SMU Data Science Review

Volume 1 | Number 2 Article 5

2018

Supervised Machine Learning Bot Detection Techniques to Identify Social Twitter Bots

Phillip George Efthimion
Southern Methodist University, phillip.efthimion@gmail.com

Scott Payne
Southern Methodist University, m.scott.payne@gmail.com

Nicholas Proferes
University of Kentucky, nproferes@uky.edu

Supervised Machine Learning Bot Detection Techniques to Identify Social Twitter Bots

Phillip G. Efthimion[1], Scott Payne[1], Nick Proferes[2]

[1]Master of Science in Data Science, Southern Methodist University
6425 Boaz Lane, Dallas, TX 75205
{pefthimion, mspayne}@smu.edu
nproferes@uky.edu

Abstract. In this paper, we present novel bot detection algorithms to identify Twitter bot accounts and to determine their prevalence in current online discourse. On social media, bots are ubiquitous. Bot accounts are problematic because they can manipulate information, spread misinformation, and promote unverified information, which can adversely affect public opinion on various topics, such as product sales and political campaigns. Detecting bot activity is complex because many bots are actively trying to avoid detection. We present a novel, complex machine learning algorithm utilizing a range of features including: length of user names, reposting rate, temporal patterns, sentiment expression, followers-to-friends ratio, and message variability for bot detection. Our novel technique for Twitter bot detection is effective at detecting bots with a 2.25% misclassification rate.

1 Introduction

The dominance of human users as the primary generators of Internet traffic is coming to an end. In 2016, bots generated more Internet traffic than humans [14]. A bot is a piece of software that completes automated tasks over the Internet. On social media, the prevalence of bots is ubiquitous. By some estimates, nearly 48 million Twitter accounts are automated [13]. Although many bots, such as 'fake follower bots', are easy to detect bots that mimic human behavior and seek to spread information while posing as a human user are more difficult to detect.

Bots serve a plethora of purposes, many of which provide services to users. Bots are categorized as "good" or "bad" based on the transparency with which they disclose their identity. These 'social spambots' can serve a variety of purposes, but can be very difficult to detect, even by human observers [15]. Bad bots do not identify themselves to the web servers they access, while good bots declare and identify themselves. Roughly 44% of Internet bot traffic is categorized as good and the other 56% is categorized as bad [14]. The ability to detect bot accounts on social media sites like Twitter is important for a healthy information exchange ecosystem.

Studies suggest that in the months leading up to the 2016 U.S. Presidential Election, a fifth of all tweets on Twitter that were related to the election came from a legion of bot accounts [1]. Taking up a large percentage of the political discourse in a

well-travelled setting, these bots had a large effect on the Presidential Election by refracting the natural conversations of the issues and events surrounding it. Identifying bots on Twitter have become such an issue that DARPA has held a competition in order to foster new strategies in countering bots on Twitter designed to influence other users.

Identifying problematic bots will allow Twitter users to be shielded by groups that aim to affect the perception of how entities and events are actually being perceived by Twitter's user base. This can lead to users having a skewed perception of the events around them. When working together in large clusters, bots have the ability to push narratives that could be false and misleading. Bots are not necessarily bad. Many serve useful purposes, but the ability to detect bot accounts protects the spontaneous nature of information exchange on social media platforms like Twitter. Additionally, methods to detect bots on Twitter are becoming more complex as the bots themselves and their purposes become more complex. At this point simple equations will not accurately identify bots.

By readily identifying Twitter accounts as bots, users will be educated not to be fooled and manipulated by bot messages on Twitter. Additionally, if bots are discovered early, their messages will not be further amplified by people forwarding them.

A rule-set can be developed to test Twitter accounts to see if they are bots by observing rule-sets from other studies and with bridging different areas to classify together. Twitter users and researchers can use rule-sets to test if accounts are bots. By training and testing these rules on a dataset where each account is confirmed and classified to be a bot accounts can be tested live on Twitter. If accounts can be classified as a bot in real-time, users will be safeguarded against messages and narratives pushed by bots on Twitter.

The rule set has proved to be very effective in classifying bots. When tested against different categories of bot accounts, the rule set proved was very effective and scored high marks in accuracy and true positivity rate. The true positivity rate tells us the percentage of Twitter accounts predicted to be true were actually true. This statistic is an important indicator that there are a low percentage of false positives and misclassification of accounts as bots when they are actually run by people. However, not every variable can be tested in real time, although they were still accurate. This list of variables is not believed to be comprehensive, but does provide an idea of how important these factors can be. Further advanced factors are believed to be needed to identify more sophisticated bots.

The remainder of this paper is organized as follows: In Section 2, background information on the subjects from related works is provided. Section 3 contains a description of the data and an initial analysis. Section 4 explains the novel method to classify Twitter accounts as humans or bot driven. Results are presented in Section 5, followed by the ethical ramifications of bots in social media in Section 6. Finally, a conclusion and plan for future work to be performed in Section 7.

2 Related Work

2.1 Twitter

Twitter, launched in 2006, is a microblogging (extremely short-form blogging), social media network [5]. Communicating via tweets, which are limited in size to only 280 characters, users relay messages to each other. These messages can be in the forms of tweeting, authoring messages; replying, responding to another person's message; and direct messaging, tweeting a message to another user that is not available for view to the public. User accounts converse with each other by tagging each other with the "@" symbol preceding the target account's name. Additionally, users have the ability to interact with other accounts on specific topics by using the hashtag symbol "#". Any tweet containing the hashtag symbol is grouped on a timeline of all tweets that contain that same hashtag.

Users can self-aggregate content they want to see by choosing the accounts they follow. Accounts they follow can be friends, companies, institutions, writers, celebrities, or politicians. Users are also able to communicate and further distribute content by 'liking' and 'retweeting' users' tweets. A tweet that is retweeted is added the user's timeline; a collection of posts that are created by or mention the user. Accounts that follow a user are able to see all content on their timeline.

Twitter activity has been classified into 4 main categories: daily chatter, conversations, URLs, and reporting news [5]. Daily chatter is users informing others about their daily lives. Conversations occur when users tag each other using the '@' symbol. URLs are used to share links to other websites with other users. Reporting news is discussion about current events. These categories can also blend together. News is spread on Twitter through using URLs to link to news articles.

Twitter was estimated to have 69 million monthly active users by the third quarter of 2017 in the United States [10] and 330 million worldwide [12], giving it a global reach. This is substantial growth since its 30 million monthly active users worldwide in the first quarter of 2010 [11].

Twitter became an effective tool in presidential elections to spread political messages. In the 2012 US presidential election, there were 45 million monthly active accounts and the number jumped to 67 million monthly active users in the most recent presidential election in 2016.

2.2 Bots

An Internet bot is an automated software application. It can run any range of tasks and does so repetitively. The implementation of bots on the Internet is so widespread that bots made up 50% of all online traffic in 2016 [14]. Some of the tasks that bots perform are feed fetchers, commercial crawlers, monitoring, and search engine bots. For example, feed fetchers change the display of websites when they are accessed for mobile users and search engine bots collect metadata that allows the search engine to perform. These tasks shape the Internet as we see it daily.

Chu et al. classifies Twitter accounts as human, bot, or cyborg accounts [21]. The distinction between these three classifiers is the level of automation placed on the account. An account that Chu et al. classified as human had no activity that is automated [21]. An account where all of its activity is automated is considered a bot. An account that is a mix of automated and non-automated tweets is considered a cyborg. An account that is classified as a cyborg can be run two different ways. The account could be run in a way that would be classified as a human, but also have some automated messages. An account that is classified as a cyborg could also be automated for all of its activity, but it's controller may sometimes send other, non-scheduled tweets. An example of an automated tweet could be a media company's Twitter account tweeting a link to an article on its website each time an article is published. This is also an example of a benign bot.

2.3 Twitter Bots

A Twitterbot is an Internet bot that operates from a Twitter account. Some of the tasks that can be automated from a Twitter bot are writing Tweets, retweeting, and liking. Twitter does not mind the use of Twitter bot accounts as long as they do not break the Terms of Service through actions such as Tweeting automated messages that are spam or Tweeting misleading links.

Twitter bots, like bots in general, serve a variety of purposes ranging from simple tasks such as following a user to more complex tasks like engaging in discussion with other users. Social bots are a type of bot that interacts with users and whose purpose is to generate content that promotes a particular viewpoint. The veracity of the content is irrelevant to the detection of the social bot. It is estimated that between 9 and 15 percent of Twitter accounts are bots [13]. The goal of our bot detection research is to develop refined techniques that are able to detect social bots that are actively avoiding being caught by traditional bot detection techniques.

There are many types of bots on Twitter. One type of bot exists only to artificially increase the number of followers that an account has [4]. The number of Twitter followers determines its influence because the extent of the followers determines how widely spread is the account's message. and the weight it's message receives. People are more likely to trust an account with 1 million Twitter followers than 100 [5]. Using bots to artificially inflate the number of followers an account is a way to increase one's popularity and attract more human followers [2].

3 Data

The test data consists of different types of bot accounts. This cluster of accounts make up the Cresci-2017 dataset. In the Cresci-2017 dataset, we have three groups each of social spambots and traditional spambots [4]. The social spambots are separated into three main groups. The first group is accounts that retweeted a political candidate in Italy. The second group is spambots attempting to get users to download a mobile app. The third group consists of spambots trying to sell products on Amazon.com.

The traditional spambots are also separated into three main groups. The first group is general spambots without a focus. The second group is spambots that attempt to promote a web URL for users to click on [4]. The third group of traditional spambots are trying to push job offers onto users and for the users to click a given URL [4]. Additionally, we have another type of bot that is fake followers [4]. A fake follower account is one that exists to just make a user appear more popular or influential than they are. Finally, we have a type of accounts that have been verified to be 'real', used by humans. These 'real' accounts were tested by Cresci by contacting users directly, to which their responses had to be manual [4]. For each of these types, there are two separate files: one for user's profile data and one for the user's tweets data. This is one of the datasets used by Botometer in order to train their model [3]. Botometer is a bot detection tool developed by Indiana University Network Science Institute. It operates by inputting the username of a Twitter account and it outputs a percent likely that the inputted account is a bot [3]. Though, the tweets may not be as current and from this year, these accounts have been verified to be bots or used by humans. Downloading current Twitter data from random users cannot be used to train the algorithm unless the account is classified. Classification allows the algorithm to classify a test set of accounts. Without an account having this distinction, which is primarily the case when

Table 1. Distribution of number of account and tweets by Dataset within Cresci 2017 dataset

Grouping	Number of Accounts	Total number of Tweets
Social Spambots	4,912	3,457,344
Traditional Spambots	1,533	6,014,982
Fake Followers	3,351	196,027
Real Users	3,474	8,377,522

There is also a dataset of accounts and their tweets collected by NBC News and released February 14, 2018. They are a group of tweets that Twitter has deemed to have participated in "malicious activity" with concern to this past U.S Presidential Election in 2016 [22]. These bots were a part of networks of accounts that had interacted with over one million users, which Twitter had to notify. These accounts have since been suspended by Twitter but can give us insight into current bot behaviors [22]. The data set consists of 454 accounts and 203,483 tweets written by them.

Figure 1 is the distribution of Twitter accounts by bot type. The largest datasets used for this project are the first and second social spambot groups and the second group of traditional bots. The type of bot with the lowest number of Twitter accounts is the fake followers dataset with less than 500 accounts. The Russian bot dataset also has just under 500 Twitter accounts. There are about 1,000 Twitter accounts in our overall dataset that have been confirmed to be both not automated and human run which are referred to as 'real users'.

