Detection of Spammers in Twitter

Priya Narayana Subramanian

August 1 2015

Abstract

With the advent of social networking, online media has gained a lot of popularity. To the extent that mainstream media has taken a backseat. Users like to have instant updates on things and people that interest them. In this regard, social media are, in a way a tailor made electronic newspaper that people have access to round the clock. Twitter with its microblogging feature, is even more efficient in delivering news in a crisp and concise fashion. But, as with any technology, twitter too can be used in negative ways. One of the most common ways to exploit twitter is spam. Spammers propagate falsified information hidden behind an shortened URL, luring the users to open the web pages. These web pages might either have some irrelevant advertisements or might be a security threat which affects the user's system without his notice. One of the most common ways to get the users to click the URLs are by using trending topics. This research work identifies characteristics that can be used to identify such spammers in twitter. Ground truth is used to train a classifier which is then validated in two different ways. Also, the importance of attributes with regards to the decision making process are discussed.

1 Introduction and Prior Works

With the size of twitter and its user base, its almost inevitable to have a spam free environment. And spam is not just a false marketing or a virus attack. Things like profanity, insults, hate speech, fraudulent reviews, fake profiles constitutes a little part of an exhaustive list, if there exists one. Why twitter ? Twitter is more easy to spam because of its microblogging facility. If you can get people interested in 140 characters, then you can get them to click a link or buy a product even. As they say, curiosity kills. Any other social media has a larger content disposition which tends to give the user more than what is necessary and hence sometime reveals the trick. Effects of spam in twitter is not only realized by the end users. Imagine the amount of useless data twitter has to store, process and maintain. Its a waste of resource and time from twitter's perspective as well. But, Detection of spammers is not so easy after all. Although twitter has become better at eliminating spammers recently, still there are a lot of spammers that are hiding perfectly. This is because spammers evolve along with the spam detection strategies.

Trending tags are a major way in which spam propagates through the network. An instance of this would be using tags like "blackfriday" and "cybermonday" during thanksgiving time. More people are likely to be searching for deals and this gives an edge to spammers who can use this tag and lure an user into doing something that they wouldn't normally do. If an ad says "iPad for 100$, hurry up! goo.gl/sX7gIV", people would normally open the link. More than 99% of the internet users will not know the implications of hitting a bad URL. This can be detected and eliminated only to an extent, partly due to the high volume of tweets that flows through the network during this time.

There has been a lot of works going on recently to detect spammers. Some of them are "Detecting Spammers on Twitter" by F. Benevenuto and G. Magno's work[1], "CATS: Characterizing Automation of Twitter Spammers" by A. Amleshwaram and N. Reddy's paper[2]. The former one defines a set of 62 characteristics with which they train a classifier to detect spammers. The latter one differs in the types of characteristics in addition to understanding spam campaigns.

Our work can be split as,

1. Collection of Data. Ground Truth has been obtained from `http://homepages.dcc.ufmg.br/~fabricio/spammerscollection.html`, the author of the first paper. Similarly Ground truth for Non Spammers are collected. Lastly, users who tweeted about trending topics are identified as a test set.

2. Constructing the Classifier. Using weka, an opensource tool for machine learning, a classifier is generated from the groundtruth.

3. Analysis of the features. Out of the 58 features (we use only 58 of the 62), we analyze what features are more important in the decision making process.

2 Background and Dataset Collection

2.1 Background

Twitter is a social network where you can follow other persons to get their updates(tweets). Anyone from a friend to a rockstar can be followed, provided they are in the system. It also serves as a great source of news to a lot of people who seek news in twitter. Tweets are posts that are limited in length to 140 characters. You can reply to a tweet or retweet, which is basically sharing. Also you can tag another person while tweeting. This is called mentioning in twitter terminology. The # symbol followed by a word, called a hashtag, is used to mark topics in a Tweet. The hashtags play a major role in identifying current trends.

2.2 About the API

Twitter provides an API to tap into his data for research and other purposes. To use the API, one has to be registered in `dev.twitter.com` and have a valid twitter account. Tweepy is the python library that we have used in this project to get data from twitter. Twitter exposes about 1% of its data to the API, and the API has a rate limiting. There is a limit of 15 requests per 15 minutes. Almost all requests are rate limited. Because of this it is a timetaking process to collect data from twitter. For instance if you are trying to collect the follower list of a public figure who has 100000 followers, it is going to take 500 requests (each request allows a maximum of 200 tuples). Hence it gets difficult. This is the reason why only 58 features has been used from the dataset. With more time, it is possible to analyse such huge volumes of data. There's a streaming API which gets the current tweets across the world.

