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 purpose of this project is to identify subtweets. The Oxford English Dictionary defines "subtweet" as a "[Twitter post] that refers to a particular user without directly mentioning them, typically as a form of furtive mockery or criticism." This paper details a process for gathering a labeled ground truth dataset, training a classifier, and creating a Twitter bot which interacts with subtweets in real time. The Naive Bayes classifier trained in this project classifies tweets as subtweets and non-subtweets with an average F_1 score of 72%.

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

"Subtweet" was coined in December of 2009 by Twitter user Chelsea Rae (Rae, 2009) and was entered into Urban Dictionary the following August (Urban Dictionary, 2010). In "To tweet or subtweet?: Impacts of social networking post directness and valence on interpersonal impressions" (Edwards and Harris, 2016), Edwards and Harris sought to analyze student participants' perceptions of known subtweeters. In the news, too, subtweets have garnered attention in *The Atlantic* (Madrigal, 2014), *The Washington Post* (Dewey, 2016), and *Slate* (Hassler, 2016). In news media, subtweets garner attention for their prevalence among government officials as well. Following President Donald Trump's inauguration, The Washington Post compiled its "A running list of all the possible subtweets of President Trump from government Twitter accounts," (Ohlheiser, 2017) cementing subtweets as particularly newsworthy.

There were over 140 million active Twitter users who sent 340 million text-based tweets to the platform every day by March of 2012 (Twitter, 2012). Since Twitter-founder Jack Dorsey sent the first Tweet in March of 2006 (Dorsey, 2006) social scientists, political scientists, and computer scientists have applied machine learning techniques to understand the patterns and structures of the conversations held on the platform. Through sentiment analysis, we are able to use machine learning to identify patterns within natural language which indicate particular feelings both broadly (e.g. positive, neutral, or negative) and toward topics (e.g. politics, terrorism, brands, etc.).

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 functions 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 a vague manner by making complaints while omitting the targets of those complaints.

Although the OED’s definition states that a subtweet “...refers to a particular user without directly mentioning them, typically as a form of furtive mockery or criticism,” it 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. Because subtweeting is colloquial in nature, we will expand the definition of subtweet to permit these less restrictive features.

Sentiment analysis on social networking services such as Twitter has garnered attention within seemingly distinct fields of interest. In “Text mining for market prediction: A systematic review,” Nassirtoussi et al. surveyed varied methods for text-mining social media for sentiment analysis of financial markets and approached that problem with both behavioral and economic considerations in mind (Nassirtoussi et al., 2014). Following a terrorist event in Woolwich, London in 2013, Burnap et al. analyzed the immediate Twitter response following the attack to inform statistics on how long it takes for responses from official sources to disseminate during crises (Burnap et al., 2014). Prior research of these kinds utilizes sentiment analysis techniques on tweets, but no known research exists which specifically performs any sentiment analysis on subtweets.

Long before Twitter, psychologist Gordon Allport wrote about “antilocution” in *The Nature of Prejudice* (Allport, 1954). For Allport, antilocution was the first of several degrees of apathy which measure prejudice in a society. It represented the kind of remarks which target a person, group, or community in a public or private setting but do not address the targeted individual directly. Different from both hate speech and subtweeting, antilocution necessitates that an in-group ostracize an unaware out-group through its biases.

The most germane research available focuses on sentiment analysis of figurative language. Determining sentiment based on features of text which are distinctly separate from their literal interpretations presents difficulties for human readers as well as computer programs. In *SemEval*, the International Workshop on Semantic Evaluation, analysis of figurative language on Twitter has been a core task for their competition since 2015 (Ghosh et al., 2015) and returns this year with a specific focus on ironic tweets (Van Hee et al., 2018). In this year’s description for “Task 3: Irony detection in English tweets,” Van Hee et al. touch upon online harassment as a potential point of significance for sentiment analysis of ironic tweets.

We pursue sentiment analysis of subtweets in order to challenge the hypocrisy of utilizing a service which presents itself as a public forum to speak in distinctly private ways. Toward this end, these are our goals: this project will 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 it will promote a particular awareness that tweets posted by public accounts are indeed publicly accessible, and that Twitter’s Terms of Service (Twitter, 2016) allows for this kind of monitoring.

Twitter bots are automated programs which access and archive Twitter. During the 2016 presidential election, The Atlantic’s Oxford University researchers revealed that more than 33% of pro-Trump tweets and nearly 20% of pro-Clinton tweets came from automated Twitter accounts between the first and second presidential debates (Guilbeault and Woolley, 2016). Sometimes indistinguishable to untrained users, Twitter bots often misuse the API and violate the Terms of Service to harass or serve an unseen agenda. In contrast, some Twitter bots are purely entertaining like "Pentametron," a Twitter bot that searches Twitter every few hours for tweets which happen to fit the rhythm of iambic pentameter (Bhatnagar, 2018).

Using a machine-learning approach to perform sentiment analysis, syntactic and linguistic features are typically utilized in probabilistic (e.g. Naive Bayes and Maximum Entropy) and linear (e.g. Support Vector Machines and Neural Networks) classification algorithms. The probabilistic approach is sometimes called *generative* because such models generate the probabilities of sampling particular terms (Medhat et al., 2014). Linear classification utilizes the vectorized feature space of words, sentences, or documents to find a separating hyperplane between multiple classes. In this project, we approach the problem of identifying subtweets using the probabilistic Naive Bayes classification algorithm and create a Twitter bot to interact with identified subtweets.

2
Methods

2.1 Gathering Tweets for the Ground Truth Dataset

2.1.1 The Twitter API

Twitter provides a free Application Programming Interface (API) to registered users and has done so since September of 2006 (Stone, 2006). The API allows developers to programmatically access and influence tweets individually or through real time search filters, and also read and write direct messages (Twitter, 2018). The creation of a Twitter application which utilizes the API requires creation and email verification of an account, and developers are also required to agree to the terms of service (Twitter, 2016). Creation of an application provides developers with authentication tokens which can then be used to access the API.

To make creation of Twitter applications easier, Tweepy (Roesslein, 2009) is an open source library for the Python programming language which provides methods and classes used to interact with the API and its status objects (Twitter, 2018). A Twitter status object is a dictionary of key and value pairs which contains text, media, and user information associated with particular tweets. There are rate limits for both reading and writing to the API which must be kept in mind when programming for it.