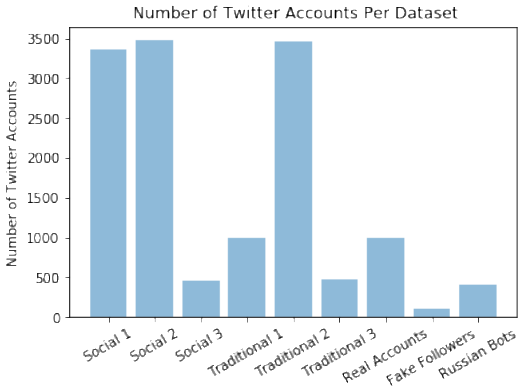

Fig. 1. Distribution of Twitter Accounts versus type and group datasets.

Figure 2 is the distribution of account followers for the different types and groups of bot datasets. The Twitter accounts in the Russian dataset have the most followers of our datasets. They have almost twice as many as the next highest on average. The Russian dataset has on average over 8,000 followers. Real user accounts have less than 1,000 followers on average. A lot less on average than the majority of bot accounts.

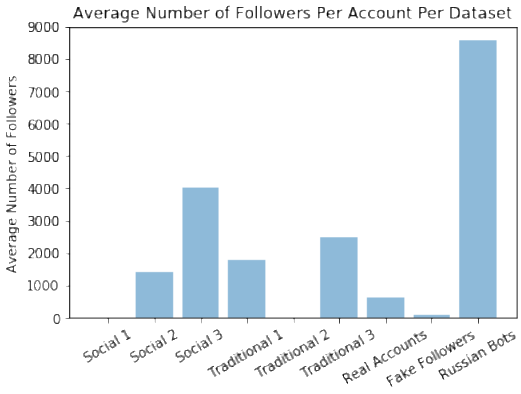

Fig. 2. Distribution of account followers for the different types and groups of bots.

Figure 3 is the average number of friends per Twitter account in each dataset. All of the different bot datasets have friended more people than the fake followers dataset. The second social spambot dataset averages having almost 2,000 friends on Twitter. This is then followed by the accounts in the second traditional and third social bot dataset. The real accounts have sent the fourth most tweets averaging over 1,000. It makes sense that the fake followers would have a very low average of friends because they exist only to follow other accounts.

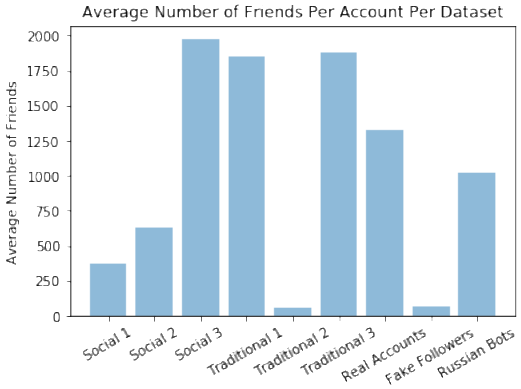

Fig. 3. Average number of friends per Twitter account in each dataset.

Figure 4 is the average number of tweets per account in each dataset. The second social spambots dataset on average has tweeted the most times, over 16,000 times. The next highest are the accounts in the Russian bot dataset and then followed by the second traditional spambot dataset. Real accounts on average have tweeted less than 1,000 times.

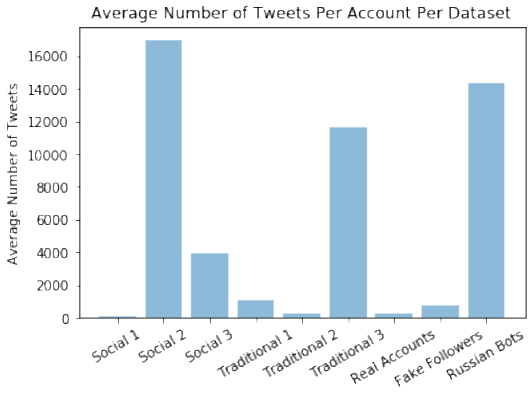

Fig.4. Average Number of Tweets Per Account Per Dataset

Figure 5 is the average number of favorites per account per dataset. Individual account owner will designate a Twitter posting as a "favorite." The account with the most favorites on average is the second social spambots dataset. Its accounts average almost 4,500 favorites. This is followed by the third social spambots dataset which averages over 1,000 favorites. The first traditional spambot dataset averages almost 150 favorites per tweet. The remaining datasets average under 50 favorites per tweet.

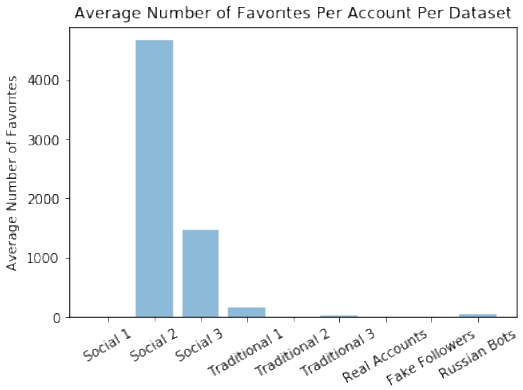

Fig. 5. Average Number of Favorites Per Account Per Dataset

4 Methods and Analysis

On Twitter, information can be gained about a user from their personal account information, tweets, likes, retweets, and direct messages. Users' direct messages are not accessible for privacy reasons. To identify bots, we set up three basic areas for analysis: profile, account activity, and text mining [7].

-Profile: On social media platforms, most users place some personal information about themselves or details to express their individuality. An example of this is a Twitter user's profile image. The image can be of the user, a corporation, or other image that expresses a characteristic that the user wants identified with their account. Lack of such imagery and individual information might be a sign that a Twitter account is a bot. Bots can lack these profile details when a botnet system creates many bots at once. However, these are not sure signs that a Twitter account is run by a bot. With the privacy concerns of today, some users on social media accounts may intentionally hold back personal information to prevent their information from being stolen. We test 14 variables related to each Twitter user's profile to see if an account is a bot. Each of these variables is described in further detail below.

A unique screen name is required for a Twitter account and cannot be changed. It is the account's unique identifier. However, the account's name is optional and can be changed any number of times. We test if an account has a name at all.

Under this philosophy, we also test if an account has a profile picture. Social media accounts will have some form of individuality and a profile picture is the most common. Accounts without those or the default profile image are more likely to be bots.

Main user engagement on Twitter is through reading the content of accounts that are followed. Bots have no reason to follow other accounts as they are trying to disseminate information, not learn from following others. Therefore, we classify an account as a bot if it follows less than 30 accounts.

On the other hand, we do not expect most users to have a large number of friends, accounts which receive and read their tweets, as this would overwhelm their timeline. Therefore, we place a cap on the number of friends an account has. An account with over 1000 friends is marked as a bot.

In addition to the absolute quantity, it is also informative to look at the ratio between the number of friends and the amount of followers an account has. Looking at others research [4], there are different rulesets for classifying bots by this ratio. The StateOfSearch.com ruleset asks for a friend to follower ratio of 100:1 for classification purposes while Socialbaker's FakeFollowersCheck believes only a 50:1 ratio is required. Both have been selected as factors for the algorithm.

Turning on geo-location is another indication of a human user, because it is an account setting bots have no need with which to engage.

The primary goal of some types of bots, such as spambots, is to initiate clicks of a link. The link could be for directing web traffic to a website or to download malicious software on an unsuspecting user. There are many valid reasons for accounts run by humans to contain links, such as to their home websites or online portfolios. Therefore, we take into account this single variable amongst the other 13 variables to determine whether an account is classified as a bot.

Interaction is a bedrock of social media. The volume of tweets generated by an account can distinguish between humans and bots of different intentions. We choose to make the cut off for human accounts a minimum of 50 tweets. We also believe that accounts that are purely fake followers will have never sent a tweet while other types of bots, such as traditional spambots, will have created some statuses to appear real. Therefore, we are grouping bots into related categories of if they contain less than 20 tweets and absolutely zero tweets.

The final profile variable is whether an account has a personalized description. Again, because bot accounts can be made thousands at a time they lack these customizations to be created more quickly.

-Account Activity: Account activity is also an indication if an account is operating by a bot. A bot's automated activity is identified through abnormal user patterns, such as posting all hours of the day and night and posts occurring at the exact time daily or weekly. With Twitter, users are able to pre-set a written tweet to be sent at a certain time. An account that sends a tweet at the same time daily, maybe advertising a limited time offer, would be an example of activity similar to how a bot would behave.

-Text Mining: Text mining also gives insight on whether an account is bot controlled. To disseminate their misinformation, bot accounts may post the same, or very similar, messages repeatedly to evade Twitter's spam filters, which identify repeated messages. Some bots are capable of slightly modifying their original message. We use the Levenshtein distance to measure for similarity of users' tweets. [4]. The Levenshtein distance is the measurement of how many changes would need to be made to convert a first string into a second [4]. A simple example would be how many changes would have to be made to make the word 'Dallas' into the word 'Texas'. By mining the text data, we see these patterns with the messages a Twitter account is sending. The following paragraphs describe the types of patterns in the text that indicate bot activity.

Spam bot accounts try to get other users to click on a website link. Therefore, if text mining concludes the presence of the same string of text in the messages, this may indicate a link and bot activity. There are other text patterns to look for, including the strings 'http', https', 'www', and 'bit.ly,' which identify that there are links to third party websites in a message [1] [4].

Spam comments in blogs contain unnecessary spaces to mask specific words that would otherwise be flagged by filters. To capture these instances, we deleted all spaces from the tweets and measured through the Levenshtein distance.

Applying the Levenshtein distance to a large dataset is very computationally expensive. Therefore, only a smaller sample of data is used when testing the Levenshtein distance.

Table 2. Bot Classification Variables By Area of Analysis

Area of Analysis	Variable
Profile	Absence of id
	Absence of a profile picture
	Absence of a screen name
	Has less than 30 followers
	Not geo-located
	Language not set to English
	Description contains a link
	Has sent less than 50 tweets
	2:1 friends/followers ratio
	Has over 1,000 followers
	Has the default profile image
	Has never tweeted
	50:1 friends/followers ratio
	100:1 friends/followers ratio
	Absence of a description
Text Analysis	Levenshtein distance between user's tweets is less than 30

After analyzing the data for each of the variables tested for the result is placed into a binary matrix. This new matrix is preparation for analysis via support vector machining.