2.3 Data Collection

The Dataset of Ground truth has two parts. Spammers and Non Spammers. The former is obtained from the author of "Detection of Spammers in Twitter" [1]. For Non spammers, we collect the

followers and following of our own account which we manually know to be genuine, a total of 469 users. 58 features that are mentioned below in the report has been extracted for each of these users.

The current trends at the time of tweet collection were 'Thanksgiving', 'TellyExpressFaceOfTheYear', 'ThanksToStarWars', 'MomInspirations', 'tejpal', 'Jordan Hill', 'GarthBrooks'. Tweets from the streaming API were obtained, filtered on these terms and saved in JSON format. Unique user obejcts were collected from these tweets and the feature set is extracted for them as well. This will serve as one of the test set. Other test set would be a random percentage of the ground truth which is not used for building the classifier. Hence if 60% is the limit, then only 60% of the ground truth will be used in construction of the data set. The rest 40% is used as a test set.

3 Twitter users' features and Main Result

We evaluate 58 features of the pre-classified dataset of spammers and non-spammer users. The 58 features would be divided into two category: content attributes and user behavior attributes. Content attributes are properties of the text of tweets posted by users, which capture specific properties related to the way users write tweets. User behavior attributes capture specific properties of the user behavior in terms of the posting frequency, social interactions, and influence on the Twitter network.

3.1 Content Attributes

In total, we have 37 attributes related to content of the tweets. That is, number of URLs in each tweet, number of hashtags, number of words on each tweet, number of users mentioned in each tweet, number of character, count of numbers and so on. The list of attributes can be found in the appendix

3.2 User behavior Attributes

In total, we have 21 attributes about the user behavior. As assumed and according to the measurement of our dataset, spammers are more likely to have less followers, more followees, lower number of times the user was mentioned, lower number of times the user was replied to, lower number of times the user replied someone, lower number of followees of the users followers (since they are more likely to be spammers as well), more fixed time between tweets, and more fixed tweets posted per day and per week.

4 Detecting Spammers

In this section we go about how we train the classifier with our ground truth and how we use it to detect spammers. There are two different testing strategies which will be discussed. Also, the effectiveness of the features will be analyzed from the decision tree constructed to classify. Finally we talk about metrics such as correctness and false positive results.

4.1 Building and Training the Classifier

We use weka (waikato environment for knowledge analysis), an opensource machine learning tool. The input data is converted into .arff format and fed into the system. J48 classifier is constructed, which classifies the instance 99.217% correctly. The confusion matrix is showed in Table 1.

The table shows that 1056 out 1065 spammers has been represented correctly by the classifier. Similarly, 466 out of the 469 real users fit into the system. A total of 12 users are found as outliers.