2.1.2 Searching for Tweets

For acquisition of a ground truth dataset, we consider **true subtweets** to be tweets to which another Twitter user replied who specifically called it out as a subtweet. We consider **true non-subtweets** to be tweets to which another user replied who specifically did **not** call it out as a subtweet.

```python
def get_subtweets(max_tweets=2500,
                  query=("subtweet AND @ since:2018-04-01"
                         "exclude:retweets filter:replies")):
    subtweets_ids_list = []
    subtweets_list = []
    for potential_subtweet_reply in tweepy.Cursor(api.search, lang="en",
                                                  tweet_mode="extended",
                                                  q=query).items(max_tweets):
        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)
    return subtweets_list
```

After the API credentials are loaded, this Python code is used to download tweets with replies which **do** and **do not** call them out as subtweets. Tweepy provides the `Cursor` object which is used to iterate through different sections of the Twitter API. In this example, we use it with Twitter's search API to find tweets matching our query. This particular version finds **true subtweets** with their accusatory replies, but it requires only modification of its query argument to download **true non-subtweets**. Within the `for` loop, we confirm that the following qualities hold true:

- The alleged subtweet or non-subtweet is not in reply to any other tweet.
- The user who posted it used English as their primary language.
- We do not download duplicate tweets.
- It does not contain the string "subtweet" (with variations in capitalization).

For acquisition of both subtweets and non-subtweets, this function can be modified to accept different queries. Respectively, these are the queries which were utilized in acquiring so-called subtweet-accusations (i.e. these replies typically claim that the tweet to which they reply was a subtweet) and non-subtweet-accusations (i.e. these replies are essentially normal replies to normal tweets):

`"subtweet AND @ since:2018-04-01 exclude:retweets filter:replies"`

`"-subtweet AND @ since:2018-04-01 exclude:retweets filter:replies"`

The only difference between the two is that the former searches for tweets which contain the string "subtweet" and the latter searches for tweets which exclude it. Both access the API as far back as it will allow (typically one week for free API users) and exclude retweets (i.e. unoriginal tweets which are reposted) while specifically searching for tweets which were in reply to other tweets. Unfortunately, tweet status objects representing replies to tweets do not contain the object data of the tweet to which they reply.

```python
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
```

To obtain this object, this code essentially goes up a chain of recursively finding the original tweet to which a reply was made until the API can no longer find another tweet object with an `in_reply_to_status_id_str` attribute, indicating that the tweet is an original **true subtweet** or **true non-subtweet**. The two versions of this program for acquiring **true subtweets** and **true non-subtweets** ran for three weeks between March and April of 2018, acquiring over 20,000 tweets which compose our ground truth dataset.

	True Subtweet Data	True Non-Subtweet Data
Tweet	Talk to him again about "dropping me" and you'll get your teeth knocked out	That's been one of my biggest issues here; the onus is on ordinary people who, in their spare time, must campaign for the basic services of a city. This is not how progressive cities should be built
Reply	Thomas don't subtweet me during work hours	i guess i am not as creative as i thought

This table features a **true subtweet** and its associated reply and a **true non-subtweet** and its associated reply.

2.1.3 Changes in Data Acquisition

This approach for creating a ground truth dataset relies on a particular phenomenon in which Twitter users call-out the subtweets of their peers. The following pattern was observed: a user posts a subtweet which is easily recognized by a peer, and that peer then replies 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 program for acquiring the ground truth dataset used the Twitter API's search functionality to specifically search for replies to tweets which contained some form of the string "subtweet." It utilized the API's status object to access the tweet to which it was replying. For 73 days, each day's alleged subtweets and their associated accusatory replies were saved.

Previously, the classifier was trained using a dataset which was half composed of these alleged subtweets and half composed of tweets randomly selected from a pre-labeled sentiment analyzed tweets dataset (Go et al., 2009). This procedure failed to make the training data representative of **true subtweets** and **true non-subtweets**. The alleged subtweets downloading program was revised and it has been set to download tweets with replies which specifically do **not** contain the string "subtweet." In both the program which downloads subtweets and the program which downloads non-subtweets, the assumptions about these interactions will not hold true in every case. They are intended as generalizations which 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. Indeed, the dataset utilized by Go et al. uses a similar method for acquiring labeled data. In their *Sentiment140* dataset, the labels were acquired according to emoticons present within the tweets instead of through hand-labeling by actual humans.

2.2 Language Analysis & Naive Bayes

Before training the classifier, the ground truth dataset is modified such that the important features within it are easily accessible. Changes are made to preserve the characteristics of the text which are relevant to the goal of the analysis and to leave out the ones which are irrelevant. Because we will be using Naive Bayes, we must keep in mind which features in each tweet (e.g. URLs and user names) ought to influence the probabilities that an entire feature-set (i.e. that whole tweet) suits a particular class.

2.2.1 Resources for Language Analysis

Regular Expressions

For text classification through machine learning, it is popular to modify the ground truth dataset to make features which are not important to the classification problem as **generic** as possible. For classification of subtweets, the classifier will treat URLs, mentions of usernames, and English first names generically. In other words, it will keep track of the existence of those features but specifically will not encounter the text contained within them. In identification of subtweets, there exists no syntactic or linguistic significance in the format of a URL or the name a user chooses to associate with themselves or another. However, the existence of those features within the tweet remains important. For this kind of substring searching, pattern matching through regular expressions was used to replace every occurence of URLs, usernames, and first names with special tokens which were not already in the dataset. The top 100 most common English names for both men and women over the last century were acquired from the United States Department of Social Security.

Tokenization

Instead of training the classifier on entire strings, **tokenization** is necessary in order to extract individual features from the text. The Natural Language Toolkit (Bird and Loper, 2004) provides a tweet tokenizer to achieve this. For some string, the tokenizer splits apart words, usernames, URLs, hashtags, and punctuating characters as individual tokens. NLTK's tweet tokenizer also appropriately distinguishes between punctuating characters and emoticons composed of punctuating characters.

N-Grams

An **n-gram** is a contiguous sequence of n tokens in a piece of text. For example, given a string such as “This is a test,” the bigrams ($n = 2$) for this string are “This is,” “is a,” and “a test.” Instead of training the classifier using unigrams ($n = 1$) exclusively, we train it using unigrams, bigrams, and trigrams ($n = 3$). Thus, when the probability that some specific token within a tweet belongs to a specific class is calculated, its neighbors are also considered in combination with it. N-grams enable the classifier to treat particular groupings of tokens with some size n as importantly as it treats the individual tokens, thus identifying particular word groupings most associated with the classes.

