Don't Take This Personally: Sentiment Analysis for Identification of "Subtweeting" on Twitter

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Noah Segal-Gould

Annandale-on-Hudson, New York
May, 2018

Abstract

The Oxford English Dictionary states that “subtweet” is defined as “(on the social media application Twitter) a post that refers to a particular user without directly mentioning them, typically as a form of furtive mockery or criticism.” Following the rapid growth and adoption of social networking websites like Twitter, sentiment analysis has garnered much research interest in recent years. To computationally identify and categorize opinions expressed in text, sentiment analysis of figurative language such as irony and sarcasm has garnered even more recent interest. In this project, I will treat the identification of subtweets as a sentiment analysis problem and utilize a novel approach for data collection and labeling, as well as Naive Bayes text classification. By identifying subtweets as they are posted online in real time, I will create a Twitter bot which archives and interacts with them. In addition to the paper, this project will be made available online with its source code and all data collected during its completion.

Contents

Dedication

I dedicate this senior project to @jack, who has willfully made numerous changes to Twitter which inevitably angered millions.

Acknowledgments

Thank you professors Sven Anderson, Keith O'Hara, and Rebecca Thomas for making this project possible through your combined efforts to teach and advise me. Thank you Benjamin Sernau '17 for enduring through three years of Computer Science courses with me and being a source of unending joy in my life. Thank you to Julia Berry '18, Aaron Krapf '18, and Zoe Terhune '18 for being my very best friends and giving me things worth caring about. Finally, thank you to my parents Tammy Segal and Emily Taylor for your constant support and patience throughout my four years at Bard College.

1
Introduction

1.1 Background

The news and social networking service Twitter had over 140 million active users who sent 340 million text-based Tweets to the platform every day by March of 2012 [1]. Since Twitter-founder Jack Dorsey sent the first Tweet in March of 2006 [2] social scientists, advertisers, and computer scientists have applied machine learning techniques to understand the patterns and structures of the conversations held on the platform. One such technique is sentiment analysis, which seeks to ascertain the opinions of bodies of text. Sentiment analysis techniques are often treated as classification problems which seek to place text into categories such as **positive**, **negative**, and **neutral**.

On Twitter, the most common way to publicly communicate with another user is to compose a tweet and place an “@” before the username of that user somewhere in the tweet (e.g. ”How are you doing, @NoahSegalGould?”). Through this method, public discussions on Twitter maintain a kind of accountability: even if one were to miss the notification that they were mentioned in a tweet, one’s own dashboard keeps a running list of their most recent mentions.

If an individual sought to disparage or mock another, they could certainly do so directly. But the targeted user would probably notice, and through the search funtions of the platform, anyone could see who has mentioned either their own or another’s username. Instead, a phenomenon persists in which users of the platform deliberately insult others in the vaguest way possible. Tweets of this kind are colloquially called “subtweets” and typically target a specific person but do not contain the username of that person.

All users do not necessarily possess the same exact definition of “subtweet.” I trust the Oxford English Dictionary’s “[tweet] that refers to a particular user without directly mentioning them, typically as a form of furtive mockery or criticism,” however that definition is perhaps too restrictive. Some individuals believe subtweets abide by this definition, but others expand it to allow inclusion of others’ real names (especially if that individual does not own a Twitter account), and some do not even require that a particular user be the target of the tweet. In this project, I implement a classifier which abides by a particularly loose definition in order to please as many parties as possible.

1.2 Motivation

The inspiration for this project came from interests I garnered taking courses within Bard College’s Computer Science department as well as its Experimental Humanities concentration. The very first course I attended at Bard College was Professor Keith O’Hara’s *Object-Oriented Programming with Robots*. It served as my first introduction to computer programming, and for my final project I created *Fuzzfeed*: a Twitter bot which generated fake *Buzzfeed* article titles. The following academic year, I wrote programs in Professor Collin Jennings’ *Signs and Symbols: Patter Recognition in Literature and Code* and Professor Rune Olsen’s *Cybergraphics* which analyzed and visualized topic models of poetry on Twitter. The first time I implemented sentiment analysis on my own was in Professor Gretta Tritch-Roman’s *Mapping the 19th Century City*, for which I sought to analyze 1860s New York City newspapers for their sentiments toward immigration. Natural language processing struck me as entertaining and fruitful, so I chose to pursue it further.

By my Junior year, my friends and I used Twitter on a daily basis. In my free time, I made Twitter bots that utilized Markov chains to generate text based on corpora of their tweets. Data collection became a passion of mine as I learned to appreciate the utility and acknowledge the deliberate limitations of web-based APIs. I taught myself web-scraping and utilized Python's *BeautifulSoup* library to programmatically acquire text from Bard College's official online course catalog as well as Twitter's web interface. These skills for programmatically interacting with the world wide web became useful resources during the completion of this project.

My peers introduced me to subtweeting, and I started to pay closer attention to tweets that followed the typical patterns of distanced criticism that subtweets were known for. Because some format seemed to exist which was popularly applied to produce the optimal subtweet, I pitched the concept of subtweet classification to my senior project adviser, Professor Sven Anderson, and I started work on this project in the Fall.

I was initially motivated to complete a senior project on this topic because I wanted to create something useful to my peers and also challenge their notions of public and private interactions on social networking applications like Twitter. Individuals I knew personally would take to the platform to complain indirectly about one another through their subtweets. Friends and I shared evenings debating on if a particular mutual friend's complaints were actually subtweets, and I wondered if that guess-work could be done by a program. I also wanted to challenge the hypocrisy of utilizing a service which presents itself as a public forum to speak in distinctly private ways. Toward this end, I decided the project would be in pursuit of the following goals: it would provide a framework for collecting examples of subtweets, train a classification algorithm using those examples, and finally utilize that classifier in real time to make tweets which were intended to be unseen by specific parties easily accessible to all parties. In presenting covertly hurtful content as obviously hurtful in a public fashion, perhaps I could promote a particular awareness that tweets posted by public accounts were indeed publicly accessible, and that Twitter's End User License Agreement (EULA) allowed for this kind of monitoring.

1.3 Literature Review

Instead of focus groups, opinion polls, and conduct surveys, sentiment analysis and opinion mining programs are increasingly applied to social networking websites to analyze the sentiments and opinions of users toward topics and products. For the Twitter social networking platform, sentiment analysis allows administrators to enforce their hateful conduct policies [3] which specifically prohibit violent threats and some types of degrading content. Sentiment analysis has been utilized for prediction of financial markets [4] as well as reactions to terrorist attacks [5], and sentiment analysis with a specific focus on figurative language such as irony has also attracted recent research interest [6]. In an attempt to mock, subtweets often exhibit figurative language such as irony and hyperbole.

"Subtweet" was coined in December of 2009 by Twitter user Chelsea Rae [7] and was entered into Urban Dictionary the following August [8]. In "To tweet or subtweet?: Impacts of social networking post directness and valence on interpersonal impressions" [9], Edwards and Harris sought to analyze student participants' perceptions of known subtweets. In the news, too, subtweets have garnered attention in *The Atlantic* [10], *The Washington Post* [11], and *Slate* [12]. Despite this recent interest, little to no research exists on applying sentiment analysis to subtweets.

1.4 Changes in Data Acquisition

The novel approach I developed for creating a ground truth dataset relied on a particular phenomenon in which Twitter users were already calling-out the subtweets of their peers. The format I noticed followed that a user would post a subtweet which was easily recognized by a peer, and that peer would then reply to that tweet in order to complain that the original user was subtweeting or to ask if the tweet was indeed a subtweet. Initially, the Python script I wrote to utilize the Twitter API's search functionality via the Tweepy library specifically searched for replies to tweets which contained some form of the string "subtweet" and then utilized the API's status object to access the tweet to which it was replying. Both the alleged subtweet and the tweet containing the accusation were saved to a comma-separated values (CSV) file. I ran the script every day for over two months.

Initially, I trained the classifier using a dataset which was half comprised of these alleged subtweets and half comprised of tweets randomly selected from a dataset provided by Alec Go [13]. That was a mistake. I had failed to make the training data representative of actual subtweets and non-subtweets. To rectify this, I revised the alleged subtweets downloading script and created one that had the opposite effect: it downloaded tweets with replies which specifically did **not** contain the string "subtweet." In both the script which downloaded subtweets and the script which downloaded non-subtweets, I knew my assumptions about these interactions would not hold true in every case. They were intended as generalizations which would make acquiring a ground truth dataset for use in performing binary classification significantly easier and less time-consuming than finding and labeling subtweets and non-subtweets by hand.

1.5 The Twitter API

[Explain how it is accessed and its limitations.]

1.6 Regular Expressions, N-Grams, & Tokenization

[Explain the use for regular expressions in identifying hashtags, URLs, and mentions.]

[Explain N-Grams and how they help a classifier.]

[Explain tokenization and specifically NLTK's TweetTokenizer.]

1.7 TF & TF-IDF

TF-IDF, or term frequency-inverse document frequency, is a statistical representation of how important a single word is for each document in a collection of documents.

[Explain vectorization and provide examples.]

1.8 Naive Bayes

Naive Bayes classifiers are probabilistic supervised learning models which make the ”naive” assumption of independence between pairs of features being classified. Sentiment analysis is popularly performed through Naive Bayes.

[Explain some probability, the independence assumption, and the multinomial distribution. Give examples.]

1.9 Statistical Considerations

In tasks pertaining to text classification, like sentiment analysis, precision refers to the number of correctly labeled items which were labeled as belonging to the positive class and in fact did belong to that class (true positives) divided by the total number of elements which were labeled as belonging to the positive class including ones which were labeled positively either correctly or incorrectly. Recall, then, refers to the true positives divided by the total number of elements that actually belong to the positive class.

[Explain F1, Precision, Recall, Accuracy, and Null Accuracy.]

2
Implementation

2.1 Searching for Tweets Using the Twitter API

[Show code and explain how searching works with Tweepy.]

2.2 Cleaning the Data

[Show code and explain how text cleaning genericizes certain features and ignores tweets lacking enough English words.]

2.3 Training the Classifier & K-Folds Cross-Validation

[Explain how pipelines are trained and how K-Folds splits the dataset.]

3
Results

3.1 Distributions & Datasets

[Explain the tables and figures.]

3.2 Confusion Matrices

A confusion matrix is a table which visualizes the performance of an algorithm. In this case, I implemented a Naive Bayes classifier from Scikit Learn on my dataset and included in my results is a confusion matrix of the performance...