Table 1: Classifier Confusion Matrix

Case	spammer	non spammer	classified as
1	1056	9	spammer
2	3	466	non spammer

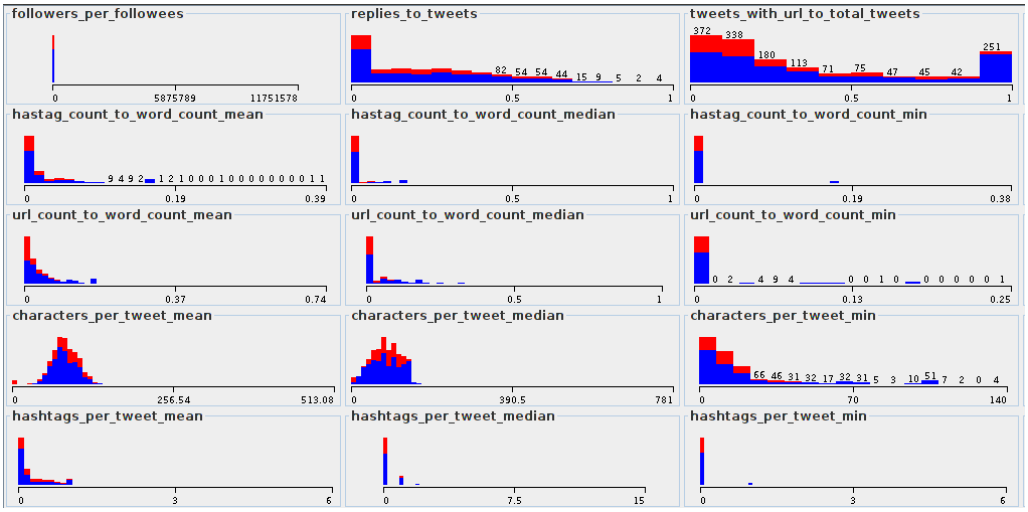

Figure 1: Feature vs Spammer Class

The classifier tries to find mean and median for each of the 58 features, and then picks up features such that it can delineate the dataset with minimal branches. If we take all possible paths the tree would be have a depth of P(58,2). Few examples of how each feature separates the ground truth can be viewed from the Figure 1 and Figure 2. Blue is Spammer, which remains predominantly at the bottom half of the feature range for most of the features, while red, the non spammer have higher values in most of the cases.

4.2 Results and Observations

Now there are two ways to test the classifier.

1. Introducing new test set. The new training set is obtained by observing the tweets from the Twitter Streaming API and filtering them on the latest trends. Using this tweet details 30 users were chosen and their features were extracted. 30 more users from a pre recognized spam list was added to this test set. This pre recognized list was obtained from internet `http://www.infochimps.com/datasets/twitter-spammers-list`. This feature set was converted into .arff file format and fed into the J48 classifier and the results were as follows. It turns out that 59 out of the 60 users are non spammers and one user was a spammer. The output of the classifier is shown in the Table.

   The expected result though should have been 30 spammers and 30 non spammers. We are unfortunate to not be able to know for sure if the result was correct. But when we looked at the one account that was reported to be a spammer, it was pretty inactive and did not look like a spammer at all. There's a high chance that there were no spammers in the set of 30 users obtained from the streaming API. 30 is a pretty small value if you consider the number

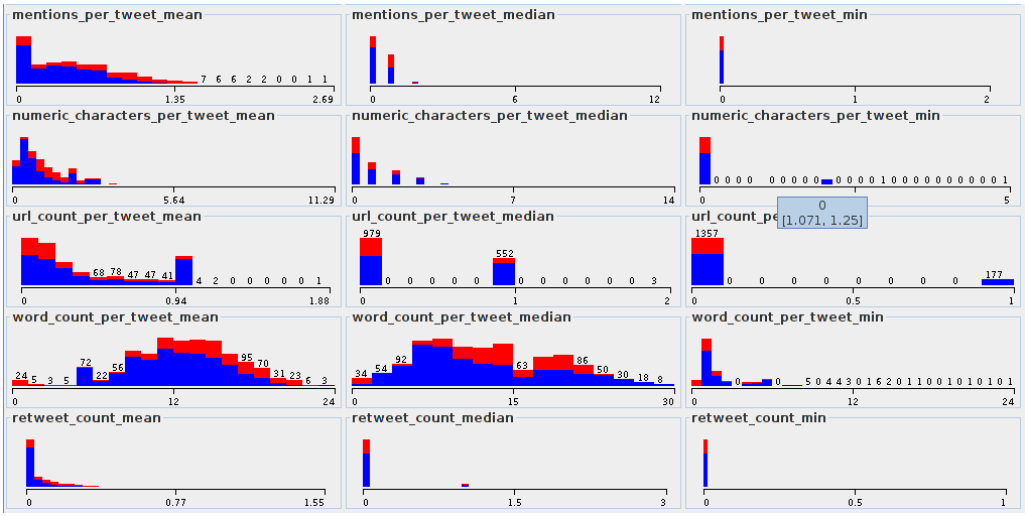

Figure 2: Feature vs Spammer Class

Table 2: Classifier Confusion Matrix for the Test set

Case	spammer	non spammer	classified as
1	0	0	spammer
2	1	59	non spammer

of user twitter has. However, the other 30 of the the 60 users are supposed to be spammers. But it turned out that the names found as spammers in the web provided list did not look like spammers either, when we tried to verify them manually. It could also be that, the classifier is not good enough to capture the latest trends in evolution of spammers. Hence, we test the classifier in another way. Also, a better way to get spammer test data is discussed later in the paper.

2. Using a part of the groundtruth. weka provides a test option called percentage split, where only a given percentage of groundtruth is used for the construction of classifier. The remaning part of the data is used to test the classifier. The part of the data is selected randomly for each run. This method was run for 10%, 50%, 66% and the results are shown in the table. The percentage of accuracy is 94.7864%, 99.3481%, 99.6169% respectively

Table 3: Classifier Confusion Matrix for the Test set at 10%

Case	spammer	non spammer	classified as
1	926	39	spammer
2	33	383	non spammer

The observation from the these values and from the table is that the data from ground truth is quite consistent and there are less abnormalities. That's why even a classifier trained with as less at 10% of data is able to give startling results.