Stop Words

A list of stop words typically contains the most common words in a language. For English text, the list is often composed of words such as “the,” “it,” and “of.” Tokens matching stop words are ignored during classifier training because they are too common to help the classifier distinguish subtweets from non-subtweets.

2.2.2 Cleaning and Preparing the Data

Although **true subtweets** and **true non-subtweets** are identified using characteristics of the Twitter API’s tweet status objects, we utilize Naive Bayes exclusively for text classification on the unicode text contained within each alleged subtweet and non-subtweet status object. Thus, we ignore the replies to these tweets.

```python
def load_data(filename, threshold=0.1):
    data = [(urls_pattern.sub("GENERIC_URL",
             at_mentions_pattern.sub("GENERIC_MENTION",
             names_pattern.sub("GENERIC_NAME",
             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"]["lang"] == "en"
            and t["reply"]["lang"] == "en"
            and 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
```

The `load_data` function genericizes URLs, mentions of user names, and mentions of English first names. Using regular expressions for pattern matching, these substrings are replaced with special identifiers. The tweets are also cleaned to make HTML characters and unicode characters more consistent. The list comprehension intentionally excludes **non-English** tweets and those which were **not** posted by accounts which list their primary language as English. NLTK's tweet tokenizer is utilized at the end to check for tweets which contain at least 10% English tokens. This language detection is performed on each token in the tweet using the pyEnchant library (Kelly, 2016) which primarily serves as a spell-checker. The resulting dataset of either **true subtweets** or **true non-subtweets** is returned as a list.

After text cleaning, we remove tweets which are present in both the dataset of **true subtweets** and **true non-subtweets**. Duplicates may have appeared because one user thought a tweet was a subtweet but another did not.

```
        subtweets_data = load_data("../data/other_data/subtweets.json")
        non_subtweets_data = load_data("../data/other_data/non_subtweets.json")
        subtweets_data = [tweet for tweet in subtweets_data
                          if tweet not in non_subtweets_data]
        non_subtweets_data = [tweet for tweet in non_subtweets_data
                              if tweet not in subtweets_data]
```

With the duplicates gone, we limit the size of the larger dataset to be the same as the smaller of the two. Thus, both the **true subtweets** and **true non-subtweets** compose the entire final ground truth dataset equally.

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

        subtweets_data = sample(subtweets_data, smallest_length)
        non_subtweets_data = sample(non_subtweets_data, smallest_length)