Table 3. Subset of Discrete Matrix to prepare for support vector machine

	id	lang-en	profile_pic	has_screen_name	30followers	geoloc	banner_link	50tweets	twice_num_followers	1000friends	NeverTweeted
2650	415062609	0	0	0	0	1	1	0	1	0	0
731	28757342	1	0	0	1	0	1	0	1	0	0
562	75727639	0	0	0	0	1	0	1	1	1	0
652	2371178828	1	0	0	1	1	0	0	1	0	0
1330	2357220996	1	0	0	1	1	0	1	1	0	0
968	2375824964	1	0	0	1	1	0	1	1	0	0
2020	1127322342	1	1	0	1	0	1	1	1	0	0
689	90211549	0	0	0	0	1	0	1	1	0	0
732	422415442	0	1	0	1	0	1	1	1	1	0
2867	1418869278	1	0	0	0	0	1	0	0	0	0
2993	618844802	1	1	0	1	0	1	1	1	0	0
388	2384827536	1	0	0	1	1	0	1	1	0	0
974	40812882	1	0	0	0	0	1	0	0	0	0
304	531152071	1	0	0	0	1	0	0	1	0	0
446	1702860350	1	0	0	0	1	1	0	0	0	0
2141	2363083622	1	0	0	1	1	0	1	1	0	0
2648	398297815	1	0	0	0	1	1	0	0	0	0
273	531139427	1	0	0	0	1	0	0	0	0	0
1139	1129477836	1	1	0	1	0	1	1	1	0	0
366	182071638	0	1	0	1	0	1	1	1	1	0
3289	1367487373	1	1	0	1	0	1	1	1	1	0
1908	2477931252	1	0	0	0	0	1	0	1	0	0
2678	617073588	1	1	0	1	0	1	1	1	0	0
887	104873917	0	0	0	0	1	0	1	1	0	0
265	62408129	0	0	0	0	1	0	1	1	1	0
219	2645582425	1	0	0	0	1	1	0	0	0	0
2696	617155487	1	1	0	1	0	1	1	1	0	0

However, before using the support vector machining algorithm, on our data, logistic regression is applied. This process of logistic regression followed by applying support vector machining was done so based on Eric Larson's instructional guide [23]. Logistic regression is excellent for preparing data for support vector machining because it outputs the data into binary classifications. This is required for support vector machine.

Support vector machining is used to test our bot detection model against different datasets of known Twitter bots. The efficacy of the model is evaluated by the misclassification rate (error rate) and the true positive rate. A low misclassification rate means we are not misidentifying accounts owned by real people as bots. The misclassification rate is derived by 1 – accuracy of model. The true positive rate is the rate that our algorithm correctly predicts that an account is a bot.

5 Results

Compared against social spambots our model is 95.77% accurate, with a misclassification rate of 4.23% The true positive rate of this model for social spambots is 96.81%. This means that we are correctly identifying that something is a

bot 96.81% of the time which is slightly better than the model's accuracy. This was done with a total dataset of 8,386 total accounts; 4,912 social spambots and 3,474 real accounts.

Looking at the weights in Figure 6, we can see that being geo-located was the best indicator that an account is a social spambot. In terms of readily available information that a user has when browsing Twitter, if the account has less than 30 followers is the best indicator. Variables that weighed negatively with our data were if there was a link in the banner, if the language was set to English, if the account had a profile picture and if the account had over 1000 friends.

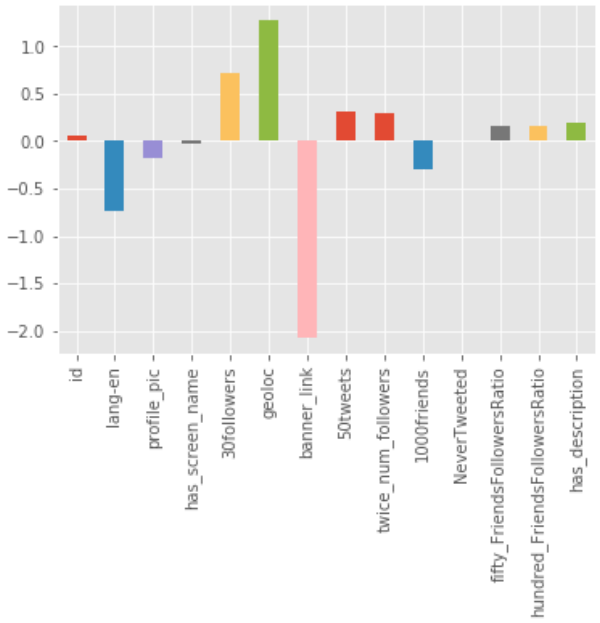

Fig. 6. Logistic Regression weights for social spambots

For tradition spambots in our model, we had an accuracy of 96.25%, misclassification rate of 3.75%, and true positive of 97.13%. As shown below the weights for the variables in the logistic regression before the support vector machining were similar to the ones above for the social spambots.

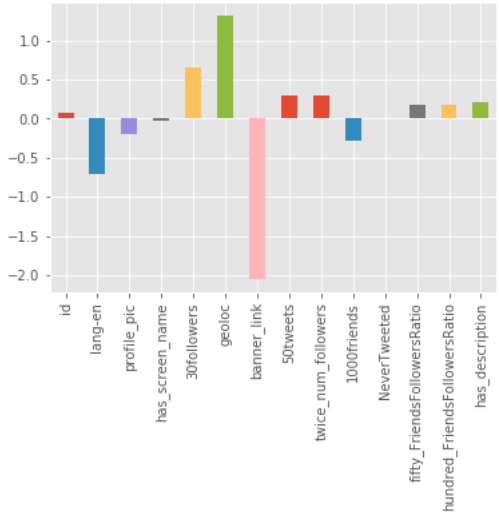

Fig. 7. Logistic Regression weights for traditional spambots

We also want to show that even though the banner_link looks like it is negatively impacting our model the density curves as instances chosen by the support vectors has not. Actually, we have found that removing the banner_link variable reduces the accuracy and true positive rate by over 5%.

The last type of bots that we compare our model to is the fake followers. With our model, looking only at the profile information, we had 100% accuracy and 100% true positive rate. This also means that there were no mis-classified variables. This is the type of bot that our model has identified the best. The weighting from the logistic regression beforehand also looks very different from the two types of spambots. The highest indicator for a fake_follower type of bot was if it had a profile picture. As these types of accounts are not expected to interact in any way with other users, less basic information for them is created. Other important indicators for identifying fake followers are if the accounts had at least 30 followers, had written 50 tweets, and had twice the number of followers than friends.

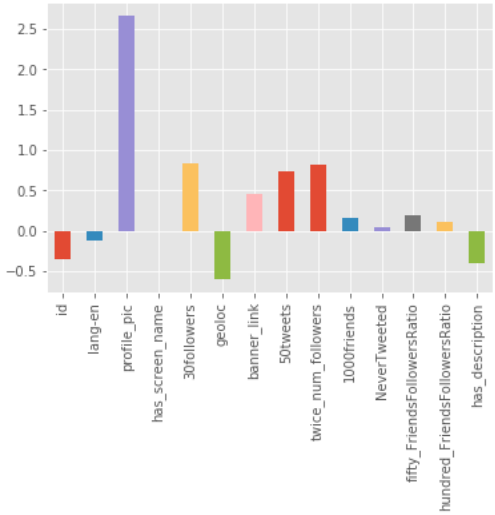

Fig. 8. Logistic Regression Weights for Fake followers bots

For the confirmed Russian bots datasets gathered from NBC News, our model provides a 99.87% accuracy along with a 0.13% misclassification rate and a 98.91% true positive rate. From the logistic regression weights, the profile picture is the most important indicator in deciding if an account is a bot.

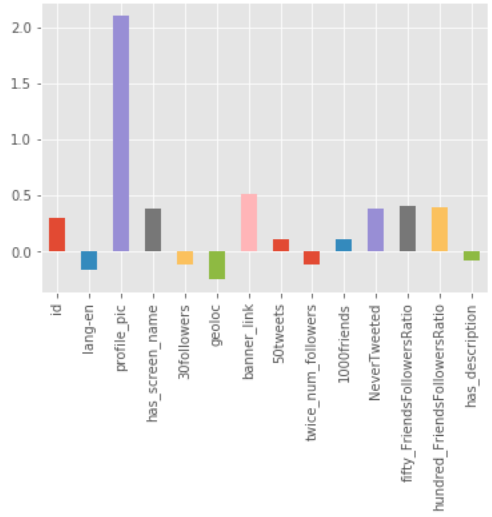

Fig. 9. Logistic Regression Weights on confirmed Russian bots

We will now show the performance of our classification model when using all of the types of bots we have in our dataset. This consists of a total of 16,649 Twitter accounts. Our model scored a 97.75% with a 2.25% misclassification rate and 98.98

true positivity rate. Here is a chart that summarizes our classification findings while using support vector machining on the profile information.

Table 4: Profile Analysis Results

	Accuracy	Misclassification Rate	True Positive Rate
Social Spambot	95.77%	4.23%	96.81%
Traditional Spambot	96.25%	3.75%	97.13%
Fake Followers	100%	0%	100%
NBC News Russian Bots	99.87%	0.13%	98.91%
Total	97.75%	2.25%	98.98%

A subset of the Russian bots and real user accounts is used for the text analysis. It scored to be 90% accurate and therefore had a 10% misclassification rate. The true positive rate for these results is 100%.

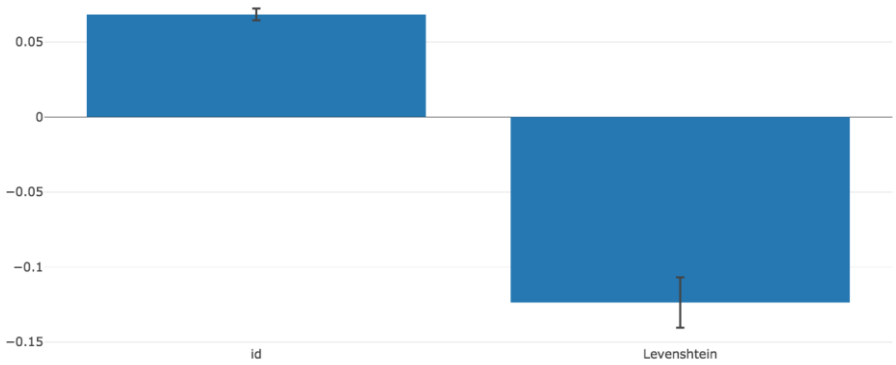

Fig.10. Logistic Regression of Levenshtein Distance

Using the same subset of the data as we did in figure 9, a complete analysis is performed using all of the variables analyzing the profile and the text. It has 100% accuracy, 0% misclassification rate, and 100% true positive rate. However, this analysis is done on a much smaller sample size.

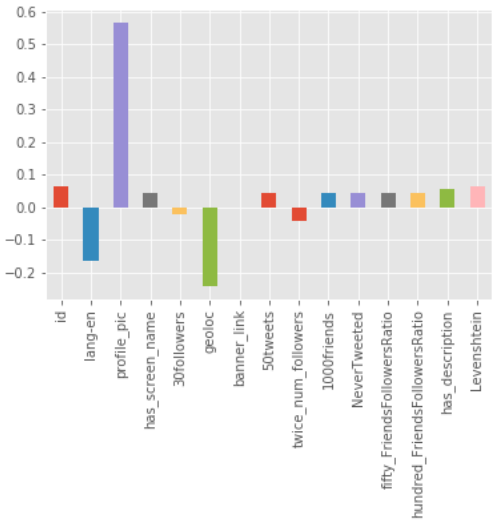

Fig. 11. Logistic Weights for all bots using all variables

On the same smaller sample, support vector machine is performed using every one of the variables. It had a 100% accuracy, 0% misclassification rate, and 100% true positivity rate.

6 Analysis

When using variables related to the Twitter account's profile, only 4.25% of the data is misclassified. Also, 96.81% of the social spambots are correctly predicted. Geolocation and having less than 30 followers were very influential weights for the model. Interestingly, it is seemingly not an important factor that social spambots, which attempt to get users to click on links, do not have links in their profile's description.

The traditional spambots performed similarly to the social spambots, but the analysis performed slightly better. Only 3.75% of accounts became misclassified and the true positivity rating is 97.13%. The weights between the traditional spambots and social spambots are nearly identical. This shows that there is similarity in how the two bot types are constructed.