Table 4: Classifier Confusion Matrix for the Test set at 50%

Case	spammer	non spammer	classified as
1	539	3	spammer
2	2	223	non spammer

Table 5: Classifier Confusion Matrix for the Test set at 66%

Case	spammer	non spammer	classified as
1	371	2	spammer
2	0	149	non spammer

4.3 Analysis of the Decision Tree and the Features

Figure 3 represents the decision tree used in classification. It is important to validate the decision tree with the data. One of the main attribute as defined by the classifier is the max time between posts. When it is above a certain value, the user always tend to be a spammer. And this tends to be true for 920 cases. Hence, almost 70% can be classified using this one feature. Also it can be observed that total tweets of friends is a good characteristic feature. It classifies 420 non spammers at level 2. Such attributes which partitions the data clearly are the most important ones. In this regard, the important features are,

1. maximum time between posts
2. follower count
3. mean post count per day
4. total tweet count of all friends
5. followee count
6. minimum post count per day
7. maximum character per tweet
8. maximum mentions per tweet

Hence a reduced feature set would also work as good as the original one. This is proposed due to the huge number of features which could get cumbersome in some cases.

5 Planned vs Happened

We started out to find a unsupervised way of detecting spammers, which takes into account the communication paths between spammers and real users. In an ideal scenario, no real user would interact with a spammer. The visualization was the interaction between two clusters, always being in one direction. And the source cluster could be identified as a spammer. This was to be tested across various spamming methodologies and see if it worked in all of them. And if it did, it would beat all

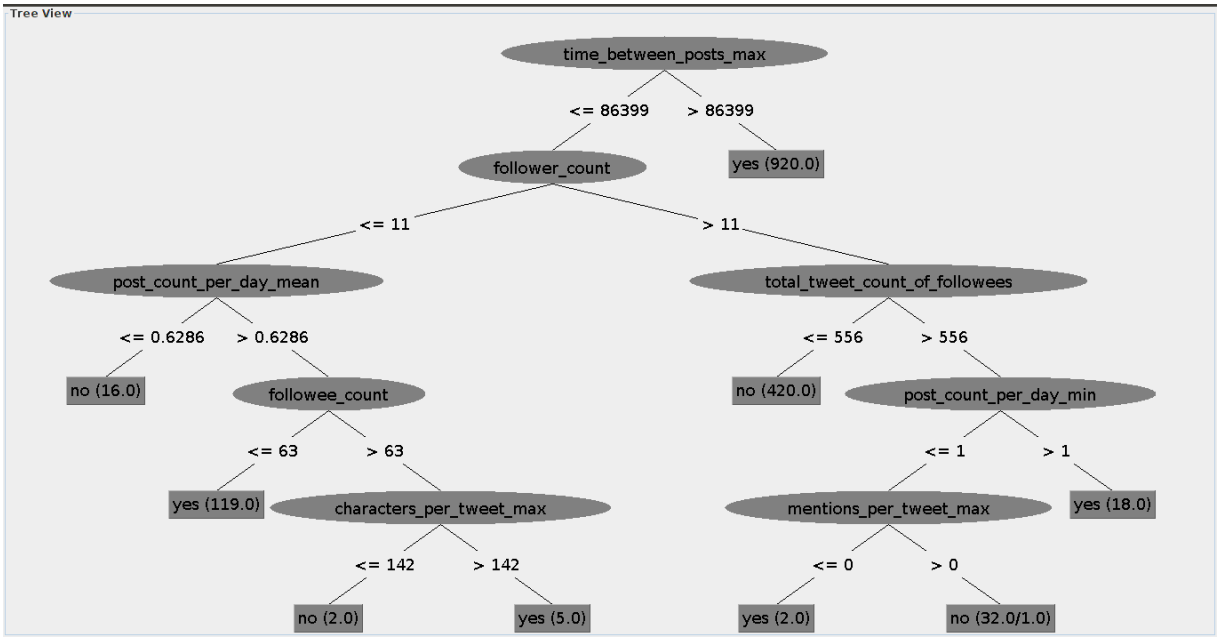

Figure 3: J48 Classifier

the spammer evolution techniques. But due to lack of time to collect enough data and the complexity of the task at hand, we had to settle with a supervised learning approach, which again was originally planned to be done with CATS paper. But since we did not get enough data (groundtruth), we had to move to this new paper. Once we got the data we needed, we collected some more on own and modelled a classifier successfully. Modelling of the classifier has been different from what has been mentioned in the original work. we decided to go with J48 since it seemed to suit our dataset much better. And we got more accuracy with J48 than the 70% accuracy mentioned in the paper.