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

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

        training_data = subtweets_data + non_subtweets_data
```

In this code, Python's `random` library provides the `sample` function which randomly returns a list of n items from a list. After both datasets are made the same length, the lists of strings are made into lists of tuples. For each tuple, the **true subtweet** or **true non-subtweet** is associated with its label (i.e. subtweet and non-subtweet). Finally, these two lists of tuples are put together for use in training the classifier.

2.2.3 TF & TF-IDF

Term frequency (TF) is a simple method for vectorizing text in which all terms (i.e. tokens, features, words, etc.) in the corpus are featured in a vector for each document, and the frequency of each term is reflected in the number representing the corresponding term. The bag of words model essentially vectorizes features using this method. Thus, $tf(t, d) = f_{t,d}$.

Unfortunately, TF falls short when the corpus of documents contains terms which appear frequently but do not necessarily help inform the classifier on terms that are best associated with a particular class. TF-IDF, or term frequency-inverse document frequency, is the product of the TF for a specific term and the inverse document frequency (IDF) for that same term. The TF is equal to the ratio between the number of occurrences of a term in a document, and the total number of words in that document. IDF, then, is the logarithm of the ratio between the number of documents in the corpus, and the number of documents which contain that term. The product of TF and IDF assigns weights which appropriately value terms which are frequent within a document but rare in the entire corpus of documents. Thus, $idf = \log \frac{N}{n_t}$ where N is the total number of documents and n_t is the number of documents which contain the term. Because the weighted feature vectors calculated using TF-IDF follow a multinomial distribution, our classification algorithm is specifically a Multinomial Naive Bayes classifier.

2.2.4 Naive Bayes

Naive Bayes stands out as particularly simple and common for use in text classification. A **bag of words** model typically ignores word positions in favor of keeping track of raw token frequencies, which are weighted to produce TF-IDF feature vectors. Then, Bayes theorem is utilized to predict the probability that a given feature set (e.g. words, sentences, etc.) belongs to a particular label (i.e. a category or class). Bayes theorem is a means of predicting the **posterior probability** that an event occurs given the observation of another event. It relies on the **conditional probability** that the other event occurs given the observation of the event and the **prior probabilities** that both events occur in general.

Thus,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where $P(A|B)$ is the **posterior probability**, $P(B|A)$ is the **conditional probability**, and $P(A)$ and $P(B)$ are the **prior probabilities**.

The **prior probability** $P(A)$ is otherwise known as the **class probability** and is equal to the general probability of encountering a particular class. In our case, we have chosen to keep our classes balanced (i.e. there are equal numbers of documents in both), so the **class probability** will always be 50%. The other **prior probability** $P(B)$ or **evidence**, then, refers to the probability of encountering the features within a document **independent** of the class label. The **conditional probability** $P(B|A)$ refers to the likelihood of encountering the features within a document given that those features belong to a particular class. The **evidence** is often ignored in the final classification step on the basis that it acts merely as a scaling factor when trying to maximize which class produces the greater **posterior probability**. The **posterior probability** is the probability that a particular document belongs to a class given the observed features within that document. The Naive Bayes algorithm maximizes this probability in order to predict which class best fits a document. In all cases where we must calculate the probability for an entire feature-set, we simply take the product of all the extracted feature probabilities in the document. Consider this classification example:

if $P(\omega = \text{subtweet} \mid \mathbf{x}) \geq P(\omega = \text{non-subtweet} \mid \mathbf{x})$ classify as subtweet,

else classify as non-subtweet.

Dropping the evidence term on the basis that it is constant for both classes, we expand that:

$$P(\omega = \text{subtweet} \mid \mathbf{x}) = P(\mathbf{x} \mid \omega = \text{subtweet}) \cdot P(\text{subtweet})$$

$$P(\omega = \text{non-subtweet} \mid \mathbf{x}) = P(\mathbf{x} \mid \omega = \text{non-subtweet}) \cdot P(\text{non-subtweet})$$

Assuming the **posterior probability** for the former is greater than or equal to the latter, the classifier predicts that the document fits that class. The naive assumption maintains that all features are treated as conditionally independent (i.e. that the presence or omission of a particular feature does not change the likelihood of encountering other features), and although this is frequently violated, Naive Bayes often performs well anyway (Zhang, 2004).

For cases in which the classifier encounters a feature absent from the features which were used to train it, a so-called **zero probability** appears. Because the probability of encountering the feature is 0, **additive smoothing** is often utilized to appropriately weight new features using an extra term α, so the probability that an entire feature-set fits into a specific class is not 0. In this project, we use **Laplace smoothing** ($\alpha = 1$). This technique smooths categorical data by including the pseudo-count α into each probability estimate. Thus, the probability of a feature given a particular class becomes:

$$P(x_i \mid \omega_j) = \frac{N_{x_i,\omega_j} + \alpha}{N_{\omega_j} + \alpha\, d} \quad (i = (1, ..., d))$$

where N_{x_i,ω_j} is the number of times feature x_i appears in samples from class ω_j, N_{ω_j} is the total count of all features in class ω_j, α is the parameter for additive smoothing, and d is the dimensionality of the feature vector $\mathbf{x} = [x_1, ..., x_d]$. Compared to datasets which contain millions of tweets such as *Sentiment140* (Go et al., 2009), our classifier has access to significantly fewer documents. We utilize Laplace smoothing because when the classifier is tested on new tweets it will likely encounter never before seen features given the limited size of the ground truth dataset.

2.2.5 Statistical Considerations

In the binary classification of subtweets and non-subtweets, we consider true positives (TP) to be **true subtweets** which were correctly labeled as **predicted subtweets**, false positives (FP) to be **true non-subtweets** which were incorrectly labeled as **predicted subtweets**, true negatives (TN) to be **true non-subtweets** which were correctly labeled as **predicted non-subtweets**, and false negatives (FN) to be **true subtweets** which were incorrectly labeled as **predicted non-subtweets**. As such, there are two ways for the classifier to be wrong: it can produce false negatives and false positives. Using TP, FP, TN, and FN, we can calculate statistical measurements on the performance of our classifier such as **precision**, **recall**, **F_1 score**, and **null accuracy**.

Precision

Precision refers to the ratio between the true positives, and the true positives and false positives. It is also known as the **positive predictive value**.

$$P = \frac{TP}{TP + FP}$$

Recall

Recall, then, refers to the ratio between the number of true positives, and the true positives and false negatives. It is also known as the **sensitivity**.

$$R = \frac{TP}{TP + FN}$$

Accuracy

The **accuracy** is the ratio between the true positives and the true negatives, and the true positives, true negatives, false positives, and false negatives. **Accuracy** alone is a particularly bad quantifier of how well a classifier performs when working with data which is class-imbalanced (i.e. there are not equal numbers of items in each class). In our ground truth dataset, the classes are balanced so measuring accuracy will still be informative.

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

F1 Score

The F_1 **score** is a weighted average of the **precision** and **recall**. Thus, it takes both false positives and false negatives into account.

$$F1 = \frac{2 * (P * R)}{P + R}$$

Null Accuracy

The **null accuracy** is the accuracy which is obtained by always predicting the most frequent class. Because there are two classes and the tweets within the ground truth dataset equally compose both, the **null accuracy** will always be 50%.

2.2.6 K-Folds Cross-Validation

Instead of using the entire ground truth dataset as training data for the Naive Bayes classifier, we can split it into a **training set** and a **test set**. The **training set** is fed to the classifier, and the **test set** is used to observe statistics about its performance. Figure 2.2.1 depicts cross-validation using k-folds, in which we make random splits in the dataset k times to create several **training set** and **test set** sections. In k-folds, we then take statistical measurements of how well the classifier performs on the **test set** for each fold. If we made one single split into training and testing sections of our ground truth dataset and only used that single **test set** to gather statistics on the classifier's performance, we would not be able to confirm that those statistics were representative of all the data in the entire ground truth dataset. Instead, we perform 10-fold cross validation, choosing 90% of the data to be the **training set**, and 10% to be the **test set** in each fold. Precision, F_1 score, and recall are calculated within each iteration of the 10 folds, thus utilizing 10 different **test sets**. Finally, we use the averages of those statistics across all folds to measure the overall performance of the classifier.

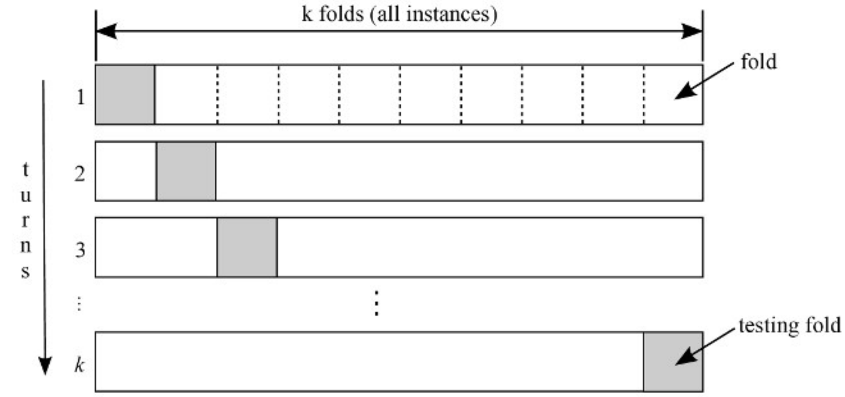

Figure 2.2.1. K-Folds Cross-Validation Example (Borovicka et al., 2012)

2.2.7 Training & Testing Naive Bayes

We utilize Scikit Learn's API for machine learning (Pedregosa et al., 2011) to create a `pipeline`. In Scikit, `pipelines` make managing machine learning algorithms easy by consolidating their parts into one object with configurable attributes.

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

Our `pipeline` contains a vectorizer and a classifier. We change the default arguments for Scikit's TF-IDF vectorizer to use NLTK's tweet tokenizer, and specify that we want to calculate our TF-IDF vectors using unigrams, bigrams, and trigrams. Then, we set the vectorizer to use Scikit's default English language stop words. The Multinomial Naive Bayes Classifier we implement using Scikit includes **Laplace smoothing** ($\alpha = 1$) by default.

Scikit also has a convenient `KFold` object which we utilize to perform cross-validation on our classifier. The goal of k-folds cross-validation is to train the classifier and acquire statistics on its performance while treating k different parts of the ground truth dataset as **test sets**. Within a single iteration for 10 iterations, the **test set** will always be 10% of the entire ground truth dataset, however the next iteration will use a different section. We ultimately average those statistics to understand the overall performance of the classifier.