[Explain how to read a confusion matrix and show the test and training figures.]

3.3 Most Informative Features

[Explain how to read the most informative features for each class (or just the ”subtweet” class) and show the table.]

3.4 Statistical Analysis

[Show the scores from K-Folds.]

3.5 The Twitter Bot

[Explain how the Twitter bot works and show code as well as examples of interactions.]

4
Conclusion

4.1 Summary of Project Achievements

[Ground truth data, classifier, cleaning, Twitter bot.]

4.2 Future Work & Considerations

[Test other classification algorithms. Utilize more training data.]

Bibliography

[1] Twitter Inc., *Twitter turns six* (2012), available at `https://blog.twitter.com/official/en_us/a/2012/twitter-turns-six.html`.

[2] Jack Dorsey, *inviting coworkers* (2006), available at `https://twitter.com/jack/status/29`.

[3] Twitter, *Hateful conduct policy* (2018), available at `https://help.twitter.com/en/rules-and-policies/hateful-conduct-policy`.

[4] Arman Khadjeh Nassirtoussi, Saeed Aghabozorgi, Teh Ying Wah, and David Chek Ling Ngo, *Text mining for market prediction: A systematic review*, Expert Systems with Applications **41** (2014), no. 16, 7653–7670.

[5] Pete Burnap, Matthew L Williams, Luke Sloan, Omer Rana, William Housley, Adam Edwards, Vincent Knight, Rob Procter, and Alex Voss, *Tweeting the terror: modelling the social media reaction to the woolwich terrorist attack*, Social Network Analysis and Mining **4** (2014), no. 1, 206.

[6] Antonio Reyes and Paolo Rosso, *On the difficulty of automatically detecting irony: beyond a simple case of negation*, Knowledge and Information Systems **40** (2014), no. 3, 595–614.

[7] Chelsea Rae, *I hate when i see people...* (2009), available at `https://twitter.com/Chelsea_x_Rae/status/6261479092`.

[8] Urban Dictionary, *Subtweet* (2010), available at `https://www.urbandictionary.com/define.php?term=subtweet`.

[9] Autumn Edwards and Christina J Harris, *To tweet or subtweet?: Impacts of social networking post directness and valence on interpersonal impressions*, Computers in Human Behavior **63** (2016), 304–310.

[10] Alexis C. Madrigal, *Behind the machine's back: How social media users avoid getting turned into big data* (2014), available at `https://goo.gl/h36jxx`.

[11] Caitlin Dewey, *Study confirms what you always knew: People who subtweet are terrible* (2016), available at `https://goo.gl/SeV3mx`.

[12] Chelsea Hassler, *Subtweeting looks terrible on you. (you know who you are.)* (2016), available at `https://goo.gl/NCz27z`.

[13] Alec Go, Richa Bhayani, and Lei Huang, *Twitter Sentiment Classification using Distant Supervision*, CS224N Project Report, Stanford (2009), 12.

Appendix A

subtweets_downloader.ipynb

Script for downloading a ground truth subtweets dataset

Import libraries for accessing the API and managing JSON data

```
import tweepy
import json
```

Load the API credentials

```
consumer_key, consumer_secret, access_token, access_token_secret = (open("../../credentials.txt")
                                                                    .read().split("\n"))
```

Authenticate the connection to the API using the credentials

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

Connect to the API

```
api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True, compression=True)
```

Define a function for recursively accessing parent tweets

```
def first_tweet(tweet_status_object):
    try:
        return first_tweet(api.get_status(tweet_status_object.in_reply_to_status_id_str, tweet_mode="extended"))
    except tweepy.TweepError:
        return tweet_status_object
```

Define a function for finding tweets with replies that specifically do call them subtweets

```
def get_subtweets(max_tweets=10000000,
                  query=("subtweet AND @ since:2018-03-01 exclude:retweets filter:replies")):
    subtweets_ids_list = []
    subtweets_list = []
    i = 0
    for potential_subtweet_reply in tweepy.Cursor(api.search, lang="en",
                                                  tweet_mode="extended", q=query).items(max_tweets):
        i += 1
        potential_subtweet_original = first_tweet(potential_subtweet_reply)
        if (not potential_subtweet_original.in_reply_to_status_id_str
            and potential_subtweet_original.user.lang == "en"):
            if (potential_subtweet_original.id_str in subtweets_ids_list
                or "subtweet" in potential_subtweet_original.full_text
                or "Subtweet" in potential_subtweet_original.full_text
                or "SUBTWEET" in potential_subtweet_original.full_text):
                continue
            else:
                subtweets_ids_list.append(potential_subtweet_original.id_str)
                subtweets_list.append({"tweet_data": potential_subtweet_original._json,
                                       "reply": potential_subtweet_reply._json})
                with open("../data/other_data/subtweets.json", "w") as outfile:
                    json.dump(subtweets_list, outfile, indent=4)
                print(("Tweet #{0} was a reply to a subtweet: {1}\n"
                       .format(i, potential_subtweet_original.full_text.replace("\n", " "))))
    return subtweets_list
```

Show the results

```
subtweets_list = get_subtweets()
print("Total: {}".format(len(subtweets_list)))
```

Appendix B

non_subtweets_downloader.ipynb

Script for downloading a ground truth non-subtweets dataset

Import libraries for accessing the API and managing JSON data

```
import tweepy
import json
```

Load the API credentials

```
consumer_key, consumer_secret, access_token, access_token_secret = (open("../../credentials.txt")
                                                                    .read().split("\n"))
```

Authenticate the connection to the API using the credentials

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

Connect to the API

```
api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True, compression=True)
```

Define a function for recursively accessing parent tweets

```
def first_tweet(tweet_status_object):
    try:
        return first_tweet(api.get_status(tweet_status_object.in_reply_to_status_id_str, tweet_mode="extended"))
    except tweepy.TweepError:
        return tweet_status_object
```

Define a function for finding tweets with replies that specifically do not call them subtweets

```
def get_non_subtweets(max_tweets=10000000,
                      query=("-subtweet AND @ since:2018-03-01 exclude:retweets filter:replies")):
    non_subtweets_ids_list = []
    non_subtweets_list = []
    i = 0
    for potential_non_subtweet_reply in tweepy.Cursor(api.search, lang="en",
                                                      tweet_mode="extended", q=query).items(max_tweets):
        i += 1
        potential_non_subtweet_original = first_tweet(potential_non_subtweet_reply)
        if (not potential_non_subtweet_original.in_reply_to_status_id_str
            and potential_non_subtweet_original.user.lang == "en"):
            if (potential_non_subtweet_original.id_str in non_subtweets_ids_list
                or "subtweet" in potential_non_subtweet_original.full_text
                or "Subtweet" in potential_non_subtweet_original.full_text
                or "SUBTWEET" in potential_non_subtweet_original.full_text):
                continue
            else:
                non_subtweets_ids_list.append(potential_non_subtweet_original.id_str)
                non_subtweets_list.append({"tweet_data": potential_non_subtweet_original._json,
                                           "reply": potential_non_subtweet_reply._json})
                with open("../data/other_data/non_subtweets.json", "w") as outfile:
                    json.dump(non_subtweets_list, outfile, indent=4)
                print(("Tweet #{0} was a reply to a non-subtweet: {1}\n"
                       .format(i, potential_non_subtweet_original.full_text.replace("\n", " "))))
    return non_subtweets_list
```

Show the results

```
non_subtweets_list = get_non_subtweets()
print(len(non_subtweets_list))
```

Appendix C

classifier_creator.ipynb

Using Scikit-Learn and NLTK to build a Naive Bayes Classifier that identifies subtweets

In all tables, assume:

- "➊" represents a single hashtag
- "➋" represents a single URL
- "➌" represents a single mention of username (e.g. "@noah")

Import libraries

```
%matplotlib inline
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction import text
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import KFold
from sklearn.pipeline import Pipeline
from sklearn.externals import joblib
from os.path import basename, splitext
from random import choice, sample
from nltk.corpus import stopwords
from string import punctuation
from pprint import pprint
from glob import glob

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

import scipy.stats
import itertools
import enchant
import nltk
import json
import re
```

Set up some regex patterns

```
hashtags_pattern = re.compile(r'(\#[a-zA-Z0-9]+)')
```

```
urls_pattern = re.compile(r'(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()<>]|\(([^\s()<>]+|(
```

```
at_mentions_pattern = re.compile(r'(?<=^|(?<=[^a-zA-Z0-9-\.]))@([A-Za-z0-9_]+)')
```

Prepare English dictionary for language detection

```
english_dict = enchant.Dict("en_US")
```

Use NLTK's tokenizer instead of Scikit's

```
tokenizer = nltk.casual.TweetTokenizer(preserve_case=False, reduce_len=True)
```

Prepare for viewing long text in CSVs and ones with really big and small numbers

```
pd.set_option("max_colwidth", 1000)
```

```
pd.options.display.float_format = "{:.4f}".format
```

Load the two data files

Only use tweets with at least 50% English words

Also, make the mentions of usernames, URLs, and hashtags generic

```
def load_data(filename, threshold=0.5):
    data = [(hashtags_pattern.sub("➊",
             urls_pattern.sub("➋",
             at_mentions_pattern.sub("➌",
             t["tweet_data"]["full_text"])))
             .replace("\u2018", "'")
             .replace("\u2019", "'")
             .replace("\u201c", "\"")
             .replace("\u201d", "\"")
             .replace("&quot;", "\"")
             .replace("&amp;", "&")
             .replace("&gt;", ">")
             .replace("&lt;", "<"))
            for t in json.load(open(filename))
            if t["tweet_data"]["user"]["lang"] == "en"
            and t["reply"]["user"]["lang"] == "en"]
    new_data = []
    for tweet in data:
        tokens = tokenizer.tokenize(tweet)
        english_tokens = [english_dict.check(token) for token in tokens]
        percent_english_words = sum(english_tokens)/len(english_tokens)
        if percent_english_words >= threshold:
            new_data.append(tweet)
    return new_data
```

```
subtweets_data = load_data("../data/other_data/subtweets.json")
```

```
non_subtweets_data = load_data("../data/other_data/non_subtweets.json")
```

Show examples

```
print("Subtweets dataset example:")
print(choice(subtweets_data))
```

```
Subtweets dataset example:
I haven't bawled at work since I was in public accounting so THANKS EVERYONE.
```

```
print("Non-subtweets dataset example:")
print(choice(non_subtweets_data))
```

```
Non-subtweets dataset example:
Next up for discussion, this nightmare ➋
```