6 Conclusion and Future Work

Due to the importance and indispensability of detecting and preventing Twitter spammers, Twitter corporation has been continuously working to keep the spammers away. However, Twitter spammers are evolving to hide from the identification techniques. A continuous understanding of how the spammers evolve through the network is necessary. Since, any feature can be worked around by the spammers, unsupervised learning looks like the way to move forward. Whatever the scheme be, spammers has to tweet to attain their goal. This creates a separate cluster of people who keeps tweeting to other ordinary user groups that does not follow the cluster. The direction of communication is one directional. This is an idea to identify a group of spammers.

The work we did in this project nailed it where it assumed that the current trends are the most exploited by twitter spammers. But the drawback was that we were not able to get enough data and literally no data on spammers. Here's an idea to get spammer data to test the classifier. Twitter nowadays are much faster in reacting to spam accounts and block them sometimes within 10 minutes from its creation. So, when data is collected from streaming API, instead of getting just the user objects, whole feature set can be extracted. This is because, once twitter blocks a person, API access to the person's info will not be available. Now, if the user is blocked at a later time, we can know for

sure that the user is a spammer. This can be done going forward. Another thing is, being able to get all features as mentioned in the research work. Code Snippets are attached in the end. The whole project can be seen in `https://github.com/littlecegian/twitter-spammer-detector`. This codebase has the main code with the code directory and the analysis under data_analytics directory. The 2 arff files are the main input data. output can be viewed in the report directory.

References

[1] Fabricio Benevenuto, Gabriel Magno, "Detecting Spammers on Twitter," *Annual Collaboration, Electronic messaging, Anti- abuse and Spam Conference*, 2010.

[2] A. A. Amleshwaram, N. Reddy, "Cats: Characterizing automation of twitter spammers" *International Conference on Communication System and Networks*, 2013.

Appendix A Data collected

We consider the following content attributes:

fraction of tweets with URLs,
number of hashtags per number of words on each tweet (mean),
number of hashtags per number of words on each tweet (median),
number of hashtags per number of words on each tweet (min),
number of hashtags per number of words on each tweet (max),
number of URLs per number of words on each tweet (mean),
number of URLs per number of words on each tweet (median),
number of URLs per number of words on each tweet (min),
number of URLs per number of words on each tweet (max),
number of characters per tweet (mean),
number of characters per tweet (median),
number of characters per tweet (min),
number of characters per tweet (max),
number of hashtags per tweet (mean),
number of hashtags per tweet (median),
number of hashtags per tweet (min),
number of hashtags per tweet (max),
number of mentions per tweet (mean),
number of mentions per tweet (median),
number of mentions per tweet (min),
number of mentions per tweet (max),
number of numeric characters per tweet (mean),
number of numeric characters per tweet (median),
number of numeric characters per tweet (min),
number of numeric characters per tweet (max),
number of URLs on each tweet (mean),
number of URLs on each tweet (median),
number of URLs on each tweet (min),
number of URLs on each tweet (max),
number of words per tweet (mean),
number of words per tweet (median),
number of words per tweet (min),
number of words per tweet (max),
number of times the tweet has been retweeted (mean),
number of times the tweet has been retweeted (median),
number of times the tweet has been retweeted (min),
number of times the tweet has been retweeted (max).

We consider the following user behavior attributes:

fraction of tweets replied,
number of followees,
number of followers,
number of tweets,
number of followees of a user's followers,
number of times mentioned,
number of times the user was replied,
number of times the user replied,
number of tweets of a user's followees,
time between posts (mean),
time between posts (median),
time between posts (min),
time between posts (max),
number of posted tweets per day (mean),
number of posted tweets per day (median),
number of posted tweets per day (min),
number of posted tweets per day (max),
number of posted tweets per week (mean),
number of posted tweets per week (median),
number of posted tweets per week (min),
number of posted tweets per week (max).