The fake followers bots were classified accurately in 100% of cases. This along with 0% misclassified, and 100% true positivity rating means that this is the type of bot the model is performing best at diagnosing. This is most likely because fake followers type bots do not perform activities besides following users. Therefore, variables from just using profile information should be enough to properly classify them.

We trained the data on these 3 types of bots and then tested the trained data on the dataset of confirmed Russian bots. When tested, the model only misclassified 2.25% of the users and had a true positivity rating of 98.98%. It is possible that these results scored higher that with the social spambot or traditional spambots because there could have been some accounts that were of the fake follower type which would inflate the scores. The heaviest weighted factor is whether an account had a profile picture. It is interesting that geolocation and having the account's settings set to English did not weigh heavily in the analysis considering these bots were Russian in origin. This means that many Russian bots have their language settings set to English. Writing in English would significantly increase the chance a bot would have a native English speaker interact with them because there would be no language barrier.

When testing for Levenshtein distance, even though the model was 90% accurate, this is most likely due to overfitting. This analysis will need to be redone with a larger dataset. Calculating the Levenshtein distance for the entire dataset is computationally heavy. A more efficient method will have to be researched in order for this variable to be effective in this analysis.

7 Ethics

There are many ethical issues regarding the use of public data gathered from the internet. In the world of social media, the information collected contains personal data that is linked to user accounts that could be linked to an individual's identity. We must ensure that we collect our data and use it in an ethical manner and obey all of Twitter's guidelines on fair use. These guidelines allow for the collection of Twitter data using proper methods to then be used in research, but the guidelines are constantly evolving. Twitter initially allowed any Twitter data collected in the proper way to be shared as a complete data set. Twitter has now amended its policies to only allow the sharing of account or tweet IDs as a data set. This requires researchers to populate the data using their own API key in a process known as "rehydrating". While this provides more protection for users to have their information removed from Twitter and not appear in future data sets that are "rehydrated" after the date a user has deleted their accounts or tweets, it complicates matters for researchers.

One of the first ethical issues is that of informed consent [17]. In studies, subjects must opt into the study in order for their work to be used. This is to ensure that subjects know exactly what the study is and what they are signing up for. However, Twitter is a public social media forum, where anyone can read a publicly shared tweet. Therefore, it can be argued that consent is not needed in this case. There could also be the case of that we are taking information from an account, not a person. A bot account may not even be able to process what it is being asked. Also, Twitter's Term and Conditions have this policy outlined that bot accounts need to identify themselves as such. One possible way to combat this issue is as Webb et al. described as an opt out approach. This is where we send each account a message saying that they can opt out of the study if they so choose.

Two other issues Webb et al. describe are do no harm and protect anonymity [17]. Only a small portion of Twitter accounts (primarily celebrity and corporate/brand

accounts) have their identities confirmed and a large amount use false names for an online persona. It is common practice to hide any personal information when performing a study, which can easily be done by not showing any account names. However, the contents of a tweet could be enough to reveal a user's identity using its contents and timestamp. Using the Twitter API, it would be very easy to identify a user by inputting the exact tweet plus a timestamp. In 2017, there have been numerous circumstances of people being doxed[1] from their tweets that led to their eventually firing. Others, such as ESPN's Jemele Hill, have been suspended for views expressed on her Twitter account. We do not believe in bringing harm to a user or risk bringing harm to them in any way. Therefore, we will not be publishing any individual tweets. We will still collect the contents of each tweet for our study, but the individual tweets themselves will not be published. The reason that we need to collect the information of the tweets is to perform text mining on each tweet's content for our algorithm. We will protect the anonymity of users in this study by not publishing personally identifying or account identifying information.

There is also the ethical dilemma of sharing the results [19]. We must answer the question of what is the ethical process of informing Twitter users that we believe an account is a bot. Because bot accounts that do not identify themselves are in violation of the Twitter TOS (Terms of Service), it is acceptable to identify them as bots. The algorithm that we create will only give a percent certainty, so it is possible that we flag an account as a malevolent bot, but if that flagged account is a person and not a bot, then we will have created a new ethical concern. The best solution to this ethical problem is to provide tools for users to be able to identify bot accounts themselves and block the bot content or report the account to Twitter if they choose.

8 Conclusions

The ruleset that we have proposed works best against bots that are the fake follower type. This can be improved even further by adding more variables about users activity patterns and the contents of the tweets. A large dataset is required to adequately analyze the tweets.

The Russian tweets may be among the less sophisticated as they were discovered. More variables are required in order to potentially find a more sophisticated bot.

With the ability to discriminate between real user accounts and malicious Twitter bots, our model could be applied to stop the spread of false information. According to a survey conducted by Zignal Labs which received responses by over 2,000 adults located the US, 86% of Americans do not always fact check articles that they have read via a link on social media [24]. Additionally, 27% of the respondents in the survey admit they do not fact-check articles they themselves share [24]. Intercepting in real time with the credibility of the information or opinion will decrease the chance the user spreads false information.

[1] Having one's personal information or documents leaked online

Out theoretical end goal is a way for Twitter users to identify whether an account is a bot or not with as little extra work as possible to make it more likely that our information gets used. Our end goal is an Internet browser extension that allow users to identify if an account is a bot without leaving the website. This information will be relayed by hovering over an account name with your mouse. When done so, our proposed extension will display a bubble containing our model's conclusion on whether the account is a bot. Our idea is that if users understand that information is from a source that they do not know and is from a bot that they will not blindly spread it without more research. In this case information is not only in the form of links to articles. It could also pertain to eye-witness claims and information from unknown reporters. As Ben Popkin from NBC News stated, many of the Russian bot accounts were 'impersonating Americans' [23]. They were also tweeting during large events such as debates, and terrorist attacks. Possibly to influence people's opinions on topics. By having a real time tool at people's fingertips, we can prevent unwelcome influence.

According to Sinan Aral and his team "it took the truth six times as long as falsehoods to reach 1,500 people' [25] The danger of one person reading incorrect media is that it can easily be spread to others. Therefore, we have developed a method to let people fact check the validity of Twitter accounts without having to leave the website or their Twitter app on their smartphone. Having this chrome extension use our ruleset to identify bots in real-time is an ideal implementation of the ruleset in future work.

References

1. A. Bessi and E. Ferrara, "Social bots distort the 2016 U.S. Presidential election online discussion," *First Monday*, vol. 21, no.11, Nov 2016.
2. Alex Hai Wang. Detecting Spam Bots in Online Social Networking Sites: A Machine Learning Approach. Sara Foresti; Sushil Jajodia. 24th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSEC), Jun 2010, Rome, Italy. Springer, Lecture Notes in Computer Science, LNCS-6166, pp.335-342, 2010, Data and Applications Security and Privacy XXIV. <10.1007/978-3-642-13739-6_25>. <hal-01056675>
3. Botometer, https://botometer.iuni.iu.edu/#!/.
4. Cresi, S., Di Pietro, R., Petrocchi, M., Spognardi, A., & Tesconi, M. (2015). Fame for sale: Efficient detection of fake Twitter followers. Decision Support Systems, 80, 56-71.
5. A. Java, X. Song, T. Finin, and B. Tseng, "Why We Twitter: Understanding Microblogging Usage and Communities, "*Proc. Ninth WebKDD and First SNA-KDD Workshop Web Mining and Social Network Analysis, 2007*, 2007.
6. Shaffer, Kris. "Spot a Bot: Identifying Automation and Disinformation on Social Media." *Medium*, Data for Democracy, 5 June 2017, medium.com/data-for-democracy/spot-a-bot-identifying-automation-and-disinformation-on-social-media-2966ad93a203.
7. Subrahmanian, V. S., Amos Azaria, Skylar Durst, Vadim Kagan, Aram Galstyan, Kristina Lerman, Linhong Zhu, Emilio Ferrara, Alessandro Flammini and Filippo Menczer. "The DARPA Twitter Bot Challenge." *Computer* 49 (2016): 38-46.
8. S. Yardi, D. Romero, G. Shoenbeck, and D. Boyd, "Detecting Spam in a Twitter Network," *First Monday*, vol. 15, no.1, Jan 2010.
9. The Fake Project, Dataset, http://mib.projects.itt.cnr.it/dataset.html.

10. Twitter. Number of monthly active Twitter users in the United States from 1st quarter 2010 to 3rd quarter 2017 (in million). In Statista – The Statistics Portal. Retrieved October 23, 2017, from https://www.statista.com/statistics/274564/monthly-active-twitter-users-in-the-united-states/.
11. Twitter. Number of monthly active Twitter users Worldwide from 1st quarter 2010 to 3rd quarter 2017 (in million). In Statista – The Statistics Portal. Retrieved October 23, 2017, from https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/.
12. Twitter. (2017). *Twitter Announces Third Quarter 2017 Results*. Retrieved from http://files.shareholder.com/downloads/AMDA-2F526X/5465364539x0x961127/658476E7-9D8B-4B17-BE5D-B77034D21FCE/TWTR_Q3_17_Earnings_Press_Release.pdf.
13. Varol, Onur, et al. "Online Human-Bot Interactions: Detection, Estimation, and Characterization." *Online Human-Bot Interactions: Detection, Estimation, and Characterization*, 9 Mar. 2017, arxiv.org/abs/1703.03107v1.
14. Zeifman, Igal. "Bot Traffic Report 2016. "*Incapsula.com*, Imperva, www.incapsula.com/blog/bot-traffic-report-2016.html.
15. Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini. 2016. The rise of social bots. *Commun. ACM* 59, 7 (June 2016), 96-104. DOI: https://doi.org/10.1145/2818717
16. Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10* (EMNLP '02), Vol. 10. Association for Computational Linguistics, Stroudsburg, PA, USA, 79-86. DOI: https://doi.org/10.3115/1118693.1118704
17. Helena Webb, Marina Jirotka, Bernd Carsten Stahl, William Housley, Adam Edwards, Matthew Williams, Rob Procter, Omer Rana, and Pete Burnap. 2017. The Ethical Challenges of Publishing Twitter Data for Research Dissemination. In *Proceedings of the 2017 ACM on Web Science Conference* (WebSci '17). ACM, New York, NY, USA, 339-348. DOI: https://doi.org/10.1145/3091478.3091489
18. Mozetič I, Grčar M, Smailović J (2016) Multilingual Twitter Sentiment Classification: The Role of Human Annotators. PLoS ONE 11(5): e0155036. https://doi.org/10.1371/journal.pone.0155036
19. Matthew L Williams, Pete Burnap, Luke Sloan. May 26, 2017. Towards an Ethical Framework for Publishing Twitter Data in Social Research: Taking into Account Users' Views, Online Context and Algorithmic Estimation. In *Sociology*. Vol 51, Issue 6, pp. 1149-1168. DOI: https://doi.org/10.1177/0038038517708140.
20. Hutto, C.J. and Eric Gilbert. VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf.
21. Chu, Zi, et al. "Detecting Automation of Twitter Accounts: Are You a Human, Bot, or Cyborg?" *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 6, 2012, pp. 811-824., doi:10.1109/tdsc.2012.75.
22. Popken, B. (2018, February 14). Twitter deleted Russian troll tweets. So we published more than 200,000 of them. Retrieved from https://www.nbcnews.com/tech/social-media/now-available-more-200-000-deleted-russian-troll-tweets-n844731
23. Larson, Eric. Logistic Regression, SVMs, and Gradient Optimization. https://github.com/eclarson/DataMiningNotebooks/blob/master/04.%20Logits%20and%20SVM.ipynb
24. Brown, E. (2017, May 10). 9 out of 10 Americans don't fact-check information they read on social media. Retrieved from http://www.zdnet.com/article/nine-out-of-ten-americans-dont-fact-check-information-they-read-on-social-media/
25. Fox, M. (2018, March 8). Want something to go viral? Make it fake news. Retreived from https://www.nbcnews.com/health/health-news/fake-news-lies-spread-faster-social-media-truth-does-n854896