Find the length of the smaller dataset

```
smallest_length = len(min([subtweets_data, non_subtweets_data], key=len))
```

Cut both down to be the same length

```
subtweets_data = subtweets_data[:smallest_length]
```

```
non_subtweets_data = non_subtweets_data[:smallest_length]
```

```
print("Smallest dataset length: {}".format(len(subtweets_data)))
```

```
Smallest dataset length: 7837
```

Prepare data for training

```
subtweets_data = [(tweet, "subtweet") for tweet in subtweets_data]
```

```
non_subtweets_data = [(tweet, "non-subtweet") for tweet in non_subtweets_data]
```

Combine them

```
training_data = subtweets_data + non_subtweets_data
```

Create custom stop words to include generic usernames, URLs, and hashtags, as well as common English first names

```
names_lower = set([name.lower() for name in open("../data/other_data/first_names.txt").read().split("\n")])
```

```
generic_tokens = {"➊", "➋", "➌"}
```

```
stop_words = text.ENGLISH_STOP_WORDS | names_lower | generic_tokens
```

Build the pipeline

```
sentiment_pipeline = Pipeline([
    ("vectorizer", TfidfVectorizer(tokenizer=tokenizer.tokenize,
                                    ngram_range=(1, 3),
                                    stop_words=stop_words)),
    ("classifier", MultinomialNB())
])
```

K-Folds splits up and separates out 10 training and test sets from the data, from which the classifier is trained and the confusion matrix and classification reports are updated

```
def confusion_matrices(training_data, num_folds=10):
    text_training_data = np.array([row[0] for row in training_data])
    class_training_data = np.array([row[1] for row in training_data])
    kf = KFold(n_splits=num_folds, random_state=42, shuffle=True)

    cnf_matrix_test = np.zeros((2, 2), dtype=int)
    cnf_matrix_train = np.zeros((2, 2), dtype=int)

    test_reports = []
    train_reports = []

    test_nulls = []
    test_accuracies = []

    train_nulls = []
    train_accuracies = []
    for i, (train_index, test_index) in enumerate(kf.split(text_training_data)):

        text_train, text_test = text_training_data[train_index], text_training_data[test_index]
        class_train, class_test = class_training_data[train_index], class_training_data[test_index]

        sentiment_pipeline.fit(text_train, class_train)

        predictions_test = sentiment_pipeline.predict(text_test)
        predictions_train = sentiment_pipeline.predict(text_train)

        cnf_matrix_test += confusion_matrix(class_test, predictions_test)
        cnf_matrix_train += confusion_matrix(class_train, predictions_train)

        print("Test Data Iteration {}:".format(i+1))

        test_report = classification_report(class_test, predictions_test, digits=4)
        test_reports.append(test_report)
        print(test_report)

        test_null = max(pd.value_counts(pd.Series(class_test)))/float(len(class_test))
        test_nulls.append(test_null)
        print("Test Data Null Accuracy: {:.4f}\n".format(test_null))
        test_accuracy = accuracy_score(class_test, predictions_test)
        test_accuracies.append(test_accuracy)
        print("Test Data Accuracy: {:.4f}\n".format(test_accuracy))
        print("="*53)

        print("Train Data Iteration {}:".format(i+1))

        train_report = classification_report(class_train, predictions_train, digits=4)
        train_reports.append(train_report)
        print(train_report)

        train_null = max(pd.value_counts(pd.Series(class_train)))/float(len(class_train))
        train_nulls.append(train_null)
        print("Train Data Null Accuracy: {:.4f}\n".format(train_null))
        train_accuracy = accuracy_score(class_train, predictions_train)
        train_accuracies.append(train_accuracy)
        print("Train Data Accuracy: {:.4f}\n".format(train_accuracy))
        print("="*53)

    def reports_mean(reports):
        reports_lists_of_strings = [report.split("\n") for report in reports]
        reports = [[[float(e) for e in report_string[2][16:].split()],
                    [float(e) for e in report_string[3][16:].split()],
                      [float(e) for e in report_string[5][16:].split()]]
                     for report_string in reports_lists_of_strings]
        mean_list = np.mean(np.array(reports), axis=0).tolist()
        print("              precision    recall  f1-score   support")
        print()
        print("non-subtweet     {0:.4f}     {1:.4f}     {2:.4f}      {3:d}".format(mean_list[0][0],
                                                                          mean_list[0][1],
                                                                          mean_list[0][2],
                                                                          int(mean_list[0][3])))
        print("    subtweet     {0:.4f}     {1:.4f}     {2:.4f}      {3:d}".format(mean_list[1][0],
                                                                          mean_list[1][1],
                                                                          mean_list[1][2],
                                                                          int(mean_list[1][3])))
        print()
        print(" avg / total     {0:.4f}     {1:.4f}     {2:.4f}      {3:d}".format(mean_list[2][0],
                                                                          mean_list[2][1],
                                                                          mean_list[2][2],
                                                                          int(mean_list[2][3])))
        print()
        print("="*53)

    print("Test Data Averages Across All Folds:")
    reports_mean(test_reports)

    print("Train Data Averages Across All Folds:")
    reports_mean(train_reports)

    print("Average Test Data Null Accuracy: {:.4f}\n".format(sum(test_nulls)/float(len(test_nulls))))
    print("Average Test Data Accuracy: {:.4f}\n".format(sum(test_accuracies)/float(len(test_accuracies))))

    print("Average Train Data Null Accuracy: {:.4f}\n".format(sum(train_nulls)/float(len(train_nulls))))
    print("Average Train Data Accuracy: {:.4f}\n".format(sum(train_accuracies)/float(len(train_accuracies))))

    return {"Test": cnf_matrix_test, "Train": cnf_matrix_train}
```

```
%%time
cnf_matrices = confusion_matrices(training_data)
cnf_matrix_test = cnf_matrices["Test"]
cnf_matrix_train = cnf_matrices["Train"]
```

```
OUTPUT HAS BEEN TRUNCATED FOR PRINTING

=====================================================
Test Data Averages Across All Folds:
              precision    recall  f1-score   support

non-subtweet     0.7125    0.6506    0.6798      783
    subtweet     0.6785    0.7376    0.7065      783

 avg / total     0.6960    0.6939    0.6933      1567

=====================================================
Train Data Averages Across All Folds:
              precision    recall  f1-score   support

non-subtweet     0.9889    0.9819    0.9854      7053
    subtweet     0.9820    0.9890    0.9855      7053

 avg / total     0.9855    0.9854    0.9854      14106

=====================================================
Average Test Data Null Accuracy: 0.5134

Average Test Data Accuracy: 0.6939

Average Train Data Null Accuracy: 0.5015

Average Train Data Accuracy: 0.9854
```

```
CPU times: user 1min 2s, sys: 1.13 s, total: 1min 4s
Wall time: 1min 5s
```

See the most informative features

How does "MultinomialNB.coef_" work?

```python
def most_informative_features(pipeline, n=10000):
    vectorizer = pipeline.named_steps["vectorizer"]
    classifier = pipeline.named_steps["classifier"]

    class_labels = classifier.classes_

    feature_names = vectorizer.get_feature_names()

    top_n_class_1 = sorted(zip(classifier.coef_[0], feature_names))[:n]
    top_n_class_2 = sorted(zip(classifier.coef_[0], feature_names))[-n:]

    return {class_labels[0]: pd.DataFrame({"Log Probability": [tup[0] for tup in top_n_class_1],
                                           "Feature": [tup[1] for tup in top_n_class_1]}),
            class_labels[1]: pd.DataFrame({"Log Probability": [tup[0] for tup in reversed(top_n_class_2)],
                                           "Feature": [tup[1] for tup in reversed(top_n_class_2)]})}
```

```python
%%time
most_informative_features_all = most_informative_features(sentiment_pipeline)
```

```
CPU times: user 1.42 s, sys: 28 ms, total: 1.44 s
Wall time: 1.51 s
```

```python
most_informative_features_non_subtweet = most_informative_features_all["non-subtweet"]
```

```python
most_informative_features_subtweet = most_informative_features_all["subtweet"]
```

```python
final_features = most_informative_features_non_subtweet.join(most_informative_features_subtweet,
                                                              lsuffix=" (Non-subtweet)",
                                                              rsuffix=" (Subtweet)")
final_features.to_csv("../data/other_data/most_informative_features.csv")
final_features.head(25)
```

	Feature (Non-subtweet)	Log Probability (Non-subtweet)	Feature (Subtweet)	Log Probability (Subtweet)
0	! ! &	-12.6618	.	-7.5300
1	! ! (	-12.6618	,	-7.9193
2	! !)	-12.6618	"	-8.0928
3	! ! .	-12.6618	people	-8.3903
4	! ! 100	-12.6618	?	-8.4594
5	! ! 15	-12.6618	don't	-8.5588
6	! ! 3	-12.6618	like	-8.5889
7	! ! 5	-12.6618	just	-8.6754
8	! ! 8am	-12.6618	i'm	-8.6969

	Feature (Non-subtweet)	Log Probability (Non-subtweet)	Feature (Subtweet)	Log Probability (Subtweet)
9	! ! :)	-12.6618	!	-8.9031
10	! ! ;)	-12.6618	it's	-8.9727
11	! ! absolutely	-12.6618	...	-9.0431
12	! ! amazing	-12.6618	you're	-9.0488
13	! ! ask	-12.6618	:	-9.0704
14	! ! awesome	-12.6618	know	-9.0928
15	! ! big	-12.6618	twitter	-9.1443
16	! ! bite	-12.6618	friends	-9.1650
17	! ! close	-12.6618	time	-9.2879
18	! ! collection	-12.6618	want	-9.2923
19	! ! come	-12.6618	u	-9.3004
20	! ! don't	-12.6618	really	-9.3518
21	! ! enter	-12.6618	shit	-9.3699
22	! ! epic	-12.6618	good	-9.4017
23	! ! extremely	-12.6618	think	-9.4155
24	! ! family	-12.6618	make	-9.4225