Appendix B CODE

Appendix B.1 streaming API

```
# authentication part is done here (not including for the sake of brevity)

def get_trends():
        stream_listener = StreamListener()
        streamer = tweepy.Stream(auth=auth, listener=stream_listener)
        setTerms = ['Thanksgiving', 'TellyExpressFaceOfTheYear', 'ThanksToStarWars',
            'MomInspirations', 'tejpal', 'Jordan Hill', 'GarthBrooks']
        response = json.dumps(streamer.filter(track = setTerms))
```

Output of this file is redirected to a json file from which user ids are extracted later.

Appendix B.2 REST API

This is where rest of the work is done. Since rate limiting is enabled, four different authentication tokens are used interchangeably so that lot more data can be obtained. Main function call happens at the end of the file where it invokes extract_user_info which then uses other functions to extract individual information.

```
from __future__ import division
```

```
import tweepy
import json
import simplejson
from array import *
from twitter import *
import jsonpickle
import inspect
from twython import Twython
import os
from tweepy.parsers import RawJsonParser
import pdb
import re
from dateutil import parser
import csv
import sys
import logging
import shutil
import numpy
import time

def get_auth(index=0):
    #auth values and tokens comes here

    consumer_key=consumer_key_array[index]
    consumer_secret=consumer_secret_array[index]
    access_token=access_token_array[index]
    access_token_secret=access_token_secret_array[index]

    auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_token, access_token_secret)
    return auth

class StreamListener(tweepy.StreamListener):
    def on_status(self, tweet):
        print "Ran on_status"

    def on_error(self, status_code):
        print >> sys.stderr, 'Encountered error with status code:', status_code
        return True

    def on_data(self, data):
        print data
        return True

user_ids = []

def print_rate_limits():
    #cleanup code
    limit = api.rate_limit_status()
    print limit['resources']['followers']
    print limit['resources']['friends']
    print limit['resources']["statuses"]["/statuses/mentions_timeline"]
```

```
def write_to_file(users_info, path):
    element_file = open(path, 'and')
    for user_info in users_info:
        element_file.write(json.dumps(user_info, sort_keys=True) + "\n")
    element_file.close()


def get_99_users(element_ids, peer_list):
    users_info = json.loads(api_raw.lookup_users(user_ids=element_ids))
    for user_info in users_info:
        peer_list.append(user_info)
    return peer_list

def write_peer_info(iterator, path):
    element_ids = []
    peer_list = []
    for element in iterator:
        dict_var = element.__getstate__()
        dict_var.pop("status", None)
        dict_var.pop("created_at", None)
        dict_var.pop("author", None)
        dict_var.pop("user", None)
        peer_list.append(json.dumps(dict_var))
    print len(peer_list)
    write_to_file(peer_list, path)
    return peer_list

def insert_into_csv(list, size=None):
    global csv_row
    if (len(list) == 0):
        csv_row.extend((0,0,0,0))
        return
    if size is None:
        size = len(list)
    list = sorted(list)
    csv_row.append(numpy.float64(sum(list))/size)
    csv_row.append(list[int(len(list)/2)])
    csv_row.append(list[0])
    csv_row.append(list[-1])

def extract_tweet_metrics(tweet_list):
    global csv_row
    total_string = ""
    hashtag_count_array = []
    hashtag_per_word_array = []
    url_count_array = []
    url_per_word_array = []
    tweet_size_array = []
    mention_count_array = []
    numeric_characters_count_array = []
    word_count_array = []
    retweet_count_array = []
    total_string = ""
```

```
    for tweet in tweet_list:
        try:
            total_string += " " + tweet["text"]
            word_count = len(re.findall(r"\w+", tweet["text"]))
            url_count =
                len(re.findall(r"((https?):((//)|(\\\\))+([\w\d:#@%/;$()~_?\+-=\\\.&](#!)?)*)",
                tweet["text"]))
            hashtag_count = len(re.findall(r"#\w+", tweet["text"]))
            tweet_size_array.append(len(tweet["text"]))
            hashtag_per_word_array.append(numpy.float64(hashtag_count)/word_count)
            hashtag_count_array.append(hashtag_count)
            url_per_word_array.append(numpy.float64(url_count)/word_count)
            url_count_array.append(url_count)
            mention_count_array.append(len(re.findall(r"@\w+", tweet["text"])))
            numeric_characters_count_array.append(len(re.findall(r"\d", tweet["text"])))
            word_count_array.append(word_count)