Appendix: Code

```
# coding: utf-8

# In[3]:


import pandas as pd
import numpy as np
import os

import Levenshtein as Lev
from sklearn.utils import shuffle
import datetime as dt
import editdistance


# In[4]:


# Russian Data Set
rus_tweets = pd.read_csv('/Users/Phillip/Downloads/RussianData/tweets.csv',
na_filter=False)

rus_tweets.info()


# In[5]:


samp_rus_tweets = rus_tweets[0:10]
samp_rus_tweets = samp_rus_tweets['text']
samp_rus_tweets = samp_rus_tweets.str.replace(' ','')
samp_rus_tweets = samp_rus_tweets.str.replace('RT@','')
#samp_rus_tweets = samp_rus_tweets.sub,'')

#samp_rus_tweets[samp_rus_tweets.find("@")+1:samp_rus_tweets.find(":")]
```

```
samp_rus_tweets


# In[6]:


from itertools import product

dist = np.empty(samp_rus_tweets.shape[0]**2, dtype=int)
for i, x in enumerate(product(samp_rus_tweets, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, samp_rus_tweets.shape[0]))

#dist_df
print(dist_df)


# In[7]:


mean_dist = dist_df.mean()
mean_dist.mean()


# In[61]:


rus_tweets_sorted = rus_tweets.sort_values(by=['user_key'])

rus_tweets_sorted = rus_tweets_sorted['text']
rus_tweets_sorted = rus_tweets_sorted.str.replace(' ','')
rus_tweets_sorted = rus_tweets_sorted.str.replace('RT@','')

rus_tweets_sorted1 = rus_tweets_sorted[4171:4207]
rus_tweets_sorted2 = rus_tweets_sorted[4208:4224]
rus_tweets_sorted3 = rus_tweets_sorted[4225:4289]
rus_tweets_sorted4 = rus_tweets_sorted[4290:4327]
rus_tweets_sorted5 = rus_tweets_sorted[4328:4340]
rus_tweets_sorted6 = rus_tweets_sorted[4341:4380]
#rus_tweets_sorted7 = rus_tweets_sorted[4381:4381]
rus_tweets_sorted8 = rus_tweets_sorted[4382:4432]
rus_tweets_sorted9 = rus_tweets_sorted[4433:4434]
rus_tweets_sorted10 = rus_tweets_sorted[4435:4487]
rus_tweets_sorted11 = rus_tweets_sorted[4488:4892]
rus_tweets_sorted12 = rus_tweets_sorted[4893:4908]
rus_tweets_sorted13 = rus_tweets_sorted[4909:4932]
```

```
rus_tweets_sorted14 = rus_tweets_sorted[4933:4941]
rus_tweets_sorted15 = rus_tweets_sorted[4942:5015]
rus_tweets_sorted16a = rus_tweets_sorted[5016:9284]
rus_tweets_sorted16b = rus_tweets_sorted[9285:14284]
#rus_tweets_sorted17 = rus_tweets_sorted[14285:14285]
rus_tweets_sorted18 = rus_tweets_sorted[14286:14317]
#rus_tweets_sorted19 = rus_tweets_sorted[14318:14318]
rus_tweets_sorted20 = rus_tweets_sorted[14319:14469]
rus_tweets_sorted21 = rus_tweets_sorted[14470:15814]
rus_tweets_sorted22 = rus_tweets_sorted[15815:15899]
rus_tweets_sorted23 = rus_tweets_sorted[15900:15902]
rus_tweets_sorted24 = rus_tweets_sorted[15903:15939]
rus_tweets_sorted25 = rus_tweets_sorted[15940:15946]
rus_tweets_sorted26 = rus_tweets_sorted[15940:15946]
rus_tweets_sorted27 = rus_tweets_sorted[15940:15946]
rus_tweets_sorted28 = rus_tweets_sorted[15940:15946]


# In[31]:


# find Lev distance for the user #1

dist = np.empty(rus_tweets_sorted1.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted1, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted1.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[32]:


# find Lev distance for the user #2

dist = np.empty(rus_tweets_sorted2.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted2, repeat=2)):
      dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted2.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[33]:
```

```
# find Lev distance for the user #3

dist = np.empty(rus_tweets_sorted3.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted3, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted3.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[34]:


# find Lev distance for the user #4

dist = np.empty(rus_tweets_sorted4.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted4, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted4.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[35]:


# find Lev distance for the user #5

dist = np.empty(rus_tweets_sorted5.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted5, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted5.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[36]:


# find Lev distance for the user #6

dist = np.empty(rus_tweets_sorted6.shape[0]**2, dtype=int)
```

```
for i, x in enumerate(product(rus_tweets_sorted6, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted6.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[37]:


# find Lev distance for the user #7

dist = np.empty(rus_tweets_sorted7.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted7, repeat=2)):
      dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted7.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[38]:


# find Lev distance for the user 8

dist = np.empty(rus_tweets_sorted8.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted8, repeat=2)):
      dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted8.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[39]:


# find Lev distance for the user 9

dist = np.empty(rus_tweets_sorted9.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted9, repeat=2)):
      dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted9.shape[0]))
mean_dist = dist_df.mean()
```

```
mean_dist.mean()


# In[40]:


# find Lev distance for the user 10

dist = np.empty(rus_tweets_sorted10.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted10, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted10.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[41]:


# find Lev distance for the user 11

dist = np.empty(rus_tweets_sorted11.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted11, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted11.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[42]:


# find Lev distance for the user 12

dist = np.empty(rus_tweets_sorted12.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted12, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted12.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[43]:
```

```
# find Lev distance for the user13

dist = np.empty(rus_tweets_sorted13.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted13, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted13.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[44]:


# find Lev distance for the user14

dist = np.empty(rus_tweets_sorted14.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted14, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted14.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[45]:


# find Lev distance for the user15

dist = np.empty(rus_tweets_sorted15.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted15, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted15.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[58]:


# find Lev distance for the user16a

dist = np.empty(rus_tweets_sorted16a.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted16a, repeat=2)):
```

```
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted16a.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[59]:


# find Lev distance for the user16b

dist = np.empty(rus_tweets_sorted16b.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted16b, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted16b.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[60]:


# Lev for 16
(98.90660460926716 + 99.0721426541758)/2


# In[47]:


# find Lev distance for the user17

dist = np.empty(rus_tweets_sorted17.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted17, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted17.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[48]:


# find Lev distance for the user18
```

```
dist = np.empty(rus_tweets_sorted18.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted18, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted18.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[49]:


# find Lev distance for the user19

dist = np.empty(rus_tweets_sorted19.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted19, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted19.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[50]:


# find Lev distance for the user20

dist = np.empty(rus_tweets_sorted20.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted20, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted20.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[51]:


# find Lev distance for the user21

dist = np.empty(rus_tweets_sorted21.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted21, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted21.shape[0]))
```

```
mean_dist = dist_df.mean()
mean_dist.mean()


# In[52]:


# find Lev distance for the user22

dist = np.empty(rus_tweets_sorted22.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted22, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted22.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[53]:


# find Lev distance for the user23

dist = np.empty(rus_tweets_sorted23.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted23, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted23.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[54]:


# find Lev distance for the user24

dist = np.empty(rus_tweets_sorted24.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted24, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted24.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[55]:
```

```
# find Lev distance for the user25

dist = np.empty(rus_tweets_sorted25.shape[0]**2, dtype=int)
for i, x in enumerate(product(rus_tweets_sorted25, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, rus_tweets_sorted25.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()


# In[128]:


# Real Tweets Data Set
real_tweets = pd.read_csv('/Users/Phillip/Downloads/cresci-
2017/datasets_full.csv/genuine_accounts.csv/tweets.csv', na_filter=False)

real_tweets['user_id'] = real_tweets['user_id'].str.replace(' ','')
real_tweets.fillna('')
real_tweets.info()


# In[90]:


# real_tweets['user_id'] = real_tweets['user_id'].astype(str).astype(int)

# real_tweets['user_id'].tail()
#real_tweets.info()

real_tweets_sorted = real_tweets.sort_values(by=['user_id'])

real_tweets_sorted = real_tweets_sorted['text']
real_tweets_sorted = real_tweets_sorted.str.replace(' ','')
real_tweets_sorted = real_tweets_sorted.str.replace('RT@','')


# In[130]:




# In[93]:
```

```
real_tweets_sorted.head()


# In[94]:


real_tweets_sorted1 = real_tweets_sorted[981668:982560]
real_tweets_sorted2 = real_tweets_sorted[982561:985677]
real_tweets_sorted3 = real_tweets_sorted[985678:988132]
real_tweets_sorted4 = real_tweets_sorted[988133:991368]
real_tweets_sorted5 = real_tweets_sorted[991369:994578]
real_tweets_sorted6 = real_tweets_sorted[994579:997773]
real_tweets_sorted7 = real_tweets_sorted[997774:1000994]
real_tweets_sorted8 = real_tweets_sorted[1000995:1004171]
real_tweets_sorted9 = real_tweets_sorted[1004172:1007390]
real_tweets_sorted10 = real_tweets_sorted[1007391:1010510]
real_tweets_sorted11 = real_tweets_sorted[1010511:1013599]
real_tweets_sorted12 = real_tweets_sorted[1013600:1013941]
real_tweets_sorted13 = real_tweets_sorted[1013942:1017137]
real_tweets_sorted14 = real_tweets_sorted[1017138:1019436]
real_tweets_sorted15 = real_tweets_sorted[1019437:1022622]
real_tweets_sorted16 = real_tweets_sorted[1022623:1025845]
real_tweets_sorted17 = real_tweets_sorted[1025846:1029038]
real_tweets_sorted18 = real_tweets_sorted[1029039:1032277]
real_tweets_sorted19 = real_tweets_sorted[1032278:1035441]
real_tweets_sorted20 = real_tweets_sorted[1035442:1036606]
real_tweets_sorted21 = real_tweets_sorted[1036607:1039781]
real_tweets_sorted22 = real_tweets_sorted[1039782:1042953]
real_tweets_sorted23 = real_tweets_sorted[1042954:1045336]
real_tweets_sorted24 = real_tweets_sorted[1045337:1045417]
real_tweets_sorted25 = real_tweets_sorted[1045418:1048574]


# In[95]:


# find Lev distance for the user #1

dist = np.empty(real_tweets_sorted1.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted1, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted1.shape[0]))
mean_dist = dist_df.mean()
mean_dist.mean()
```

```