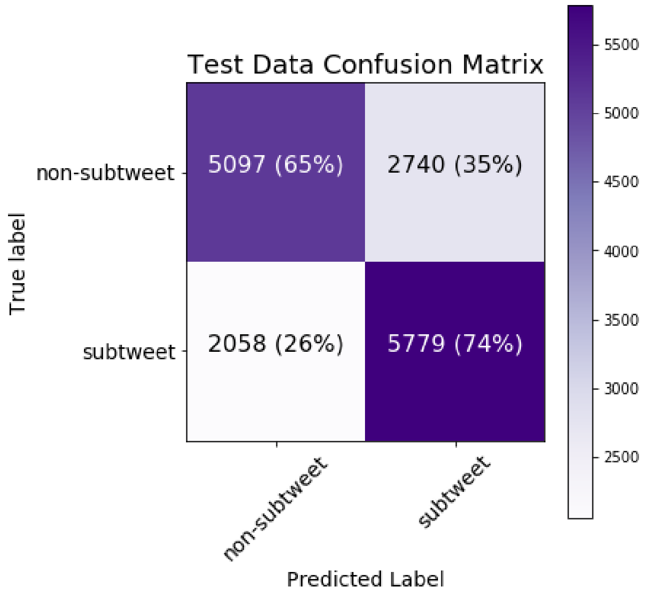
Define function for visualizing confusion matrices

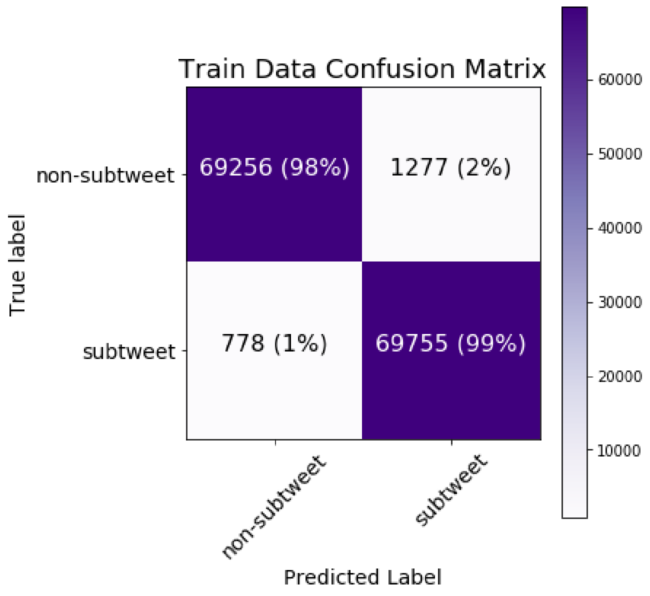
```python
def plot_confusion_matrix(cm, classes=["non-subtweet", "subtweet"],
                          title="Confusion Matrix", cmap=plt.cm.Purples):
    
    cm_normalized = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]
    
    plt.imshow(cm, interpolation="nearest", cmap=cmap)
    plt.colorbar()
    
    plt.title(title, size=18)
    
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45, fontsize=14)
    plt.yticks(tick_marks, classes, fontsize=14)
    
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, "{} ({:.0%})".format(cm[i, j], cm_normalized[i, j]),
                horizontalalignment="center", size=16,
                color="white" if cm[i, j] > thresh else "black")
    
    plt.tight_layout()
    
    plt.ylabel("True label", fontsize=14)
    plt.xlabel("Predicted Label", fontsize=14)
```

Show the matrices

```python
np.set_printoptions(precision=2)

plt.figure(figsize=(6, 6))
```

```
plot_confusion_matrix(cnf_matrix_test, title="Test Data Confusion Matrix")

plt.figure(figsize=(6, 6))
plot_confusion_matrix(cnf_matrix_train, title="Train Data Confusion Matrix")

plt.show()
```





Update matplotlib style

```
plt.style.use("fivethirtyeight")
```

Save the classifier for another time

```
joblib.dump(sentiment_pipeline, "../data/other_data/subtweets_classifier.pkl");
```

Print tests for the classifier

```
def process_tweets_for_testing(filenames):
    dataframes = {}
    for filename in filenames:
        username = splitext(basename(filename))[0][:-7]
        dataframes[username] = {}

        user_df = pd.read_csv(filename).dropna()
        user_df["Text"] = user_df["Text"].str.replace(hashtags_pattern, "➊")
        user_df["Text"] = user_df["Text"].str.replace(urls_pattern, "➋")
        user_df["Text"] = user_df["Text"].str.replace(at_mentions_pattern, "➌")
        user_df["Text"] = user_df["Text"].str.replace("\u2018", "'")
        user_df["Text"] = user_df["Text"].str.replace("\u2019", "'")
        user_df["Text"] = user_df["Text"].str.replace("\u201c", "\"")
        user_df["Text"] = user_df["Text"].str.replace("\u201d", "\"")
        user_df["Text"] = user_df["Text"].str.replace("&quot;", "\"")
        user_df["Text"] = user_df["Text"].str.replace("&amp;", "&")
        user_df["Text"] = user_df["Text"].str.replace("&gt;", ">")
        user_df["Text"] = user_df["Text"].str.replace("&lt;", "<")

        predictions = sentiment_pipeline.predict_proba(user_df["Text"])[:, 1].tolist()
        user_df["SubtweetProbability"] = predictions

        dataframes[username]["all"] = user_df

        scores = user_df[["SubtweetProbability"]].rename(columns={"SubtweetProbability": username})

        dataframes[username]["scores"] = scores
        dataframes[username]["stats"] = scores.describe()

    return dataframes
```

Load the CSV files

```
filenames = glob("../data/data_for_testing/friends_data/*.csv")
```

```
%%time
dataframes = process_tweets_for_testing(filenames)
```

```
CPU times: user 9.54 s, sys: 145 ms, total: 9.68 s
Wall time: 10.4 s
```

Show a random table

```
chosen_username = choice(list(dataframes.keys()))
dataframes[chosen_username]["all"].sort_values(by="SubtweetProbability", ascending=False).head(5)
```

	Text	Date	Favorites	Retweets	Tweet ID	SubtweetProbability
462	ppl saying zionist shit on the internet really fucks w my high	2017-07-17 02:27:07	11	0	886834632125288448	0.8244
15	i hate seeing shitty straight people yelling at their kids in public like why did you breed	2018-03-21 12:49:00	24	3	976500935437496320	0.8140

	Text	Date	Favorites	Retweets	Tweet ID	SubtweetProbability
392	some1 replied to my tweet about cis ppl making xcuses 4 not dating trans ppl w "bc they have a fucking cock"	2017-08-01 19:08:46	10	0	892522524361322496	0.8044
563	cw my shit mental health: u know shit is f'd up when ur lookin @ a meme abt dying of old age and yr like "this meme is actually optimistic"	2017-06-20 22:12:12	2	1	877348396029358080	0.7965
477	I FUCKING LOVE QUEER PEOPLE	2017-07-09 21:20:04	19	1	884220643226644480	0.7938

Prepare statistics on tweets

```
tests_df = pd.concat([df_dict["scores"] for df_dict in dataframes.values()], ignore_index=True)
```

```
test_df_stats = tests_df.describe()
```

```
test_df_stats.columns = ["User {}".format(i + 1) for i, column in enumerate(test_df_stats.columns)]
```

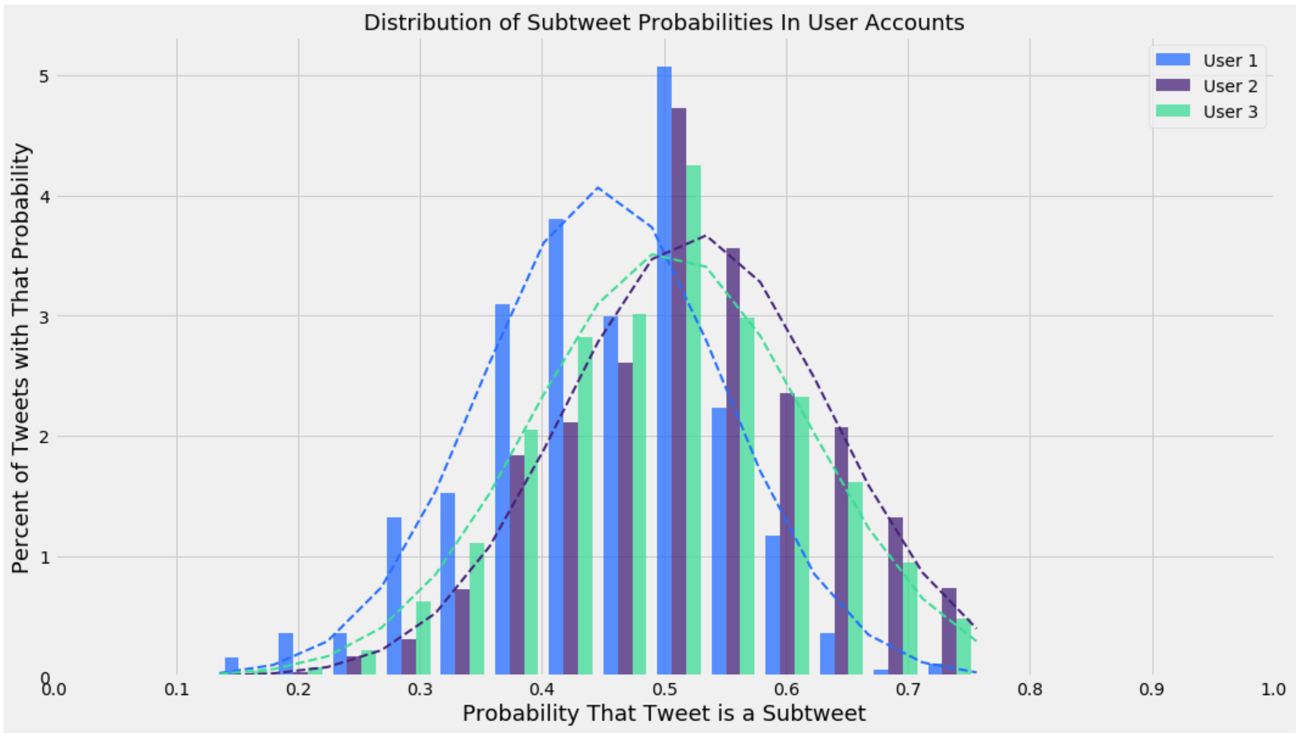
```
test_df_stats
```

	User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8
count	621.0000	2640.0000	2066.0000	3488.0000	4356.0000	1939.0000	1169.0000	638.000
mean	0.4996	0.5086	0.5438	0.5270	0.5187	0.4976	0.4388	0.5408
std	0.1059	0.1150	0.1136	0.1086	0.1023	0.1106	0.0981	0.1152
min	0.1981	0.0953	0.1266	0.1626	0.1522	0.0566	0.1497	0.1983
25%	0.4291	0.4304	0.4669	0.4538	0.4492	0.4260	0.3733	0.4700
50%	0.4971	0.5037	0.5417	0.5217	0.5180	0.4981	0.4379	0.5327
75%	0.5670	0.5847	0.6213	0.5982	0.5843	0.5669	0.5016	0.6190
max	0.8457	0.8579	0.8497	0.8749	0.8674	0.8766	0.8157	0.8498