            retweet_count_array.append(len(re.findall(r"RT\s*@\w+", tweet["text"])))
        except Exception, ex:
            print "The tweet text is " + tweet["text"]
            raise

    insert_into_csv(hashtag_per_word_array)
    insert_into_csv(url_per_word_array)
    insert_into_csv(tweet_size_array)
    insert_into_csv(hashtag_count_array)
    insert_into_csv(mention_count_array)
    insert_into_csv(numeric_characters_count_array)
    insert_into_csv(url_count_array)
    insert_into_csv(word_count_array)
    insert_into_csv(retweet_count_array)

def extract_tweet_timing_metrics(tweet_list):
    global csv_row
    if (len(tweet_list) == 0):
        insert_into_csv(tweet_list)
        insert_into_csv(tweet_list)
        insert_into_csv(tweet_list)
        return

    intertweet_time_array = []
    daily_tweet_counts = []
    weekly_tweet_counts = []
    daily_tweet_count = 0
    weekly_tweet_count = 0
    start_date = parser.parse(str(tweet_list[0]["created_at"])).date()
    for index, tweet in enumerate(tweet_list):
        time_first = parser.parse(str(tweet["created_at"]))
        if index < len(tweet_list) - 1:
            time_next = parser.parse(str(tweet_list[index+1]["created_at"]))
            intertweet_time_array.append((time_first - time_next).seconds)
        if (time_first.date() == start_date): #same day, and hence same week
            daily_tweet_count += 1
            weekly_tweet_count += 1
```

```
            else:
                daily_tweet_counts.append(daily_tweet_count)
                daily_tweet_count = 1
                if (((start_date - time_first.date()).days > 7) or (time_first.weekday() >
                    start_date.weekday())):
                    weekly_tweet_counts.append(weekly_tweet_count)
                    weekly_tweet_count = 1
                else:
                    weekly_tweet_count += 1
                start_date = time_first.date()

        daily_tweet_counts.append(daily_tweet_count)
        weekly_tweet_counts.append(weekly_tweet_count)

        time_first = parser.parse(str(tweet_list[0]["created_at"]))
        time_last = parser.parse(str(tweet_list[-1]["created_at"]))
        day_count = (time_first - time_last).days
        insert_into_csv(intertweet_time_array)
        insert_into_csv(daily_tweet_counts, day_count)
        insert_into_csv(weekly_tweet_counts, round(numpy.float64(day_count)/7))


    def retrieve_tweets(user_id, directory):
        global csv_row
        tweet_file = open(directory + 'tweets.json', 'w')
        page_no = 1
        tweet_list = list()
        while page_no <= 16:
            tweets = json.loads(api_raw.user_timeline(id=user_id, page=page_no, count=200,
                include_rts=True))
            if tweets:
                for tweet in tweets:
                    tweet_list.append(tweet)
            else:
                break
            page_no += 1
        url_count = 0
        reply_count = 0
        for tweet in tweet_list:
            tweet_file.write(json.dumps(tweet) + '\n')
            try:
                if (re.search(r"((https?):((//)|(\\\\))+([\w\d:#@%/;$()~_?\+-=\\\.&](#!)?)*)",
                    tweet["text"])):
                    url_count += 1
                if (tweet["in_reply_to_status_id_str"]):
                    reply_count += 1
            except:
                print tweet["text"]
        csv_row.append(numpy.float64(reply_count)/len(tweet_list))
        csv_row.append(numpy.float64(url_count)/len(tweet_list))
        csv_row.append(reply_count)
        extract_tweet_metrics(tweet_list)
        tweet_file.close()
```

```
    return tweet_list

def get_friend_list(friends):
    friend_names = open("follower_names.txt", 'w')
    friend_ids = open("follower_ids.txt", 'w')
    for friend in friends:
        # print friend.id
        friend_ids.write(str(friend.id) + '\n')
        friend_names.write(friend.screen_name + '\n')
    friend_ids.close()
    friend_names.close()

def retrieve_friends_followers_and_mentions(user_id, directory):
    global csv_row
    print "before"
    print_rate_limits()
    # followers = tweepy.Cursor(api.followers, user_id=user_id, count=200).items()
    # get_friend_list(followers)
    friends = tweepy.Cursor(api.friends, user_id=user_id, count=200).items()
    mentions = tweepy.Cursor(api.mentions_timeline, user_id=user_id, count=200).items()
    print "after"
    print_rate_limits()
    write_peer_info(friends, directory + "friends.json")
    write_peer_info(mentions, directory + "mentions.json")

    mention_count = 0
    reply_count = 0
    for mention in mentions:
        mention_count += 1
        if(mention.in_reply_to_status_id_str):
