column in row "i". As an example, the frequency of nouns is put into the "Noun.text" column in row "i". The counts are then initialized to zero, and the program grabs the next tweet until it reaches the last row of the dataset.

```
#####Record part of speech data, calculate relative frequencies of parts of
speech
    df$PerPronouns.text[i] <- PerP.count
    df$PosPronouns.text[i] <- PosP.count
    df$Noun.text[i] <- noun.count
    df$verb.text[i] <- verb.count
    df$AvgPerPronoun.text[i] <- (PerP.count / wordCount)*100
    df$AvgPosPronoun.text[i] <- (PosP.count / wordCount)*100
    df$AvgNoun.text[i] <- (noun.count / wordCount)*100
    df$AvgVerb.text[i] <- (verb.count / wordCount)*100

    #######Record other features of tweet
    df$SentenceLength[i] <- nchar(s)
    df$WordCount[i] <- wordCount
    df$name.length[i] <- nchar(twitter.name)
    df$ContainsLink[i] <- contains
    df$SentimentScore.text[i] <- avgSentiment
    df$NegativeWords[i] <- negative
    df$PositiveWords[i] <- positive
    df$AvgNegative[i] <- (negative / wordCount)*100
    df$AvgPositive[i] <- (positive / wordCount)*100
    df$HashtagCount[i] <- Hashtag.count

    ####Reset counts
    negative <- 0
    positive <- 0
    PerP.count <- 0
    PosP.count <- 0
    noun.count <- 0
    verb.count <- 0
    Hashtag.count <- 0
}
```

Appendix B: R Code for Model Training and Testing

R Code for splitting the data into training and test sets, creating models for decision tree, random forest, and naïve Bayes algorithms, and using these models to make predictions on the test set.

B.1 R code for Decision Trees, Random Forest, and Naïve Bayes

This section presents the libraries used to create models of decision trees, random forests, and naïve Bayes. Section 3.1 has additional information about these libraries. Data was randomly sampled by having various percentages of the data go into training. Sampling was done without replacement. All of the training and test sets were saved to be used in multiple experiments for comparable results.

```
library(rpart)
library(rpart.plot)
library(randomForest)
library(ISLR)
library(e1071)

df <- read.csv("midterm2.csv")
df <- read.csv("BotorNot.csv")

#Randomly split dataset into training and test set
sample <- floor(0.70*nrow(df))
set.seed(15)
x <- sample(seq_len(nrow(df)),size = sample,replace=F)
train = df[x,]
test <- df[-x,]
#Save training and test set
write.csv(df,'midTermTrain70.csv')
write.csv(df,'midTermTest30.csv')
train <- read.csv("train70.csv")
test <- read.csv("test30.csv")
```

B.2 Training and Testing the Model

This section gives the R code that was used to create, train, and test the model for decision tree, random forest and naïve Bayes algorithms. In random forest, node size was kept constant at 2. Adjustments were made for values for “Mtry”, or features available at each split, and “ntree”, number of trees in the forest.

```
#Decision Tree Training
#training model
model <- rpart(gender ~ fav_number + tweet_count + SentimentScore.text + AvgNegative +
AvgPositive,method='class',data=train)
model <- rpart(gender~.,data=train,method='class')
rpart.plot(model) #plot decision tree
#make predictions on test set and calculate accuracy
p <- predict(model,test,type='class')
table.dt <- table(test$gender,p)
sum <- sum(table.dt)
accuracy <- (table.dt[1,1] + table.dt[2,2]) / sum
accuracy <- accuracy * 100

#=====================================================
#Naive Bayes training
#create model
model <-  naiveBayes(Class ~.,method="class",data=train)
model <- naiveBayes(Class ~ followers_count + favourites_count + default_profile +
statuses_count, method='class',data=train)
model <- naiveBayes(gender ~ fav_number + tweet_count + SentimentScore.text + AvgNegative
+ AvgPositive,method = 'class',data=train)
model <- naiveBayes(gender ~.,method="class",data=train)

#train model
trainPred <- predict(model,newdata = train)
trainTable<- table(train$gender,trainPred)
trainAccuracy <- ((trainTable[1,1] + trainTable[2,2]) / sum(trainTable)) * 100

#make predictions on test set
testPred <- predict(model,newdata=test)
testTable <- table(test$gender,testPred)
sum.test <- sum(testTable)
testAccuracy <- ((testTable[1,1] + testTable[2,2]) / sum.test) * 100
```

```
#Random forest training
model <- randomForest(Class ~.,data=train,ntree = 1000,mtry=9,nodesize=2,na.action =
na.roughfix)
model <- randomForest(gender ~.,data=train,ntree=1000,mtry=20,nodesize=2)
model$confusion
table1 <- model$confusion
modelAccuracy <-
(table1[1,1]+table1[2,2])/(table1[1,1]+table1[1,2]+table1[2,1]+table1[2,2])*100
modelAccuracy
Importance <- as.data.frame(model$importance)
Importance
Importance <- arrange(Importance,desc(MeanDecreaseGini))
Importance
#make predictions on test set
p2 <- predict(model,test)
table.rf <- table(test$gender,p2) #generate confusion matrix
accuracy.rf <- ((table.rf[1,1] + table.rf[2,2])/( table.rf[1,1] + table.rf[1,2] + table.rf[2,1] +
table.rf[2,2])) *100
accuracy.rf
```