User 1 User 2 User 3 User 4 User 5 User 6 User 7 User 8

Plot a histogram with three random users

```
random_choices = sample(list(dataframes.values()), 3)
scores = [df_dict["scores"][df_dict["scores"].columns[0]].tolist()
          for df_dict in random_choices]

fig = plt.figure(figsize=(16, 9))
ax = fig.add_subplot(111)

n, bins, patches = ax.hist(scores,
                           bins="scott",
                           color=["#256EFF", "#46237A", "#3DDC97"],
                           density=True,
                           label=["User 1", "User 2", "User 3"],
                           alpha=0.75)

stats = [df_dict["stats"][df_dict["stats"].columns[0]].tolist()
         for df_dict in random_choices]

line_1 = scipy.stats.norm.pdf(bins, stats[0][1], stats[0][2])
ax.plot(bins, line_1, "--", color="#256EFF", linewidth=2)

line_2 = scipy.stats.norm.pdf(bins, stats[1][1], stats[1][2])
ax.plot(bins, line_2, "--", color="#46237A", linewidth=2)

line_3 = scipy.stats.norm.pdf(bins, stats[2][1], stats[2][2])
ax.plot(bins, line_3, "--", color="#3DDC97", linewidth=2)

ax.set_xticks([float(x/10) for x in range(11)], minor=False)
ax.set_title("Distribution of Subtweet Probabilities In User Accounts", fontsize=18)
ax.set_xlabel("Probability That Tweet is a Subtweet", fontsize=18)
ax.set_ylabel("Percent of Tweets with That Probability", fontsize=18)

ax.legend()

plt.show()
```



Plot a histogram with all of them

First, get some statistics

```
new_tests_df = pd.concat([df_dict["scores"].rename(columns={df_dict["scores"].columns[0]:"SubtweetProbability"})
                          for df_dict in dataframes.values()], ignore_index=True)

new_tests_df_stats = new_tests_df.describe()
```

Then view them

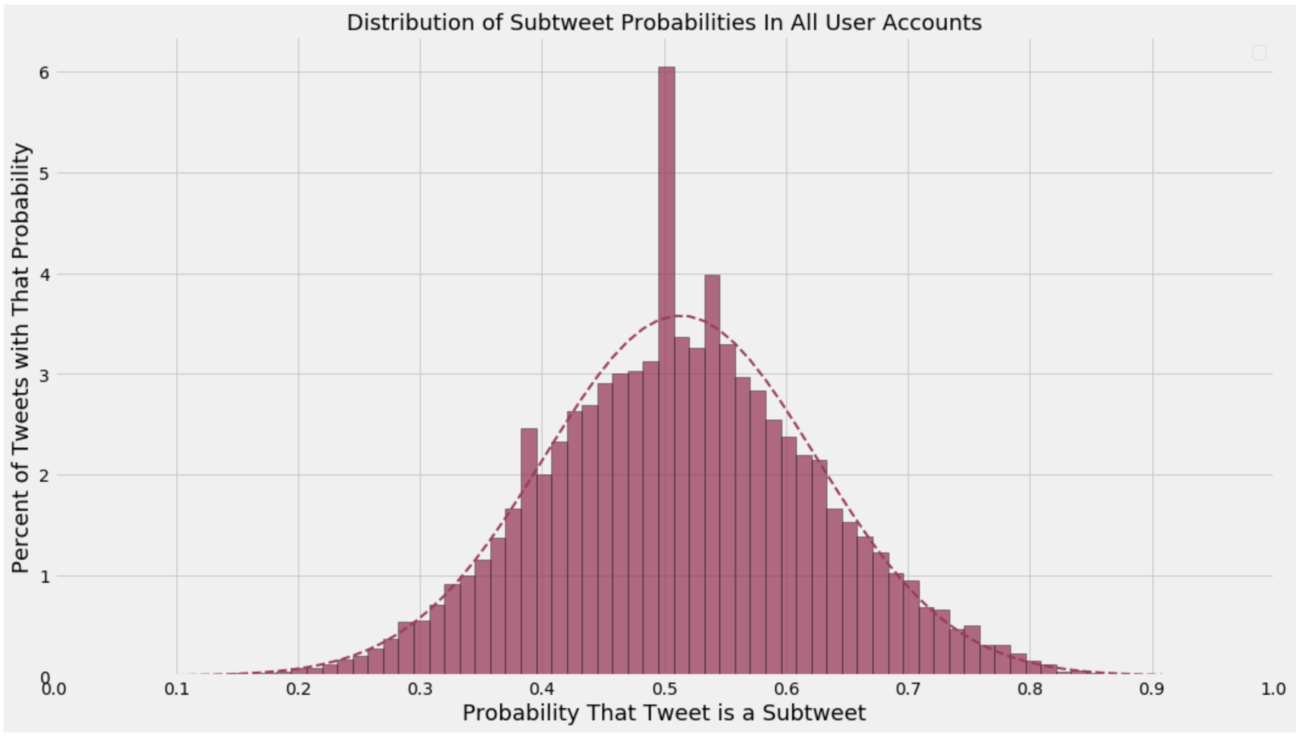
```
new_tests_df_stats
```

	SubtweetProbability
count	28632.0000
mean	0.5133
std	0.1115
min	0.0566
25%	0.4385
50%	0.5093
75%	0.5860
max	0.9091

Now plot

```
fig = plt.figure(figsize=(16, 9))
ax = fig.add_subplot(111)

n, bins, patches = ax.hist(new_tests_df["SubtweetProbability"].tolist(),
                           bins="scott",
                           color="#983B59",
                           edgecolor="black",
                           density=True,
                           alpha=0.75)

line = scipy.stats.norm.pdf(bins, new_tests_df_stats["SubtweetProbability"][1],
                              new_tests_df_stats["SubtweetProbability"][2])

ax.plot(bins, line, "--", color="#983B59", linewidth=2)


ax.set_xticks([float(x/10) for x in range(11)], minor=False)
ax.set_title("Distribution of Subtweet Probabilities In All User Accounts", fontsize=18)
ax.set_xlabel("Probability That Tweet is a Subtweet", fontsize=18)
ax.set_ylabel("Percent of Tweets with That Probability", fontsize=18)

ax.legend()

plt.show()
```

```
No handles with labels found to put in legend.
```



Statisitics on training data

Remove mentions of usernames for these statistics

```
training_data = [(tweet[0]
                  .replace("➊", "")
                  .replace("➋", "")
                  .replace("➌", "")) for tweet in training_data]
```

Lengths

```
length_data = [len(tweet) for tweet in training_data]
```

```
length_data_for_stats = pd.DataFrame({"Length": length_data, "Tweet": training_data})
```

```
# length_data_for_stats = length_data_for_stats[length_data_for_stats["Length"] <= 280]
```

```
# length_data_for_stats = length_data_for_stats[length_data_for_stats["Length"] >= 5]
```

```
length_data = length_data_for_stats.Length.tolist()
```

Top 5 longest tweets

```
length_data_for_stats.sort_values(by="Length", ascending=False).head()
```

	Length	Tweet
8887	281	This Tweet does not endorse the use of Nazi Symbols in any form! I think the image which has been published on social media and MSM is a day or two old. It conjures up strong emotions for many people, My question is simple what meaning do you think is being conveyed by the image?

	Length	Tweet
2198	281	I need to learn how to do this. I ask "how can I help" a lot because I genuinely want to make things better for friends , but this *can* put a burden back upon those who are suffering. Sometimes it may be best to just have exuberant and fearless compassion the same way a pet does
1531	281	hi! I'm not normally v personal like this and I probably won't be at least for a v long time but I thought I'd share this \nwhile I was scrolling on Twitter today I had like a sudden impulse to just dump all my thoughts about what id been reading and seeing and so far it actually-
10533	281	Some people are undecided about testing on animals. Understandable. There's so much propaganda and secrecy about it. Here's a quick test though, & you're answer should tell you. What would you do if some man came to your house & squirted disinfectant in your beautiful dog's eyes?
10521	281	Enthralled by Raja Shiv Chhatrapati, a well mounted magnum opus on life of the Maratha warrior at Red Fort. Vividly brought out his philosophies, struggles, inspiration from mother Jijayee & penchant for gender equality through well conceived music, dance & dialogues. A must see!

Top 5 shortest tweets

```
length_data_for_stats.sort_values(by="Length", ascending=True).head()
```

	Length	Tweet
7699	1	A
3473	2	no
5896	2	uh
6676	2	i-
2038	2	Ha

Tweet length statistics

```
length_data_for_stats.describe()
```

	Length
count	15674.0000
mean	106.8089
std	73.8680
min	1.0000
25%	48.0000
50%	87.0000
75%	150.0000
max	281.0000
	Length

Punctuation

```
punctuation_data = [len(set(punctuation).intersection(set(tweet))) for tweet in training_data]
```

```
punctuation_data_for_stats = pd.DataFrame({"Punctuation": punctuation_data, "Tweet": training_data})
```

Top 5 most punctuated tweets

```
punctuation_data_for_stats.sort_values(by="Punctuation", ascending=False).head()
```

	Punctuation	Tweet
8957	11	Going to go ahead and crown myself the absolute emperor of finding things on menus that sound interesting, deciding I would like to try them, then being told "I'm sorry sir, that's actually not available..."\n\n[then why the @#$% is it ON YOUR MENUUUUUUUU--]
6725	9	4-yo: DADDEEEEEE!? LET'S PLAY!\nMe: Ok, baby. \n4yo: you play w/ her. put a dress on her DADDEEEEEE. \nMe: Ok. *puts doll in dollhouse*\n4yo: SHE DOESN'T GO THERE!!
11718	9	Self-employed people: have you ever turned to social media to call out a client who is many weeks/months delinquent on a payment? \n(Obviously, you're probably burning a bridge with that move, but if they don't pay...)
13365	9	Billboard Hot 100: (-3) Tell Me You Love Me, [19 weeks]. *peak: *
11845	9	Tucker Carlson Tonight & TFW you're asking about America\nbut you're scolded it's really about Israel ...\n \nTucker: "What is the American national security interest ... in Syria?"\n\nSen. Wicker(R): "Well, if you care about Israel ..." \n\nThat was the exact question & answer\nShocking