# In[96]:


# find Lev distance for the user #2

dist = np.empty(real_tweets_sorted2.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted2, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted2.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[97]:


# find Lev distance for the user #3

dist = np.empty(real_tweets_sorted3.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted3, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted3.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[121]:


# find Lev distance for the user #4

dist = np.empty(real_tweets_sorted4.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted4, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted4.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()
```

```
# In[99]:


# find Lev distance for the user #5

dist = np.empty(real_tweets_sorted5.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted5, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted5.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[100]:


# find Lev distance for the user #6

dist = np.empty(real_tweets_sorted6.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted6, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted6.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[101]:


# find Lev distance for the user #7

dist = np.empty(real_tweets_sorted7.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted7, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted7.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[102]:
```

```
# find Lev distance for the user #8

dist = np.empty(real_tweets_sorted8.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted8, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted8.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[103]:


# find Lev distance for the user #9

dist = np.empty(real_tweets_sorted9.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted9, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted9.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[104]:


# find Lev distance for the user #10

dist = np.empty(real_tweets_sorted10.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted10, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted10.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[105]:


# find Lev distance for the user #11
```

```
dist = np.empty(real_tweets_sorted11.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted11, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted11.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[106]:


# find Lev distance for the user #12

dist = np.empty(real_tweets_sorted12.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted12, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted12.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[107]:


# find Lev distance for the user #13

dist = np.empty(real_tweets_sorted13.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted13, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted13.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[108]:


# find Lev distance for the user #13

dist = np.empty(real_tweets_sorted13.shape[0]**2, dtype=int)
```

```
for i, x in enumerate(product(real_tweets_sorted13, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted13.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[109]:


# find Lev distance for the user #14

dist = np.empty(real_tweets_sorted14.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted14, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted14.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[110]:


# find Lev distance for the user #15

dist = np.empty(real_tweets_sorted15.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted15, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted15.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[111]:


# find Lev distance for the user #16

dist = np.empty(real_tweets_sorted16.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted16, repeat=2)):
    dist[i] = editdistance.eval(*x)
```

```
dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted16.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[112]:


# find Lev distance for the user #17
dist = np.empty(real_tweets_sorted17.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted17, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted17.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[113]:


# find Lev distance for the user #18

dist = np.empty(real_tweets_sorted18.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted18, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted18.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[114]:


# find Lev distance for the user #19

dist = np.empty(real_tweets_sorted19.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted19, repeat=2)):
    dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted19.shape[0]))
```

```
mean_dist = dist_df.mean()
mean_dist.mean()


# In[115]:


# find Lev distance for the user #20

dist = np.empty(real_tweets_sorted20.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted20, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted20.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[116]:


# find Lev distance for the user #21

dist = np.empty(real_tweets_sorted21.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted21, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted21.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[117]:


# find Lev distance for the user #22

dist = np.empty(real_tweets_sorted22.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted22, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted22.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()
```

```
# In[118]:


# find Lev distance for the user #23

dist = np.empty(real_tweets_sorted23.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted23, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted23.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[119]:


# find Lev distance for the user #24

dist = np.empty(real_tweets_sorted24.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted24, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted24.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()


# In[120]:


# find Lev distance for the user #25

dist = np.empty(real_tweets_sorted25.shape[0]**2, dtype=int)
for i, x in enumerate(product(real_tweets_sorted25, repeat=2)):
        dist[i] = editdistance.eval(*x)

dist_df = pd.DataFrame(dist.reshape(-1, real_tweets_sorted25.shape[0]))

mean_dist = dist_df.mean()
mean_dist.mean()
```

```
# In[123]:


os.chdir("/Users/Phillip/Downloads/cresci-2017/datasets_full.csv/")
os.getcwd()
os.listdir()


# In[124]:


# need genuine accounts for support vector machining

real = pd.read_csv('genuine_accounts.csv/users.csv')
real = real.fillna('')
real.info()


# In[126]:


real = real.sort_values(by=['id'])
real.tail()


# In[75]:


df = pd.read_csv('social_spambots_2.csv/tweets.csv')
df = df.fillna('')
df.info()


# In[78]:


df = pd.read_csv('social_spambots_2.csv/tweets.csv')
df = df.fillna('')
#df['default_profile'].isnull().values.sum()

# need genuine accounts for support vector machining

real = pd.read_csv('genuine_accounts.csv/tweets.csv')
real = real.fillna('')

# temp. subset for testing SVM
```

```
# real = real[1:1000]

# fake followers
fake_followers = pd.read_csv('fake_followers.csv/tweets.csv')
fake_followers.fillna('')


# traditional spambots

trad_spam_1 = pd.read_csv('social_spambots_1.csv/tweets.csv')


trad_spam_1 = trad_spam_1.fillna('')


# social spambots

social_spam_1 = pd.read_csv('social_spambots_1.csv/tweets.csv')
social_spam_1 = social_spam_1.fillna('')

social_spam_2 = pd.read_csv('social_spambots_2.csv/tweets.csv')
social_spam_2 = social_spam_2.fillna('')


social_spam_3 = pd.read_csv('social_spambots_3.csv/tweets.csv')
social_spam_3 = social_spam_3.fillna('')

rus_tweets.fillna('')
rus_tweets = rus_tweets.replace(np.nan, '', regex=True)

# column detailing if they are a bot
# will be deleted later for SVM
real['knownbot'] = 0
df['knownbot'] = 1
fake_followers['knownbot'] = 1
trad_spam_1['knownbot'] = 1

social_spam_1['knownbot'] = 1
social_spam_2['knownbot'] = 1
social_spam_3['knownbot'] = 1
rus_tweets['knownbot'] = 1

#len(real['default_profile'])


# In[38]:
```

```
# combine dataframe. append dataframes.

# combine all social spambots
#all_trad_spam = pd.concat([trad_spam_1,trad_spam_2,trad_spam_3])

#all_social_spambots = pd.concat([social_spam_1,social_spam_2,social_spam_3])

#all_bots = pd.concat([all_social_spambots,fake_followers])

# df = pd.concat([df,real])

#df = pd.concat([real, rus_users])

#len(df['default_profile'])

df = shuffle(df)
df.info()


# In[ ]:


# # Average number of Tweets
# ss1 = social_spam_1['num_hashtags'].mean()
# ss2 = social_spam_2['num_hashtags'].mean()
# ss3 = social_spam_3['num_hashtags'].mean()
# ts1 = trad_spam_1['num_hashtags'].mean()
# #ts2 = trad_spam_2['num_hashtags'].mean()
# #ts3 = trad_spam_3['num_hashtags'].mean()
# r1 = real['num_hashtags'].mean()
# f1 = fake_followers['num_hashtags'].mean()
# #rus1 = rus_tweets['num_hashtags'].mean()


# sets = [ss1,ss2,ss3,ts1,ts2,ts3,r1,f1]

# # xlabel = ('Social 1', 'Social 2', 'Social 3', 'Traditional 1', 'Real Accounts', 'Fake
Followers', 'Russian Bots')
# xlabel = ('Social 1', 'Social 2', 'Social 3', 'Traditional 1', 'Real Accounts', 'Fake
Followers')
# ypos = np.arange(len(sets))
# amount = [ss1,ss2,ss3,ts1,r1,f1]

# plt.bar(xlabel, sets, align='center', alpha=0.5)
# plt.xticks(ypos,xlabel,rotation=30)
# plt.ylabel('Average Number of Favorites')
```

```
# plt.title('Average Number of Favorites Per Account Per Dataset')
# plt.show()


# In[ ]:


# # convert timestamp to datetime format

# # month/day/year hour:minute:second AM

# df['timestamp'] = df['timestamp'].apply(lambda x: dt.datetime.strptime(x,'%b%d%Y:%H:%M:%S.%f'))

# # df['Mycol'] = df['Mycol'].apply(lambda x: dt.datetime.strptime(x,'%d%b%Y:%H:%M:%S.%f'))
# df.info()


# In[ ]:


# # create empty DF and add id
# score = pd.DataFrame()
# score['id'] = df['id']


# In[ ]:


# #function that will be used for scoring

# # is language english
# def scoring (row):
#     if row['lang'] == 'en':
#         return 1
#     else:
#         return 0

# # function is applied
# df.apply (lambda row: scoring (row),axis=1)

# #output of function applied to rows is assigned to df collumn
# df['score'] = df.apply (lambda row: scoring (row),axis=1)

# # no null values in new score collumn (this collumn could be part of a new df)
# df['score'].isnull().values.sum()
```

```
# # assigns function output to new df
# score['lang-en'] = df.apply (lambda row: scoring (row),axis=1)

# coding: utf-8

# In[396]:


import pandas as pd
import numpy as np
import os

import matplotlib.pyplot as plt
import Levenshtein as Lev
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
from sklearn.utils import shuffle
import datetime as dt

from mlxtend.plotting import plot_decision_regions
from itertools import product


# In[397]:


# Russian Data Set
rus_users = pd.read_csv('/Users/Phillip/Downloads/RussianData/users.csv',
na_filter=False)
rus_users.fillna('')
#rus_users.rename(columns={})
#rus_users['knownbot'] = 1
#list(rus_users)

rus_users[['id','followers_count','statuses_count','favourites_count','friends_count']] =
rus_users[['id','followers_count','statuses_count','favourites_count','friends_count']].ap
ply(pd.to_numeric)
rus_users['id'] = rus_users['id'].fillna(0).astype(int)
rus_users['followers_count'] = rus_users['followers_count'].fillna(0).astype(int)
rus_users['statuses_count'] = rus_users['statuses_count'].fillna(0).astype(int)
rus_users['favourites_count'] = rus_users['favourites_count'].fillna(0).astype(int)
rus_users['friends_count'] = rus_users['friends_count'].fillna(0).astype(int)
#rus_users = rus_users.replace(np.nan, '', regex=True)
#rus_users[('followers_count','statuses_count','favourites_count','friends_count')].appl
ymap(int)
rus_users.fillna('')
```

```
#rus_users = rus_users.replace(np.nan, '', regex=True)