            reply_count += 1

    # get_friend_list(friends)
    # return
    # csv_row.append(sum([follower.friends_count for follower in followers]))
    csv_row.extend((mention_count, reply_count))
    csv_row.append(sum([friend.statuses_count for friend in friends]))

def insert_user_info(user_info):
    global csv_row
    csv_row.append(user_info["friends_count"])
    csv_row.append(user_info["followers_count"])
    csv_row.append(user_info["statuses_count"])

def extract_user_info(users_info):
    global csv_row
    data = []
    fp = open('spammer-result.csv', 'w')
    writer = csv.writer(fp, delimiter=',')
    try:
        for user_info in users_info:
            try:
                print "user_id is " + str(user_info["id"])
```

```
                print "user_name is " + str(user_info["screen_name"])
                csv_row = []
                if (user_info["friends_count"] > 2900):
                    print "too many friends to pull"
                    continue
                if (user_info["protected"] == True):
                    print "user is protected"
                    continue
                # pickled = jsonpickle.encode(user_info)
                # user_json_object = json.loads(pickled)
                csv_row =
                    [numpy.float64(user_info["followers_count"])/user_info["friends_count"]]
                    #initializing the csv with the first feature
                user_json_string = json.dumps(user_info, indent=4, sort_keys=True)
                user_id = user_info["id"]
                directory = "data_collected/" + str(user_id) + "/"
                if not os.path.exists(directory):
                    os.makedirs(directory)
                else:
                    continue
                tweet_list = retrieve_tweets(user_id, directory) #extracts 36 features
                insert_user_info(user_info) #3 more features
                retrieve_friends_followers_and_mentions(user_id, directory)
                extract_tweet_timing_metrics(tweet_list)
                user_details = open(directory + 'user.json','w')
                user_details.write(user_json_string)
                user_details.close()
                csv_row.append(user_info["screen_name"])
                csv_row.append(user_info["id"])
                # data.append(csv_row)
                writer.writerow(csv_row)
            except Exception, ex:
                shutil.rmtree(directory)
                logging.exception("Exception happened!")
                fp.close()
                raise
    finally:
        fp.close()

def chunks(l, n): #Yield successive n-sized chunks from l.
    for i in xrange(0, len(l), n):
        yield l[i:i+n]

def get_user_ids_from_trends():
    users_ids = ""
    for json_line in open("trends.json"):
        if (json_line == '\n'):
            continue
        id = str(json.loads(json_line)["user"]["id"])
        if id not in users_ids:
            users_ids += id + "\n"
    random = open("anonymous.txt", "w")
    random.write(users_ids)
```

```
    random.close()
    exit

def read_file_into_array():
    return [int(name.strip()) for name in open('spammer_ids.txt')]

def get_user_features_for_non_spammers():
    screen_names = read_file_into_array()
    chunked_names = chunks(screen_names, 99)
    for id_set in chunked_names:
        users_info = json.loads(api_raw.lookup_users(user_ids=id_set))
        extract_user_info(users_info)

def get_user_ids_from_user_names():
    screen_names = [name.strip() for name in open('flat_spammer_list.txt')]
    # return
    # for name_list in chunked_names:
    users = json.loads(api_raw.lookup_users(screen_names=screen_names))
    user_ids = [str(user['id']) for user in users]
    # user_names = [user['screen_name'] for user in users]
    user_file = open("spammer_ids.txt", "w")
    user_file.write("\n".join(user_ids))
    user_file.close()
    exit

index = 0
while True:
    try:
        print "Trying auth " + str(index)
        auth = get_auth(index)
        api_raw = tweepy.API(auth, parser=RawJsonParser())
        api = tweepy.API(auth)

        # get_user_features_for_non_spammers()
        get_user_ids_from_user_names()
        # get_user_ids_from_trends()
        # break
    except tweepy.TweepError:
        print "Rate limit exceeded. Moving on to next appln"
        index += 1
        if (index == 4):
            print "sleeping for 15 minutes"
            time.sleep(60 * 15)
        index %= 4
    except Exception, ex:
        logging.exception("Something awful happened!")
        print ex

# print_rate_limits()
# extract_user_info([json.loads(api_raw.get_user(screen_name='littlecegian07'))])
```