Tweets punctuation statistics

```
punctuation_data_for_stats.describe()
```

	Punctuation
count	15674.0000
mean	1.9168
std	1.5787
min	0.0000
25%	1.0000
50%	2.0000
75%	3.0000
max	11.0000

Stop words

```
stop_words_data = [len(set(stopwords.words("english")).intersection(set(tweet.lower())))
                   for tweet in training_data]
```

```
stop_words_data_for_stats = pd.DataFrame({"Stop words": stop_words_data, "Tweet": training_data})
```

Top 5 tweets with most stop words

```
stop_words_data_for_stats.sort_values(by="Stop words", ascending=False).head()
```

	Stop words	Tweet
0	8	I don't yet have adequate words to do so, but someday I wanna write about the beautiful dance which happens in Google docs between a writer & a good editor working simultaneously towards a deadline. When it's working, it's a beautiful dance—though no one really sees it.
9063	8	Honestly yea i fucked up but all of you are trash asf and your opinions mean nothing to me because mother fucker i can fix shit but yall are to close minded to see.
9035	8	The role of DAG Rod Rosenstein will be an Oscar winner in the future film about the Trump presidency. I'd like the story of the first few months to be told through the eyes of the bewildered Sean Spicer.
9038	8	Done watching 'Hacksaw Ridge'. If there's one thing I learned from that movie, it is simply, Have Faith in God.
9039	8	I feel people who can't celebrate or at the very least respect Cardi B's success have never watched the grind from the ground up. They can't understand that her work ethic has gotten her where she is now. You don't have to stand for what's she's about but she's worked for it

Top 5 tweets with fewest stop words

```
stop_words_data_for_stats.sort_values(by="Stop words", ascending=True).head()
```

	Stop words	Tweet
3632	0	...
8290	0	24
11925	0	FUCK
10940	0	78 ... !
1796	0	fuck u

Tweets stop words statistics

```
stop_words_data_for_stats.describe()
```

	Stop words
count	15674.0000
mean	7.1515
std	1.3116
min	0.0000
25%	7.0000
50%	8.0000
75%	8.0000
max	8.0000

Unique words

```
unique_words_data = [len(set(tokenizer.tokenize(tweet))) for tweet in training_data]
```

```
unique_words_data_for_stats = pd.DataFrame({"Unique words": unique_words_data, "Tweet": training_data})
```

```
# unique_words_data_for_stats = unique_words_data_for_stats[unique_words_data_for_stats["Unique words"] >= 2]
```

```
unique_words_data = unique_words_data_for_stats["Unique words"].tolist()
```

Top 5 tweets with most unique words

```
unique_words_data_for_stats.sort_values(by="Unique words", ascending=False).head()
```

	Tweet	Unique words
13936	GIVE AWAY!\n\nThe rules are really easy, all you have to do is :\n1. Must be following me (i check) \n2. RT and fav this tweet\n3. tag your mutuals/anyone\n4. only 1 winner! \n5. i ship worldwide;) \n\nit ends in 8th May 2018 or when this tweet hit 2k RT and like!\n\nGood luck! ❤️	59
4881	got into a tepid back nd forth w/ a uknowwhoAJ+columnist bc i said they steal their "hot takes" from blk twitter & alike. wallahi my bdeshi ass did not sign up 4 this app to be called asinine by a 30yrold pakistani whos whole politics is Post Colonial Memes for Oriental Minded T-	57
7013	Crazy how wrong u can be about someone. A girl I graduated w/ was always doing drugs& got pregnant at 16. I assumed she'd end up being a loser but it turn out she now has 4 beautiful kids& is making over $4,500/month just off of child support payments from the 3 different dads	57
4992	Got into an argument w/ someone I went to HS w/ & I would js like to repeat again tht I cannot wait to stunt on all the ppl who were bitches to me in HS @ our reunion. Catch me rollin up w/ my sexy ass gf, a nice car, a bomb body & the career of my dreams as a big fuck u to them	55
11542	Thought I'd bring this back... and no, I'm not talking about myself here. I wish just once I'd be so bored with my life that I'd find the time to bash people/celebs I don't like.. I mean if I despise someone THAT much, why still watch his/her every move? 🤦‍♀️	55

Top 5 tweets with fewest unique words

```
unique_words_data_for_stats.sort_values(by="Unique words", ascending=True).head()
```

	Tweet	Unique words
6106	Annoying	1
2525	Bitch	1
12087	Chandler	1
14559	Yes yes yes yes yes yes	1
14442	Hello\n	1

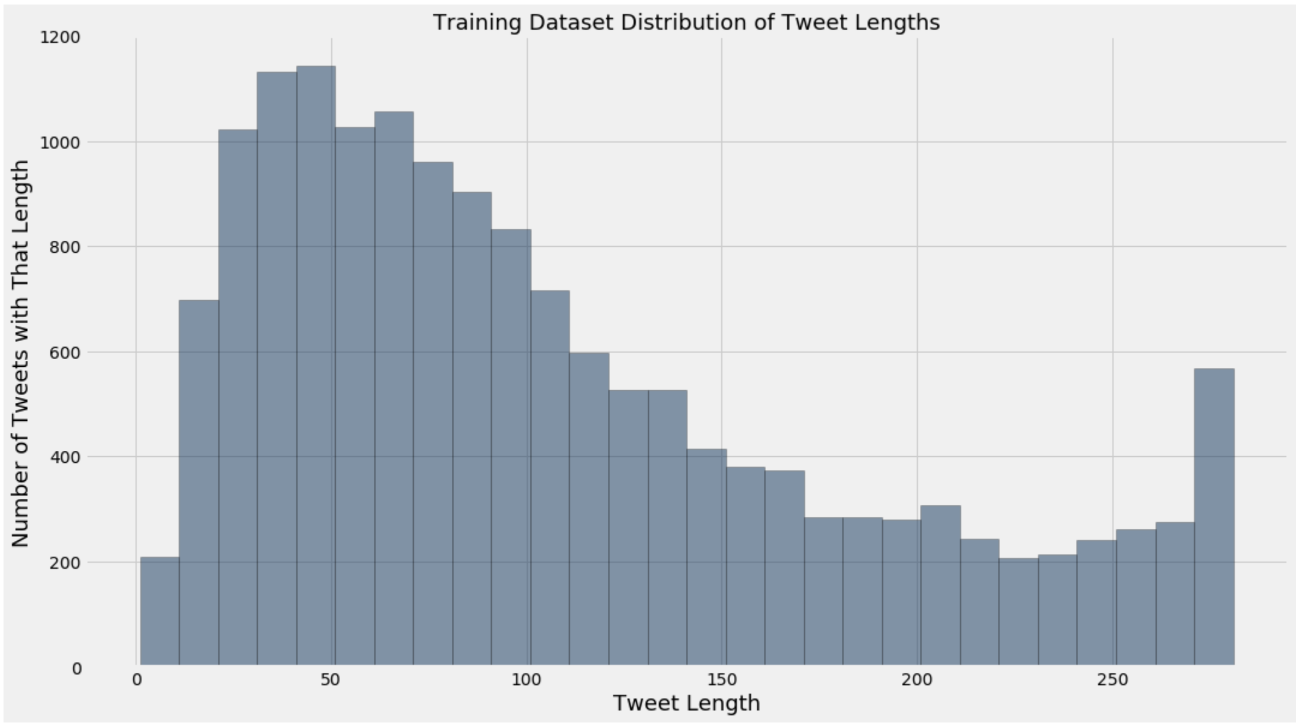
Tweets unique words statistics

```
unique_words_data_for_stats.describe()
```

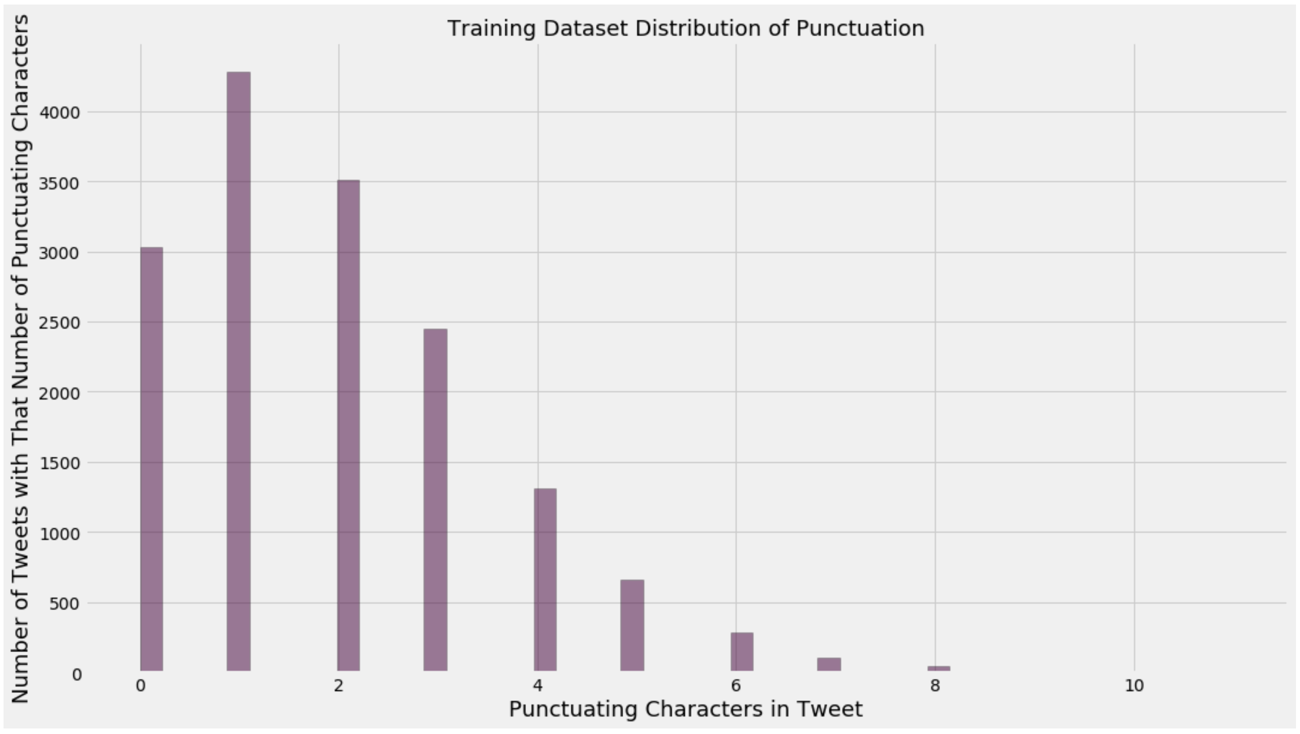
	Unique words
count	15674.0000
mean	19.2412
std	11.9298
min	1.0000
25%	10.0000
50%	17.0000
75%	27.0000
max	59.0000