```python
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)
    for train_index, test_index in 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)
        cnf_matrix_test += confusion_matrix(class_test, predictions_test)
```

In each iteration of the 10 folds, the above program splits apart a training set and a **test set**. The classifier is trained on the **training set** using `sentiment_pipeline.fit`, and the classifier's predictions for the **test set** in that fold are used to add toward a confusion matrix which will categorically visualize the performance of the classifier. We also calculate the **precision**, **recall**, and F_1 **score** for each fold's individual **test set**.

2.3 Creating the Twitter Bot

We use Tweepy to interact with the Twitter API. It provides a convenient object for streaming Twitter data in real time. The `StreamListener` class can track tweets by searching for specific users, locations, and keywords. For our purposes, it has to be extended to track subtweets.

```python
class StreamListener(tweepy.StreamListener):
    def on_status(self, status):
        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
        text = status.full_text

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

        tokens = tokenizer.tokenize(text)

        english_tokens = [english_dict.check(token) for token in tokens]
```

```python
        percent_english_words = sum(english_tokens)/float(len(english_tokens))

        # Make sure the tweet contains some English
        is_english = False
        if percent_english_words >= 0.1:
            is_english = True

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

This part of the program is for live classification of subtweets and gathers information on tweet status objects as they are encountered. We extract data from the tweet object including the ID of that tweet and the text contained within it. We utilize the same techniques for cleaning and genericizing the tweet which we used in preparing our ground truth data for the classifier. The `pipeline` has a `predict_proba` method which takes as its input a list of strings and outputs an array of probabilities for each class. `subtweet_probability`, then, uses that method to predict the probability that the tweet fits the "subtweet" class according to the Naive Bayes classifier and the vectorizer we used in our `pipeline`. We also check that the potential subtweet meets specific requirements before the Twitter bot will interact with it.

```python
if all([subtweet_probability >= THRESHOLD,
        "RT" != text[:2], is_english,
        not retweeted, not in_reply_to]):
```

Included in this conditional statement is a comparison to determine if the probability that the tweet is a subtweet meets a specific threshold. We do not want to call out subtweets unless the probability is high enough. Following this check, we can interact with the tweet in several ways.

```python
# Quote the tweet
api.update_status(("Is this a subtweet? {:.2%} \n" +
                   "https://twitter.com/{}/status/{}").format(subtweet_probability,
                                                              screen_name,
                                                              id_str))
# Like the tweet
api.create_favorite(id_str)

# Reply to the tweet
api.update_status("@{} Is this a subtweet? {:.2%}".format(screen_name,
                                                         subtweet_probability),
                  id_str)
```

To quote the potential subtweet means that the tweet being referenced is embedded within our own tweet with a caption above it. Finally, we instantiate our custom `StreamListener` class and use it to filter through tweets in real time.