#int(rus_users['followers_count'])

#rus_users.shape
rus_users.info()


# In[398]:


os.chdir("/Users/Phillip/Downloads/cresci-2017/datasets_full.csv/")
os.getcwd()
os.listdir()


# In[399]:


df = pd.read_csv('social_spambots_1.csv/users.csv')
df.info()


# In[400]:


df = df.fillna('')
df.ix[:5,:20]


# In[401]:


list(df)
df.info()


# In[402]:


df['default_profile'].isnull().values.sum()
len(df['default_profile'])
df.head()


# In[403]:
```

```
# need genuine accounts for support vector machining

real = pd.read_csv('genuine_accounts.csv/users.csv')
real = real.fillna('')

# temp. subset for testing SVM

# real = real[1:1000]

# fake followers
fake_followers = pd.read_csv('fake_followers.csv/users.csv')
fake_followers.fillna('')


# traditional spambots

trad_spam_1 = pd.read_csv('traditional_spambots_1.csv/users.csv')
trad_spam_2 = pd.read_csv('traditional_spambots_2.csv/users.csv')
trad_spam_3 = pd.read_csv('traditional_spambots_3.csv/users.csv')
#trad_spam_4 = pd.read_csv('social_spambots_4.csv/users.csv')

trad_spam_1 = trad_spam_1.fillna('')
trad_spam_2 = trad_spam_2.fillna('')
trad_spam_3 = trad_spam_3.fillna('')
#trad_spam_4 = trad_spam_4.fillna('')

# social spambots

social_spam_1 = pd.read_csv('social_spambots_1.csv/users.csv')
social_spam_1 = social_spam_1.fillna('')

social_spam_2 = pd.read_csv('social_spambots_2.csv/users.csv')
social_spam_2 = social_spam_2.fillna('')


social_spam_3 = pd.read_csv('social_spambots_3.csv/users.csv')
social_spam_3 = social_spam_3.fillna('')

rus_users.fillna('')
rus_users = rus_users.replace(np.nan, '', regex=True)

# column detailing if they are a bot
# will be deleted later for SVM
real['knownbot'] = 0
df['knownbot'] = 1
fake_followers['knownbot'] = 1
trad_spam_1['knownbot'] = 1
```

```
trad_spam_2['knownbot'] = 1
trad_spam_3['knownbot'] = 1
#trad_spam_4['knownbot'] = 1
social_spam_1['knownbot'] = 1
social_spam_2['knownbot'] = 1
social_spam_3['knownbot'] = 1
rus_users['knownbot'] = 1

len(real['default_profile'])



# In[404]:


# Number of Twitter Accounts Per Dataset

ss1 = len(social_spam_1)
ss2 = len(social_spam_2)
ss3 = len(social_spam_3)
ts1 = len(trad_spam_1)
ts2 = len(trad_spam_2)
ts3 = len(trad_spam_3)
r1 = len(real)
f1 = len(fake_followers)
rus1 = len(rus_users)

# # Average number of followers
# ss1 = social_spam_1['followers_count'].mean()
# ss2 = social_spam_2['followers_count'].mean()
# ss3 = social_spam_3['followers_count'].mean()
# ts1 = trad_spam_1['followers_count'].mean()
# ts2 = trad_spam_2['followers_count'].mean()
# ts3 = trad_spam_3['followers_count'].mean()
# r1 = real['followers_count'].mean()
# f1 = fake_followers['followers_count'].mean()
# rus1 = rus_users['followers_count'].mean()

# # Average number of friends
# ss1 = social_spam_1['friends_count'].mean()
# ss2 = social_spam_2['friends_count'].mean()
# ss3 = social_spam_3['friends_count'].mean()
# ts1 = trad_spam_1['friends_count'].mean()
# ts2 = trad_spam_2['friends_count'].mean()
# ts3 = trad_spam_3['friends_count'].mean()
# r1 = real['friends_count'].mean()
```

```
# f1 = fake_followers['friends_count'].mean()
# rus1 = rus_users['friends_count'].mean()

# # Average number of Tweets
# ss1 = social_spam_1['statuses_count'].mean()
# ss2 = social_spam_2['statuses_count'].mean()
# ss3 = social_spam_3['statuses_count'].mean()
# ts1 = trad_spam_1['statuses_count'].mean()
# ts2 = trad_spam_2['statuses_count'].mean()
# ts3 = trad_spam_3['statuses_count'].mean()
# r1 = real['statuses_count'].mean()
# f1 = fake_followers['statuses_count'].mean()
# rus1 = rus_users['statuses_count'].mean()

# # Average number of Favorites Per Dataset
# ss1 = social_spam_1['favourites_count'].mean()
# ss2 = social_spam_2['favourites_count'].mean()
# ss3 = social_spam_3['favourites_count'].mean()
# ts1 = trad_spam_1['favourites_count'].mean()
# ts2 = trad_spam_2['favourites_count'].mean()
# ts3 = trad_spam_3['favourites_count'].mean()
# r1 = real['favourites_count'].mean()
# f1 = fake_followers['favourites_count'].mean()
# rus1 = rus_users['favourites_count'].mean()


sets = [ss1,ss2,ss3,ts1,ts2,ts3,r1,f1,rus1]

xlabel = ('Social 1', 'Social 2', 'Social 3', 'Traditional 1', 'Traditional 2', 'Traditional 3',
'Real Accounts', 'Fake Followers', 'Russian Bots')
ypos = np.arange(len(sets))
amount = [ss1,ss2,ss3,ts1,ts2,ts3,r1,f1,rus1]

plt.bar(xlabel, sets, align='center', alpha=0.5)
plt.xticks(ypos,xlabel,rotation=30)
# plt.ylabel('Average Number of Followers')
# plt.title('Average Number of Followers Per Account Per Dataset')
plt.ylabel('Number of Twitter Accounts')
plt.title(' Number of Twitter Accounts Per Dataset')
plt.show()

print(sets)
```

```
# In[405]:


# combine dataframe. append dataframes.

# combine all social spambots
all_trad_spam = pd.concat([trad_spam_1,trad_spam_2,trad_spam_3])

all_social_spambots = pd.concat([social_spam_1,social_spam_2,social_spam_3])

all_bots = pd.concat([all_trad_spam,all_social_spambots,fake_followers])

# df = pd.concat([df,real])

df = pd.concat([real, all_bots])

len(df['default_profile'])
df.head()


# In[406]:


# Prepare of Levenshtein Distance

LevD_Rus =
[87.22685185185186,75.6484375,93.87158203125,95.54127100073046,84.86111111
11111,88.12097304404995,85.50719999999995,0.0,93.0051775147929,92.93727330
653853,92.25777777777778,87.29300567107751,78.875,88.66128729592792,98.989
37363172149,85.52549427679502,88.27911111111112,92.20204768105165,94.7046
4852607702,34.5,87.5246913580247,76.22222222222223]

LevD_Real =
[65.56188793259464,68.06269743639605,80.99069695768078,35.81521735079752,
41.59524071487558,54.80695468844404,69.17062343273791,77.40079028640484,6
7.02438312151091,69.26917965276283,55.803792341740724,59.65946285291664,7
0.02652325009018,74.53508708143696,82.96344570432927,72.94090538318754,75
.49669423401855,86.23952096036828,59.65265730087915,57.51488084694314,56.
52436582043217,59.63556497551856,77.69237444844183,87.1303125,81.64039747
253511]

# sort real dataset

real = real.sort_values(by=['screen_name'])
real_Lev = real.tail(25)
real_Lev['LevD'] = LevD_Real
```

```
# sort Russian bots dataset

rus_users = rus_users.sort_values(by=['screen_name'])
rus_users_Lev = rus_users.iloc[3:28]
rus_users_Lev = rus_users_Lev.drop(rus_users_Lev.index[18])
rus_users_Lev = rus_users_Lev.drop(rus_users_Lev.index[16])
rus_users_Lev = rus_users_Lev.drop(rus_users_Lev.index[6])
rus_users_Lev['LevD'] = LevD_Rus

df = pd.concat([real_Lev, rus_users_Lev])

real_Lev['LevD'].describe()


# In[407]:


df = shuffle(df)
df.head()


# In[408]:


#function that will be used for scoring

# is language english
def scoring (row):
    if row['lang'] == 'en':
        return 1
    else:
        return 0


# In[409]:


# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()
```

```
# In[410]:


# create empty DF and add id
score = pd.DataFrame()
score['id'] = df['id']

# assigns function output to new df
score['lang-en'] = df.apply (lambda row: scoring (row),axis=1)


# In[411]:


score['id']


# In[412]:


# has profile image.
# change from using profile_banner_url

def scoring (row):
    if row['profile_image_url'] == '':
        return 1
    else:
        return 0

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['profile_pic'] = df.apply (lambda row: scoring (row),axis=1)

score['profile_pic'].tail()


# In[413]:
```

```
# has screen name.
# change from screen_name to name. screen_name = @handle. name: can be changed, not required.

def scoring (row):
    if row['name'] == '':
        return 1
    else:
        return 0

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['has_screen_name'] = df.apply (lambda row: scoring (row),axis=1)

score['has_screen_name'].head()


# In[414]:


# has 30 followers

def scoring (row):
    if row['followers_count'] < 30:
        return 1
    else:
        return 0

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['30followers'] = df.apply (lambda row: scoring (row),axis=1)
```

```
score['30followers'].head()


# In[415]:


# is geolocalized

def scoring (row):
    if row['geo_enabled'] == '':
        return 1
    else:
        return 0

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['geoloc'] = df.apply (lambda row: scoring (row),axis=1)

score['geoloc'].head()


# In[416]:


# profile banner contains a link ('http') from profile_banner_url
# change to if the description contains

def scoring (row):
    if row['profile_banner_url'] == '':
        return 0
    else:
        return 1

# def scoring (row):
#     if 'http' not in row['description']:
#         return 0
#     elif row['description'] == '':
#         return 1
```

```
#       else:
#           return 1

# df['description'] = df['description']

# def scoring (row):
#       if row['description'] == ('http'):
#           return 0
#       else:
#           return 1

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['banner_link'] = df.apply (lambda row: scoring (row),axis=1)

score['banner_link'].head()


# In[417]:


# has done 50 tweets

def scoring (row):
    if row['statuses_count'] > 50:
        return 0
    else:
        return 1

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
```

```
score['50tweets'] = df.apply (lambda row: scoring (row),axis=1)

score['50tweets'].head()


# In[418]:


# 2* num followers >= # of friends

def scoring (row):
    if 2*row['followers_count'] >= row['friends_count']:
        return 0
    else:
        return 1

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['twice_num_followers'] = df.apply (lambda row: scoring (row),axis=1)

score['twice_num_followers'].head()


# In[419]:


# does not have 1000s of friends, spambot

def scoring (row):
    if row['friends_count'] > 1000:
        return 1
    else:
        return 0

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)
```

```
# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['1000friends'] = df.apply (lambda row: scoring (row),axis=1)

score['1000friends'].head()


# In[420]:


# sent less than 20 tweets, spambot

def scoring (row):
    if row['statuses_count'] < 20:
        return 1
    else:
        return 0

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['1000friends'] = df.apply (lambda row: scoring (row),axis=1)

score['1000friends'].head()


# In[421]:


# egg avatar, default profile image

def scoring (row):
    if row['default_profile_image'] == '':
        return 0
    else:
        return 1
```

```
# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['profile_pic'] = df.apply (lambda row: scoring (row),axis=1)

score['profile_pic'].head()


# In[422]:


# Never tweeted

def scoring (row):
    if row['statuses_count'] == 0:
        return 1
    else:
        return 0

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['NeverTweeted'] = df.apply (lambda row: scoring (row),axis=1)

score['NeverTweeted'].head()


# In[423]:


# 50:1 friends/followers

def scoring (row):
```

```
    if 50*row['followers_count'] <= row['friends_count']:
        return 1
    else:
        return 0

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['fifty_FriendsFollowersRatio'] = df.apply (lambda row: scoring (row),axis=1)

score['fifty_FriendsFollowersRatio'].head()


# In[424]:


# 100:1 friends/followers

def scoring (row):
    if 100*row['followers_count'] <= row['friends_count']:
        return 1
    else:
        return 0

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['hundred_FriendsFollowersRatio'] = df.apply (lambda row: scoring
(row),axis=1)

score['hundred_FriendsFollowersRatio'].head()
```

```
# In[425]:


# Beginning of next draft...

# profile contains a description

def scoring (row):
    if row['description'] == '':
        return 1
    else:
        return 0

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['has_description'] = df.apply (lambda row: scoring (row),axis=1)

score['has_description'].head()


# In[426]:


# known bot

def scoring (row):
    if row['knownbot'] == 1:
        return 1
    else:
        return 0

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()
```

```