Plot them

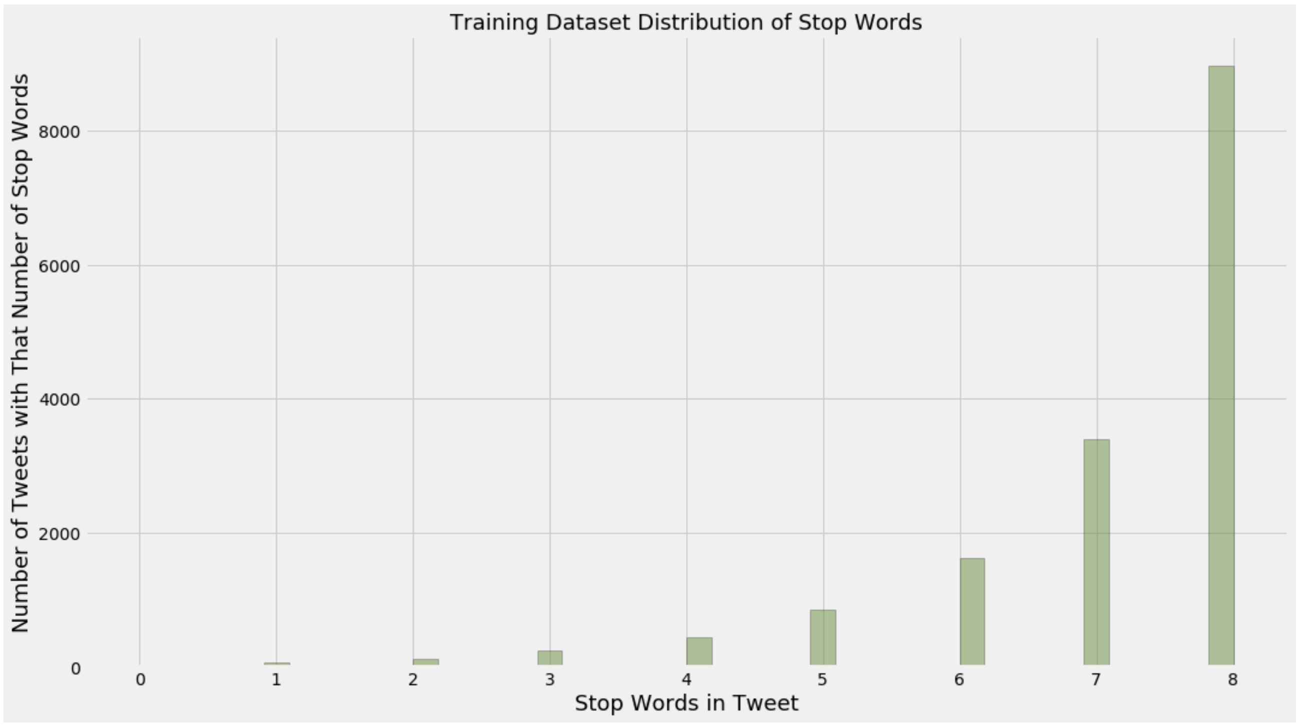
```
length_mean = length_data_for_stats.describe().Length[1]
length_std = length_data_for_stats.describe().Length[2]

fig = plt.figure(figsize=(16, 9))
ax = fig.add_subplot(111)

n, bins, patches = ax.hist(length_data,
                           bins="scott",
                           edgecolor="black",
                           # density=True,
                           color="#12355b",
                           alpha=0.5)

# length_line = scipy.stats.norm.pdf(bins, length_mean, length_std)
# ax.plot(bins, length_line, "--", linewidth=3, color="#415d7b")

ax.set_title("Training Dataset Distribution of Tweet Lengths", fontsize=18)
ax.set_xlabel("Tweet Length", fontsize=18);
ax.set_ylabel("Number of Tweets with That Length", fontsize=18);

plt.show()
```



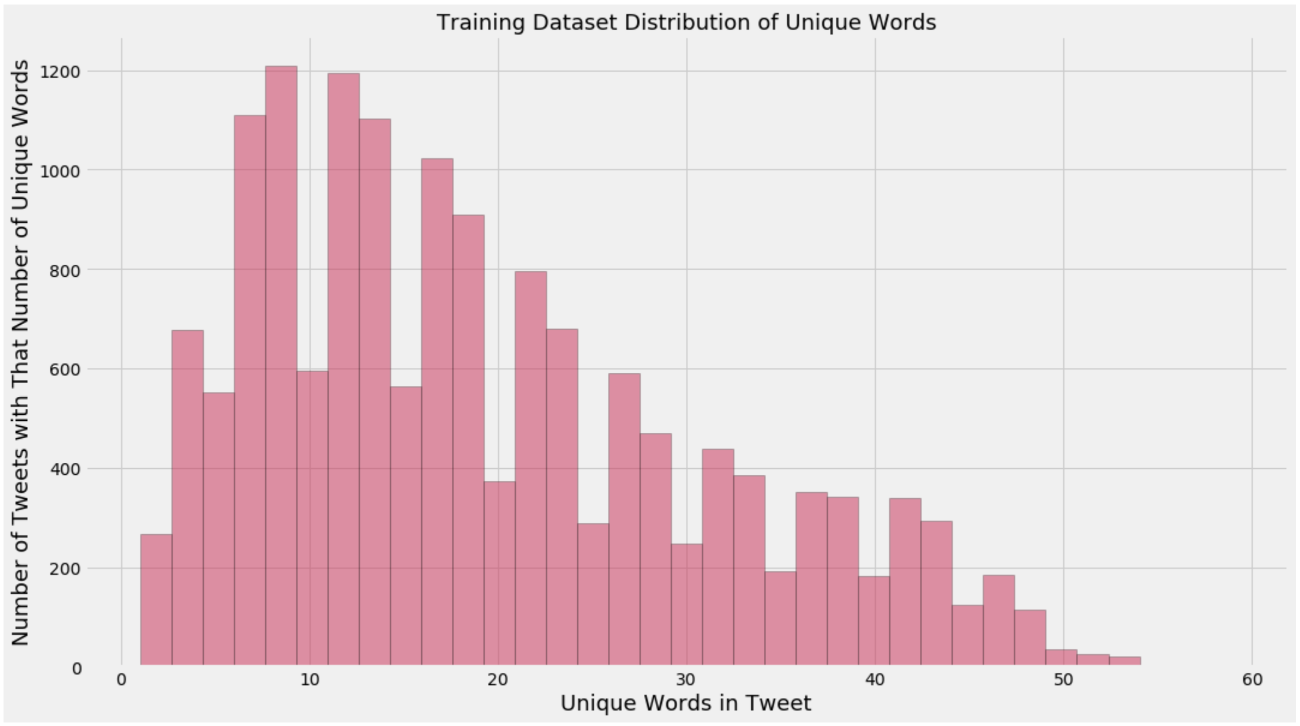
```python
punctuation_mean = punctuation_data_for_stats.describe().Punctuation[1]
punctuation_std = punctuation_data_for_stats.describe().Punctuation[2]

fig = plt.figure(figsize=(16, 9))
ax = fig.add_subplot(111)

n, bins, patches = ax.hist(punctuation_data,
                           bins="scott",
                           edgecolor="black",
                           # density=True,
                           color="#420039",
                           alpha=0.5)

# punctution_line = scipy.stats.norm.pdf(bins, punctuation_mean, punctuation_std)
# ax.plot(bins, punctution_line, "--", linewidth=3, color="#673260")

ax.set_title("Training Dataset Distribution of Punctuation", fontsize=18)
ax.set_xlabel("Punctuating Characters in Tweet", fontsize=18)
ax.set_ylabel("Number of Tweets with That Number of Punctuating Characters", fontsize=18)

plt.show()
```



```
stop_words_mean = stop_words_data_for_stats.describe()["Stop words"][1]
stop_words_std = stop_words_data_for_stats.describe()["Stop words"][2]

fig = plt.figure(figsize=(16, 9))
ax = fig.add_subplot(111)

n, bins, patches = ax.hist(stop_words_data,
                           bins="scott",
                           edgecolor="black",
                           # density=True,
                           color="#698f3f",
                           alpha=0.5)

# stop_words_line = scipy.stats.norm.pdf(bins, stop_words_mean, stop_words_std)
# ax.plot(bins, stop_words_line, "--", linewidth=3, color="#87a565")

ax.set_title("Training Dataset Distribution of Stop Words", fontsize=18)
ax.set_xlabel("Stop Words in Tweet", fontsize=18)
ax.set_ylabel("Number of Tweets with That Number of Stop Words", fontsize=18)

plt.show()
```



```
unique_words_mean = unique_words_data_for_stats.describe()["Unique words"][1]
unique_words_std = unique_words_data_for_stats.describe()["Unique words"][2]

fig = plt.figure(figsize=(16, 9))
ax = fig.add_subplot(111)

n, bins, patches = ax.hist(unique_words_data,
                           bins="scott",
                           edgecolor="black",
                           # density=True,
                           color="#ca2e55",
                           alpha=0.5)

# unique_words_line = scipy.stats.norm.pdf(bins, unique_words_mean, unique_words_std)
# ax.plot(bins, unique_words_line, "--", linewidth=3, color="#d45776")

ax.set_title("Training Dataset Distribution of Unique Words", fontsize=18)
ax.set_xlabel("Unique Words in Tweet", fontsize=18)
ax.set_ylabel("Number of Tweets with That Number of Unique Words", fontsize=18)

plt.show()
```


Training Dataset Distribution of Unique Words
Number of Tweets with That Number of Unique Words
Unique Words in Tweet
0
200
400
600
800
1000
1200
0
10
20
30
40
50
60