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

stream.filter(follow=user_ids, stall_warnings=True, languages=["en"], async=True)
print("Streaming has started.")
sleep(DURATION)
stream.disconnect()
```

The `user_ids` list contains strings of Twitter user IDs. Every time a user whose ID is in the list sends a tweet, the program will classify that tweet and use the predicted probability that it is a subtweet to call it out on Twitter from the account linked to the application's API credentials. We filter the stream asynchronously in order to use the `sleep` function from the `time` Python library, so we can run the Twitter bot for a limited number of seconds.

3
Results

3.1 Ground Truth Dataset

The acquisition of the ground truth dataset relies on assumptions and generalizations of how Twitter users accuse others of subtweeting. For each subtweet and non-subtweet we acquired, we **did not** also acquire every single reply to that tweet, because doing so would violate the API's rate limits (Twitter, 2018). Instead, we maintain that using Twitter's search API, a single reply to a tweet is enough to indicate that it **is** or **is not** a subtweet. By ignoring the associated metadata of tweet objects and their replies, we obtain the unicode text contained within subtweets and non-subtweets for our ground truth dataset.

True Subtweets	True Non-Subtweets
I know who I want to take me home	He's followed the putrid smell of GENERIC_MENTION which has led him to GENERIC_MENTION's whereabouts. Some odd neighbor boy was watching him as he approached the house. Liam didn't hesitate to drink the young one dry. "HONEY, I'M HOME!" He calls, kicking in the door.
One bad chapter doesn't mean your story is over	YG so low? I guess the only group that sold physicals well is Bigbang GENERIC_URL
What do you offer someone who doesn't like tea, coffee, or hot chocolate? YOU'RE A GUEST, YOU NEED A HOT DRINK, IT IS THE LAW.	is everything ok?? GENERIC_URL
on the flip side, i agree that "big" accounts have an obligation to do their due diligence to not sic their followers on others	The parade of falsehoods about CIA nominee Gina Haspel GENERIC_URL
im bout to drop like 20lbs and then become a rapper.	Very first day of fortnite got 2 second place games and a third place gg not bad for a trash player :p
It's hilarious that your only "godly" when it's convenient for you or when you want to put it on twitter.	Rosie the Corpse Flower bloomed at Tucson Botanical Gardens: See pix allery here GENERIC_URL #Tucson GENERIC_MENTION GENERIC_URL
It really bothers me when people try to take advantage or assume I do shoots for free GENERIC_URL	What is Uncle's GREATEST prediction?
I just don't get people lol	Didi Gregious is my new favorite player.
y'all kids today with your "waifu this" and "waifu that," back in my day our waifus had metal jaws and carried high-frequency blades	GENERIC_MENTION we need to get you drawn as Dekamaster Doggy Krueger