# assigns function output to new df
score['knownbot'] = df.apply (lambda row: scoring (row),axis=1)

score['knownbot'].head()


# In[427]:


# Levenshtein Distance less than 30

def scoring (row):
    if row['LevD'] < 30:
        return 1
    else:
        return 0

# function is applied
df.apply (lambda row: scoring (row),axis=1)

#output of function applied to rows is assigned to df collumn
df['score'] = df.apply (lambda row: scoring (row),axis=1)

# no null values in new score collumn (this collumn could be part of a new df)
df['score'].isnull().values.sum()

# assigns function output to new df
score['Levenshtein'] = df.apply (lambda row: scoring (row),axis=1)

score['Levenshtein'].head()


# In[428]:


score.shape


# In[429]:



score.describe()


# In[430]:
```

```
# Following code is based from Eric Larson code for his Data Mining Class
#
https://github.com/eclarson/DataMiningNotebooks/blob/master/04.%20Logits%20and
%20SVM.ipynb

from sklearn.model_selection import ShuffleSplit

# we want to predict the X and y data as follows:
if 'knownbot' in score:
    y = score['knownbot'].values # get the labels we want
    del score['knownbot'] # get rid of the class label
    X = score.values # use everything else to predict!

    ## X and y are now numpy matrices, by calling 'values' on the pandas data
frames we
    #    have converted them into simple matrices to use with scikit learn


# to use the cross validation object in scikit learn, we need to grab an instance
#    of the object and set it up. This object will be able to split our data into
#    training and testing splits
num_cv_iterations = 3
num_instances = len(y)
cv_object = ShuffleSplit(n_splits=num_cv_iterations,
                                    test_size  = 0.2)

print(cv_object)


# In[431]:


# run logistic regression and vary some parameters
from sklearn.linear_model import LogisticRegression
from sklearn import metrics as mt

# first we create a reusable logisitic regression object
#   here we can setup the object with different learning parameters and constants
lr_clf = LogisticRegression(penalty='l2', C=1.0, class_weight=None) # get object

# now we can use the cv_object that we setup before to iterate through the
#    different training and testing sets. Each time we will reuse the logisitic
regression
#    object, but it gets trained on different data each time we use it.
```

```
iter_num=0
# the indices are the rows used for training and testing in each iteration
for train_indices, test_indices in cv_object.split(X,y):
    # I will create new variables here so that it is more obvious what
    # the code is doing (you can compact this syntax and avoid duplicating memory,
    # but it makes this code less readable)
    X_train = X[train_indices]
    y_train = y[train_indices]

    X_test = X[test_indices]
    y_test = y[test_indices]

    # train the reusable logisitc regression model on the training data
    lr_clf.fit(X_train,y_train)    # train object
    y_hat = lr_clf.predict(X_test) # get test set precitions

    # now let's get the accuracy and confusion matrix for this iterations of
training/testing
    acc = mt.accuracy_score(y_test,y_hat)
    conf = mt.confusion_matrix(y_test,y_hat)
    print("====Iteration",iter_num," ====")
    print("accuracy", acc )
    print("confusion matrix\n",conf)
    iter_num+=1

# Also note that every time you run the above code
#    it randomly creates a new training and testing set,
#    so accuracy will be different each time


# In[432]:


# interpret the weights

# iterate over the coefficients
weights = lr_clf.coef_.T # take transpose to make a column vector
variable_names = score.columns
for coef, name in zip(weights,variable_names):
    print(name, 'has weight of', coef[0])

# does this look correct?


# In[433]:
```

```
from sklearn.preprocessing import StandardScaler

# we want to normalize the features based upon the mean and standard deviation of
each column.
# However, we do not want to accidentally use the testing data to find out the mean
and std (this would be snooping)
# to Make things easier, let's start by just using whatever was last stored in the
variables:
##       X_train , y_train , X_test, y_test (they were set in a for loop above)

# scale attributes by the training set
scl_obj = StandardScaler()
scl_obj.fit(X_train) # find scalings for each column that make this zero mean and unit
std
# the line of code above only looks at training data to get mean and std and we can
use it
# to transform new feature data

X_train_scaled = scl_obj.transform(X_train) # apply to training
X_test_scaled = scl_obj.transform(X_test) # apply those means and std to the test set
(without snooping at the test set values)

# train the model just as before
lr_clf = LogisticRegression(penalty='l2', C=0.05) # get object, the 'C' value is less
(can you guess why??)
lr_clf.fit(X_train_scaled,y_train)    # train object

y_hat = lr_clf.predict(X_test_scaled) # get test set precitions

acc = mt.accuracy_score(y_test,y_hat)
conf = mt.confusion_matrix(y_test,y_hat)
print('accuracy:', acc )
print(conf )

# sort these attributes and spit them out
zip_vars = zip(lr_clf.coef_.T,score.columns) # combine attributes
zip_vars = sorted(zip_vars)
for coef, name in zip_vars:
      print(name, 'has weight of', coef[0]) # now print them out


# In[434]:


# now let's make a pandas Series with the names and values, and plot them
from matplotlib import pyplot as plt
get_ipython().magic('matplotlib inline')
```

```
plt.style.use('ggplot')


weights = pd.Series(lr_clf.coef_[0],index=score.columns)
weights.plot(kind='bar')
plt.show()


# In[435]:


from sklearn.preprocessing import StandardScaler
# we want to normalize the features based upon the mean and standard deviation of
each column.
# However, we do not want to accidentally use the testing data to find out the mean
and std (this would be snooping)

from sklearn.pipeline import Pipeline
# you can apply the StandardScaler function inside of the cross-validation loop
#    but this requires the use of PipeLines in scikit.
#    A pipeline can apply feature pre-processing and data fitting in one compact
notation
#    Here is an example!

std_scl = StandardScaler()
lr_clf = LogisticRegression(penalty='l2', C=0.05)

# create the pipline
piped_object = Pipeline([('scale', std_scl),    # do this
                                    ('logit_model', lr_clf)]) # and then do this

weights = []
# run the pipline cross validated
for iter_num, (train_indices, test_indices) in enumerate(cv_object.split(X,y)):
        piped_object.fit(X[train_indices],y[train_indices])    # train object
        # it is a little odd getting trained objects from a    pipeline:
        weights.append(piped_object.named_steps['logit_model'].coef_[0])


weights = np.array(weights)


# In[436]:


import plotly
plotly.offline.init_notebook_mode() # run at the start of every notebook
```

```
error_y=dict(
                type='data',
                array=np.std(weights,axis=0),
                visible=True
            )

graph1 = {'x': score.columns,
                'y': np.mean(weights,axis=0),
      'error_y':error_y,
           'type': 'bar'}

fig = dict()
fig['data'] = [graph1]
fig['layout'] = {'title': 'Logistic Regression Weights, with error bars'}

plotly.offline.iplot(fig)


# In[437]:



# # not sure if needed so haven't fixed yet

# Xnew = df_imputed[['Age','Pclass','IsMale']].values

# weights = []
# # run the pipline corssvalidated
# for iter_num, (train_indices, test_indices) in enumerate(cv_object.split(Xnew,y)):
#         piped_object.fit(Xnew[train_indices],y[train_indices])    # train object
#         weights.append(piped_object.named_steps['logit_model'].coef_[0])

# weights = np.array(weights)

# error_y=dict(
#                 type='data',
#                 array=np.std(weights,axis=0),
#                 visible=True
#             )

# graph1 = {'x': ['Age','Pclass','IsMale'],
#                 'y': np.mean(weights,axis=0),
#         'error_y':error_y,
#              'type': 'bar'}

# fig = dict()
```

```
# fig['data'] = [graph1]
# fig['layout'] = {'title': 'Logistic Regression Weights, with error bars'}

# plotly.offline.iplot(fig)


# In[438]:


# okay, so run through the cross validation loop and set the training and testing variable for one single iteration
for train_indices, test_indices in cv_object.split(X,y):
    # I will create new variables here so that it is more obvious what
    # the code is doing (you can compact this syntax and avoid duplicating memory,
    # but it makes this code less readable)
    X_train = X[train_indices]
    y_train = y[train_indices]

    X_test = X[test_indices]
    y_test = y[test_indices]

X_train_scaled = scl_obj.transform(X_train) # apply to training
X_test_scaled = scl_obj.transform(X_test)


# In[439]:


score.head()


# In[440]:


# lets investigate SVMs on the data and play with the parameters and kernels
from sklearn.svm import SVC

# train the model just as before
svm_clf = SVC(C=0.5, kernel='rbf', degree=3, gamma='auto') # get object
svm_clf.fit(X_train_scaled, y_train)    # train object

y_hat = svm_clf.predict(X_test_scaled) # get test set precitions

acc = mt.accuracy_score(y_test,y_hat)
conf = mt.confusion_matrix(y_test,y_hat)
print('accuracy:', acc )
print(conf)
```

```
# In[441]:


# SVM Without logistic regression
svm_clf = SVC(C=0.5, kernel='rbf', degree=3, gamma='auto') # get object
svm_clf.fit(X, y)    # train object
y_hat = svm_clf.predict(X) # get test set precitions
acc = mt.accuracy_score(y,y_hat)
conf = mt.confusion_matrix(y,y_hat)
print('accuracy:', acc )
print(conf)


# In[363]:


score.head()


# In[364]:


# look at the support vectors
print(svm_clf.support_vectors_.shape)
print(svm_clf.support_.shape)
print(svm_clf.n_support_ )


# In[365]:


# if using linear kernel, these make sense to look at (not otherwise, why?)
print(svm_clf.coef_)
weights = pd.Series(svm_clf.coef_[0],index=df_imputed.columns)
weights.plot(kind='bar')


# In[366]:


# Now let's do some different analysis with the SVM and look at the instances that
were chosen as support vectors

# now lets look at the support for the vectors and see if we they are indicative of
anything
```

```
# grabe the rows that were selected as support vectors (these are usually instances that
are hard to classify)

# make a dataframe of the training data
score_tested_on = score.iloc[train_indices] # saved from above, the indices chosen for
training
# now get the support vectors from the trained model
score_support = score_tested_on.iloc[svm_clf.support_,:]

score_support['knownbot'] = y[svm_clf.support_] # add back in the 'Survived'
Column to the pandas dataframe
score['knownbot'] = y # also add it back in for the original data
score_support.info()


# In[367]:


# now lets see the statistics of these attributes
from pandas.tools.plotting import boxplot

# group the original data and the support vectors
df_grouped_support = score_support.groupby(['knownbot'])
df_grouped = score.groupby(['knownbot'])

# plot KDE of Different variables
vars_to_plot = ['banner_link','profile_pic','has_screen_name','30followers']

for v in vars_to_plot:
    plt.figure(figsize=(10,4))
    # plot support vector stats
    plt.subplot(1,2,1)
    ax = df_grouped_support[v].plot.kde()
    plt.legend(['real','bot'])
    plt.title(v+' (Instances chosen as Support Vectors)')

    # plot original distributions
    plt.subplot(1,2,2)
    ax = df_grouped[v].plot.kde()
    plt.legend(['real','bot'])
    plt.title(v+' (Original)')



# In[145]:
```

```
# Levenshtein Tests

Lev.distance('Phillip Efthimion', '@RTscott_payne: Phillip Efthimion')


# In[699]:


tweets = pd.read_csv('social_spambots_2.csv/tweets.csv')
tweets = tweets.fillna('')
tweets['text'] = tweets['text'].astype(str)
tweets.info()

rus_users['followers_count'] = rus_users['followers_count'].fillna(0).astype(int)


# In[712]:


choices = tweets['text'][3:20]
process.extract(tweets['text'][2], choices, limit=2)
#process.extractOne(tweets['text'][1], choices)
```