Appendix D

live_subtweets_classifier.ipynb

Script for running a Twitter bot that interacts with subtweets

Import some libraries

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction import text
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold
from sklearn.externals import joblib
from nltk.corpus import stopwords
from string import punctuation
from pprint import pprint
from random import choice
from time import sleep

import pandas as pd
import numpy as np

import itertools
import enchant
import tweepy
import nltk
import json
import re
```

Prepare the probability threshold for interacting with a potential subtweet and the duration for which the bot should run

```
THRESHOLD = 0.75 # 75% positives and higher, only
DURATION = 60*10 # 10 minutes
```

Set up regular expressions for genericizing extra features

```
hashtags_pattern = re.compile(r'(\#[a-zA-Z0-9]+)')
```

```
urls_pattern = re.compile(r'(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()<>]|\(([^\s()<>]+|(
```

```
at_mentions_pattern = re.compile(r'(?<=^|(?<=[^a-zA-Z0-9-\.]))@([A-Za-z0-9_]+)')
```

Load the classifier pipeline which was previously saved

```
sentiment_pipeline = joblib.load("../data/other_data/subtweets_classifier.pkl")
```

Load the Twitter API credentials

```
consumer_key, consumer_secret, access_token, access_token_secret = (open("../../credentials.txt")
                                                                    .read().split("\n"))
```

Connect to the API

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth, retry_delay=1, timeout=120, # 2 minutes
                 compression=True,
                 wait_on_rate_limit=True, wait_on_rate_limit_notify=True)
```

Create lists to fill with tweets while the bot is streaming

```
subtweets_live_list = []
non_subtweets_live_list = []
```

Use pyenchant to check if words are English

```
english_dict = enchant.Dict("en_US")
```

Use NLTK for tokenizing

```
tokenizer = nltk.casual.TweetTokenizer(preserve_case=False, reduce_len=True)
```

Create a custom StreamListener class for use with Tweepy

```
class StreamListener(tweepy.StreamListener):
    def on_status(self, status):
        choices = ["retweet", "like", "retweet and like", "reply"]

        id_str = status.id_str
        screen_name = status.user.screen_name
        created_at = status.created_at
        retweeted = status.retweeted
        in_reply_to = status.in_reply_to_status_id_str

        # The text and media of the tweet vary based on if it's extended or not
        if "extended_tweet" in status._json:
            if "full_text" in status._json["extended_tweet"]:
                text = status._json["extended_tweet"]["full_text"]
                has_media = "media" in status._json["extended_tweet"]["entities"]
            else:
                pass # Something else?
        elif "text" in status._json:
            text = status._json["text"]
            has_media = "media" in status._json["entities"]

        # Genericize extra features and clean up the text
        text = (hashtags_pattern.sub("➊",
                urls_pattern.sub("➋",
                at_mentions_pattern.sub("➌",
                text)))
                .replace("\u2018", "'")
                .replace("\u2019", "'")
                .replace("\u201c", "\"")
                .replace("\u201d", "\"")
                .replace("&quot;", "\"")
                .replace("&amp;", "&")
                .replace("&gt;", ">")
            .replace("&lt;", "<"))

        includes_subtweet = any(["subtweet" in text,
                                 "Subtweet" in text,
                                 "SUBTWEET" in text])

        tokens = tokenizer.tokenize(text)

        english_tokens = [english_dict.check(token) for token in tokens]
        percent_english_words = sum(english_tokens)/float(len(english_tokens))

        # Make sure the tweet is mostly english
        is_mostly_english = False
        if percent_english_words >= 0.5:
            is_mostly_english = True

        # Calculate the probability using the pipeline
        positive_probability = sentiment_pipeline.predict_proba([text]).tolist()[0][1]

        row = {"tweet": text,
               "screen_name": screen_name,
               "time": created_at,
               "subtweet_probability": positive_probability}

        print_list = pd.DataFrame([row]).values.tolist()[0]

        # Only treat it as a subtweet if all conditions are met
        if all([positive_probability >= THRESHOLD,
                "RT " != text[:3], is_mostly_english, not includes_subtweet,
                not retweeted, not in_reply_to, not has_media]):

            decision = choice(choices)
            if decision == "retweet":
                api.update_status(("Is this a subtweet? {:.3%} \n" +
                                   "https://twitter.com/{}/status/{}").format(positive_probability,
                                                                              screen_name,
                                                                              id_str))
                print("Retweet!")

            elif decision == "like":
                api.create_favorite(id_str)
                print("Like!")

            elif decision == "retweet and like":
                api.update_status(("Is this a subtweet? {:.3%} \n" +
                                   "https://twitter.com/{}/status/{}").format(positive_probability,
                                                                              screen_name,
                                                                              id_str))
                api.create_favorite(id_str)
                print("Retweet and like!")

            elif decision == "reply":
                api.update_status("@{} Is this a subtweet? {:.3%}".format(screen_name,
                                                                          positive_probability),
                                  id_str)
                print("Reply!")

            subtweets_live_list.append(row)
            subtweets_df = pd.DataFrame(subtweets_live_list).sort_values(by="subtweet_probability",
                                                                          ascending=False)

            subtweets_df.to_csv("../data/data_from_testing/live_downloaded_data/subtweets_live_data.csv")

            print(("Subtweet from @{0} (Probability of {1:.3%}):\n" +
                   "Time: {2}\n" +
                   "Tweet: {3}\n" +
                   "Total tweets acquired: {4}\n").format(print_list[0],
                                                          print_list[1],
                                                          print_list[2],
                                                          print_list[3],
                                                          (len(subtweets_live_list)
                                                          + len(non_subtweets_live_list))))

            return row
        else:
            non_subtweets_live_list.append(row)
            non_subtweets_df = pd.DataFrame(non_subtweets_live_list).sort_values(by="subtweet_probability",
                                                                                  ascending=False)
            non_subtweets_df.to_csv("../data/data_from_testing/live_downloaded_data/non_subtweets_live_data.csv")

            return row
```

Create a function for downloading IDs if users I follow who also follow me

```python
def get_mutuals():
    my_followers = [str(user_id) for ids_list in
                    tweepy.Cursor(api.followers_ids,
                                  screen_name="NoahSegalGould").pages()
                    for user_id in ids_list]
    my_followeds = [str(user_id) for ids_list in
                   tweepy.Cursor(api.friends_ids,
                                 screen_name="NoahSegalGould").pages()
                   for user_id in ids_list]

    my_mutuals = list(set(my_followers) & set(my_followeds))

    bots = ["890031065057853440", "895685688582180864",
            "894658603977777152", "970553455709446144",
            "786489395519983617", "975981192817373184"]

    # Remove known twitter bots
    my_mutuals = [m for m in my_mutuals if m not in bots]

    with open("../data/other_data/NoahSegalGould_Mutuals_ids.json", "w") as outfile:
        json.dump(my_mutuals, outfile, sort_keys=True, indent=4)

    return my_mutuals
```

Create a function for downloading IDs of users who follow my mutuals who are also followed by my mutuals

```python
def get_mutuals_and_mutuals_mutuals_ids(mutuals_threshold=250):
    my_mutuals = get_mutuals()
    my_mutuals_mutuals = my_mutuals[:]

    for i, mutual in enumerate(my_mutuals):
        start_time = time()
        user = api.get_user(user_id=mutual)
        name = user.screen_name
        is_protected = user.protected
        if not is_protected:
            mutuals_followers = []
            followers_cursor = tweepy.Cursor(api.followers_ids, user_id=mutual).items()
            while True:
                try:
                    mutuals_follower = followers_cursor.next()
                    mutuals_followers.append(str(mutuals_follower))
                except tweepy.TweepError:
                    sleep(30) # 30 seconds
                    continue
                except StopIteration:
                    break
            mutuals_followeds = []
            followeds_cursor = tweepy.Cursor(api.friends_ids, user_id=mutual).items()
            while True:
                try:
                    mutuals_followed = followeds_cursor.next()
                    mutuals_followeds.append(str(mutuals_followed))
                except tweepy.TweepError:
```

```
                sleep(30) # 30 seconds
                continue
            except StopIteration:
                break
        mutuals_mutuals = list(set(mutuals_followers) & set(mutuals_followeds))
        print("{} mutuals for mutual {}: {}".format(len(mutuals_mutuals), i+1, name))
        if len(mutuals_mutuals) <= mutuals_threshold: # Ignore my mutuals if they have a lot of mutuals
            my_mutuals_mutuals.extend(mutuals_mutuals)
        else:
            print("\tSkipping: {}".format(name))
    else:
        continue
    end_time = time()
    with open("../data/other_data/NoahSegalGould_Mutuals_and_Mutuals_Mutuals_ids.json", "w") as outfile:
        json.dump(my_mutuals_mutuals, outfile, sort_keys=True, indent=4)
    print(("{0:.2f} seconds for getting the mutuals' IDs of mutual {1}: {2}\n")
          .format((end_time - start_time), i+1, name))
my_mutuals_mutuals = [str(mu) for mu in sorted([int(m) for m in list(set(my_mutuals_mutuals))])]
with open("../data/other_data/NoahSegalGould_Mutuals_and_Mutuals_Mutuals_ids.json", "w") as outfile:
    json.dump(my_mutuals_mutuals, outfile, indent=4)
return my_mutuals_mutuals
```

```
# %%time
# my_mutuals_mutuals = get_mutuals_and_mutuals_mutuals_ids()
```

Load the IDs JSON

```
my_mutuals_mutuals = json.load(open("../data/other_data/NoahSegalGould_Mutuals_and_Mutuals_Mutuals_ids.json"))
```

```
print("Total number of my mutuals and my mutuals' mutuals: {}".format(len(my_mutuals_mutuals)))
```

```
Total number of my mutuals and my mutuals' mutuals: 4218
```

Begin streaming

```
stream_listener = StreamListener()
stream = tweepy.Stream(auth=api.auth, listener=stream_listener, tweet_mode="extended")
```

```
%%time
# stream.filter(locations=[-73.920176, 42.009637, -73.899739, 42.033421],
# stall_warnings=True, languages=["en"], async=True)
stream.filter(follow=my_mutuals_mutuals, stall_warnings=True, languages=["en"], async=True)
print("Streaming has started.")
sleep(DURATION)
stream.disconnect()
```

```
Streaming has started.
CPU times: user 20.8 s, sys: 3.2 s, total: 24 s
Wall time: 15min
```