I follow this gorgeous dog account on insta and his owner has such an annoying voice. SHUT UP AND LET ME ENJOY YOUR MAGNIFICENT DOG! I'M NOT HERE FOR YOU!	"Huge Caravan" Of Central Americans Is Headed For The U.S. Border In Hopes Of Asylum GENERIC_URL

When the vectorizer runs over these strings in order to tokenize them and extract features for use in the classifier, GENERIC_URL, GENERIC_MENTION, and GENERIC_NAME stand in place of actual URLs, mentions of user names, and English first names in the original tweets. We use these generic tokens because we want the classifier to probabilistically acknowledge the presence of these features while specifically ignoring the content within them.

3.1.1 Characteristics of the Ground Truth Dataset

With tens of thousands of tweets contained within the ground truth dataset, understanding the characteristics of these tweets by example alone is insufficient. To improve our overall understanding of these characteristics, we gathered statistics on the distributions of tweet lengths, punctuating characters, stop words, and unique words in both the **true subtweets** and true **non-subtweets** parts of the dataset.

Twitter serves as a micro-blogging framework which limits the lengths of English text posts to 280 characters. Because Twitter is also a multimedia platform which supports embedding URLs, images, and videos, tweets do not necessarily contain this maximum number of characters. In contrast to the popularity of multimedia tweets, subtweets are characterized as exclusively text-based. Between both **true subtweets** and **true non-subtweets**, Figure 3.1.1 illustrates that the ground truth dataset contains more short tweets ($1 \leq length \leq 125$) than long tweets ($125 < length \leq 280$).

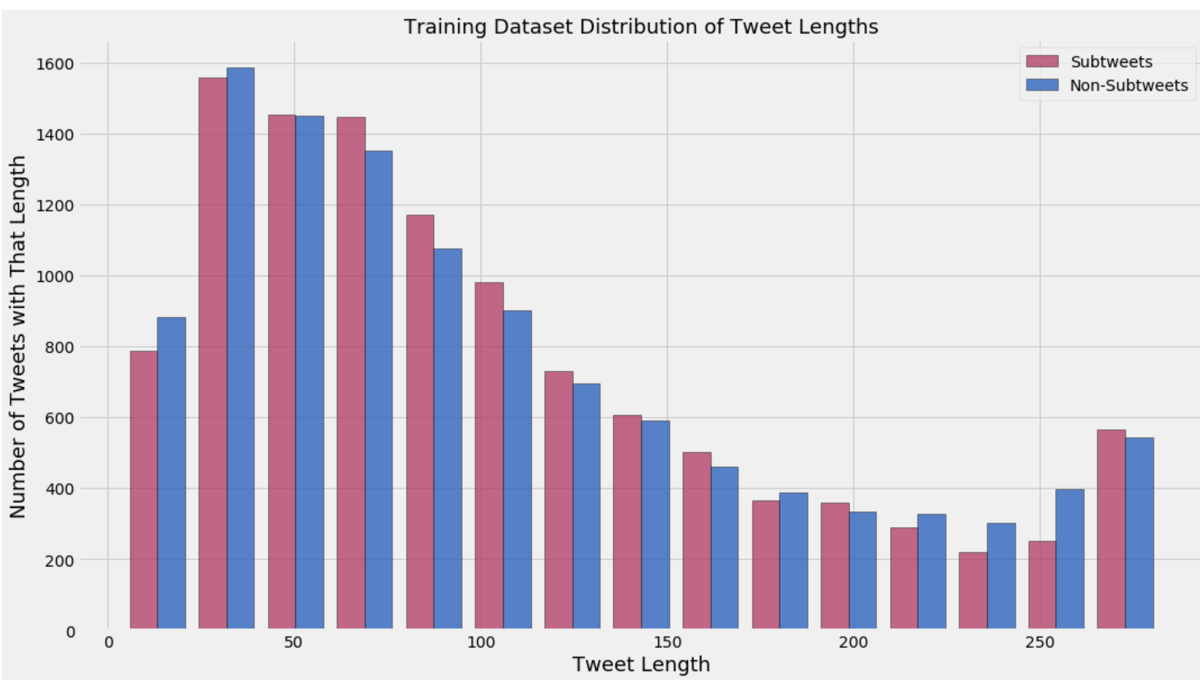

Figure 3.1.1. Histogram of Ground Truth Tweet Lengths
$(\mu_s = 103.71, \sigma_s = 71.99), (\mu_n = 105.63, \sigma_n = 74.95)$

Twitter does not enforce any grammar or syntax regulations in tweets, thus punctuating characters are often ignored altogether for use in the stream-of-consciousness style of writing. As shown in Figure 3.1.2, both **true subtweets** and **true non-subtweets** follow this trend in omitting punctuation, on average containing just under 2 punctuating characters (such as quotation marks, periods, commas, and apostrophes).

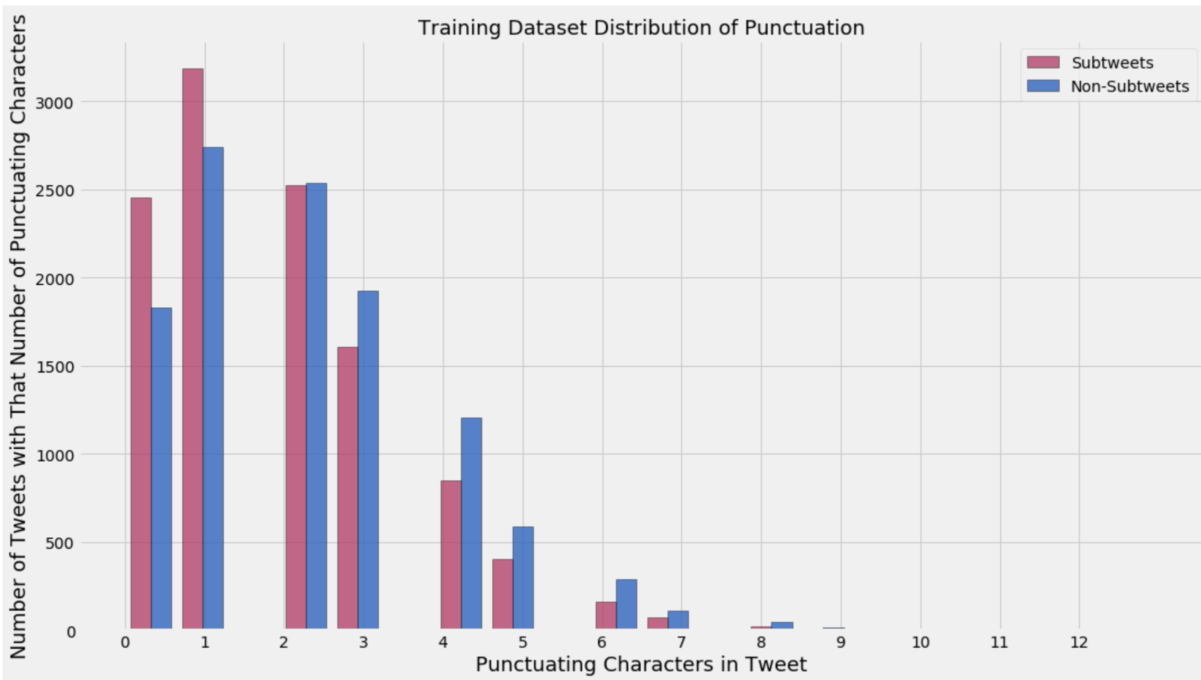

Figure 3.1.2. Histogram of Ground Truth Tweet Punctuation
$(\mu_s = 1.79, \sigma_s = 1.54), (\mu_n = 2.16, \sigma_n = 1.68)$

English stop words are ignored by the vectorizer because they are too common to help the classifier distinguish between the appropriate features for the classes. As shown in Figure 3.1.3,our ground truth dataset contains tweets which are more likely to contain between 5 and 8 unique stop words than they are to contain fewer. In comparison Figure 3.1.4 illustrates the distribution of the number **unique English words** per tweet in the ground truth dataset. As we expect of text-based posts within 280 characters in length, tweets containing fewer unique English words are more common.

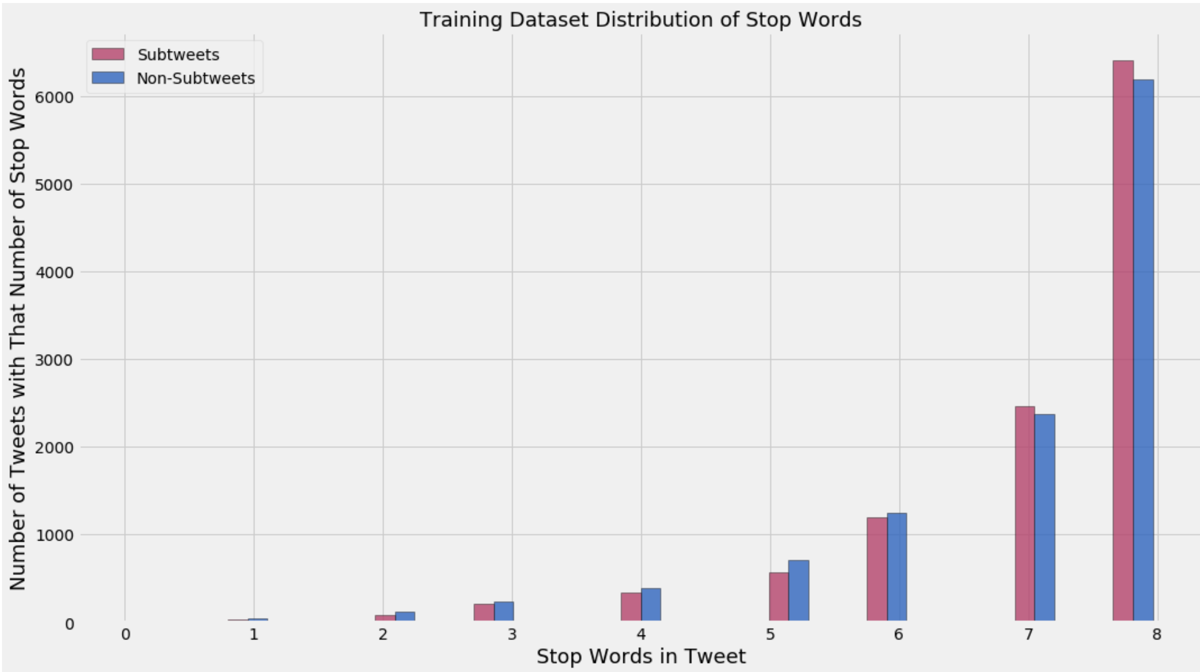

Figure 3.1.3. Histogram of Ground Truth Unique English Stop Words
$(\mu_s = 7.14, \sigma_s = 1.31), (\mu_n = 7.05, \sigma_n = 1.39)$

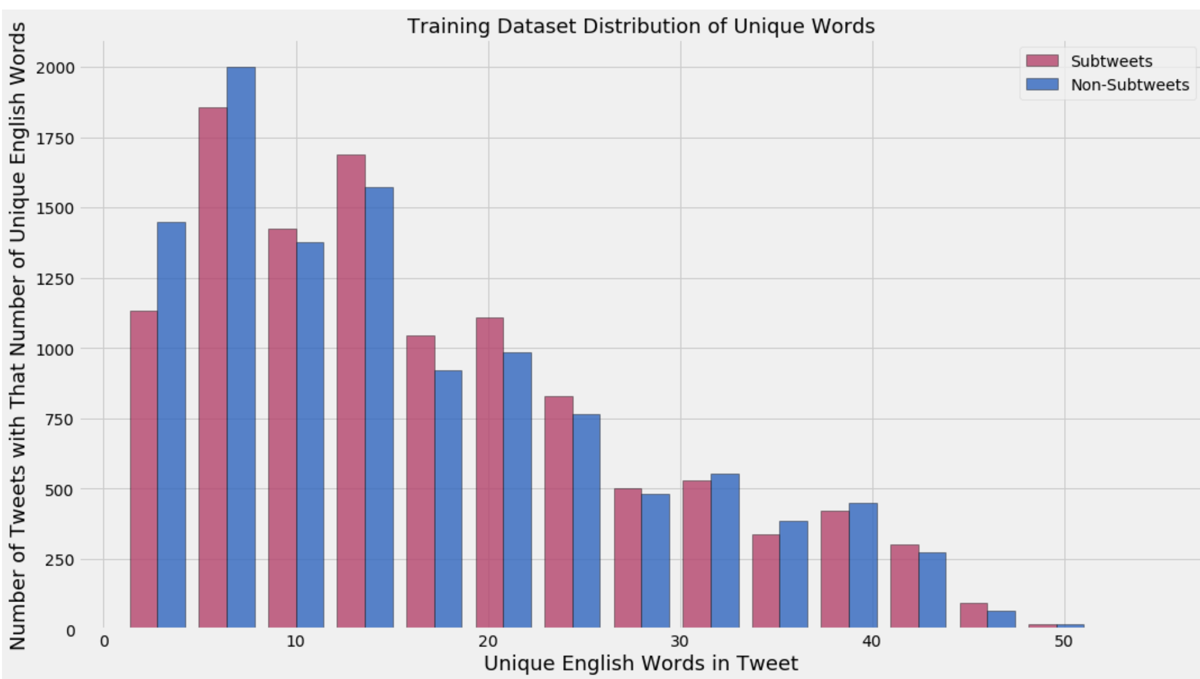

Figure 3.1.4. Histogram of Ground Truth English Unique Words
$(\mu_s = 16.80, \sigma_s = 10.95), (\mu_n = 16.25, \sigma_n = 11.15)$

3.2 Statistics

3.2.1 The Confusion Matrix

We used k-folds cross-validation to test the performance of our Naive Bayes classifier on each **test set** fold of our ground truth dataset. Each individual **test set** was composed of only 10% of the tweets in the entire dataset. By keeping track of the classifier's predictions on this **test set**, we accumulated a **confusion matrix** of **true positives**, **true negatives**, **false positives**, and **false negatives** for all 10 folds. Figure 3.2.1 illustrates these outcomes in terms of raw counts and normalized over the entire ground truth dataset.

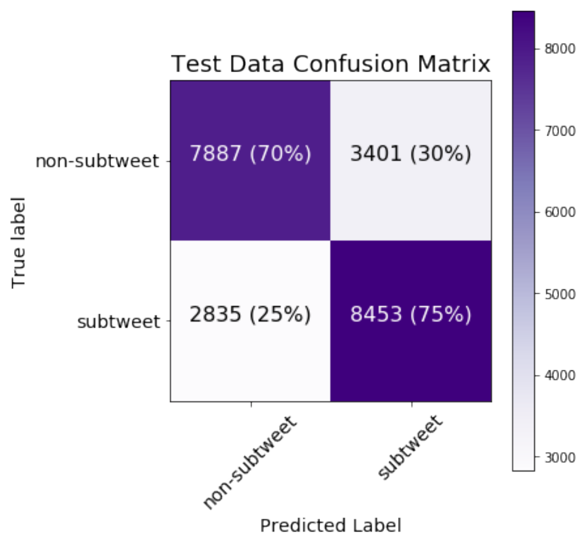

Figure 3.2.1. Confusion Matrix on Test Data from Each Fold on the Ground Truth Dataset

We read the confusion matrix by matching rows with columns to see how well the classifier performs on **true subtweets** and **true non-subtweets** in making classifications to produce **predicted subtweets** and **predicted non-subtweets**. These results indicate that the Naive Bayes classifier performs 5% better when classifying true positives than it does when classifying true negatives. In comparison, it is 5% **worse** at classifying false positives than it is at classifying false negatives.

3.2.2 Precision, Recall, & F1

The k-folds cross-validation we utilized to measure the statistical performance of our classifier was averaged across all 10 folds.

	Precision	Recall	F_1 Score
Non-Subtweets	0.7357	0.6988	0.7166
Subtweets	0.7132	0.7490	0.7305

The average F_1 scores for non-subtweets and subtweets are respectively 71.66% and 73.05%. Thus, our classifier performs similarly well on both.

3.3 Known Subtweeters

In order to test how the classifier performed on tweets from known subtweeters, we acquired all publicly available tweets from 11 different accounts from which users had previously subtweeted. Figure 3.3.1 shows the distributions of subtweet probabilities produced by the classifier on all these accounts.

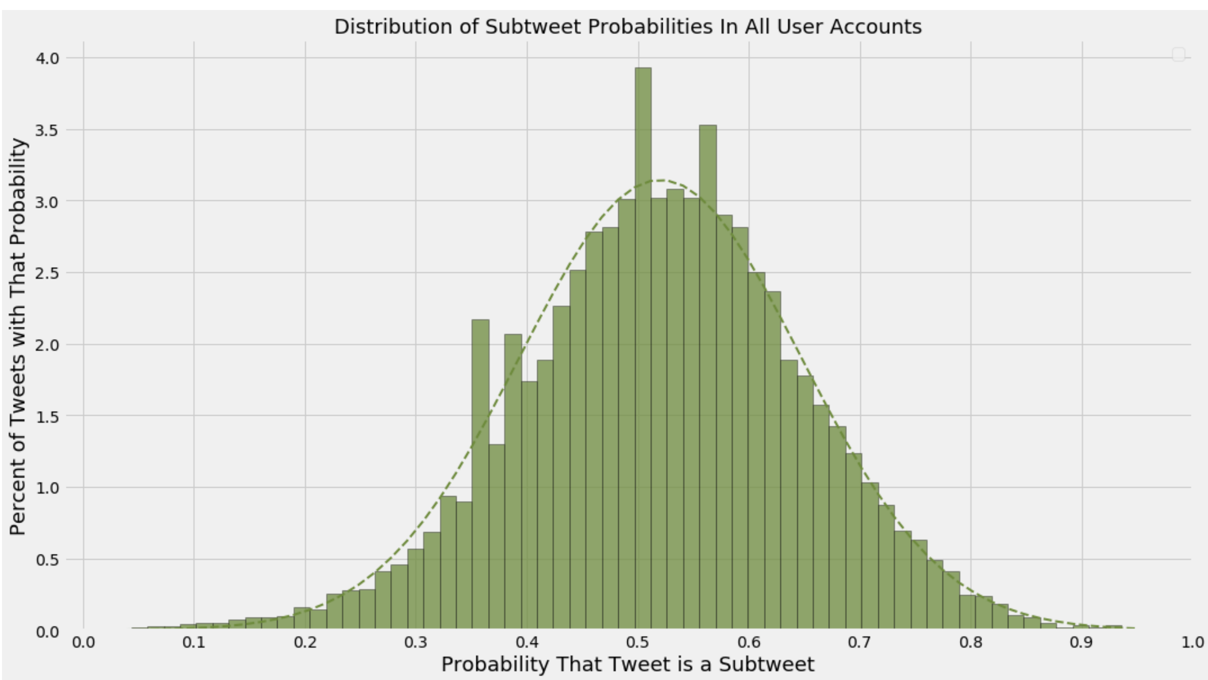

Figure 3.3.1. Distribution of Subtweet Probabilities for 11 Known Subtweeters($\mu = 0.520, \sigma = 0.127$)

Of these 11 accounts, the Naive Bayes classifier predicts that out of 26,928 tweets, 15,538 were subtweets (i.e. the predicted subtweet probability was at least 50%). In comparison, if we only accept classifications of at least 75%, then just 924 tweets (3.4%) were predicted to be subtweets. Of these accounts, we selected the users who produced the minimum percentage of subtweets (27.6%), the median percentage of subtweets (58.8%), and the maximum percentage of subtweets (68.4%). Figure 3.3.2 shows the distributions of the subtweet probabilities for all the tweets from these users.

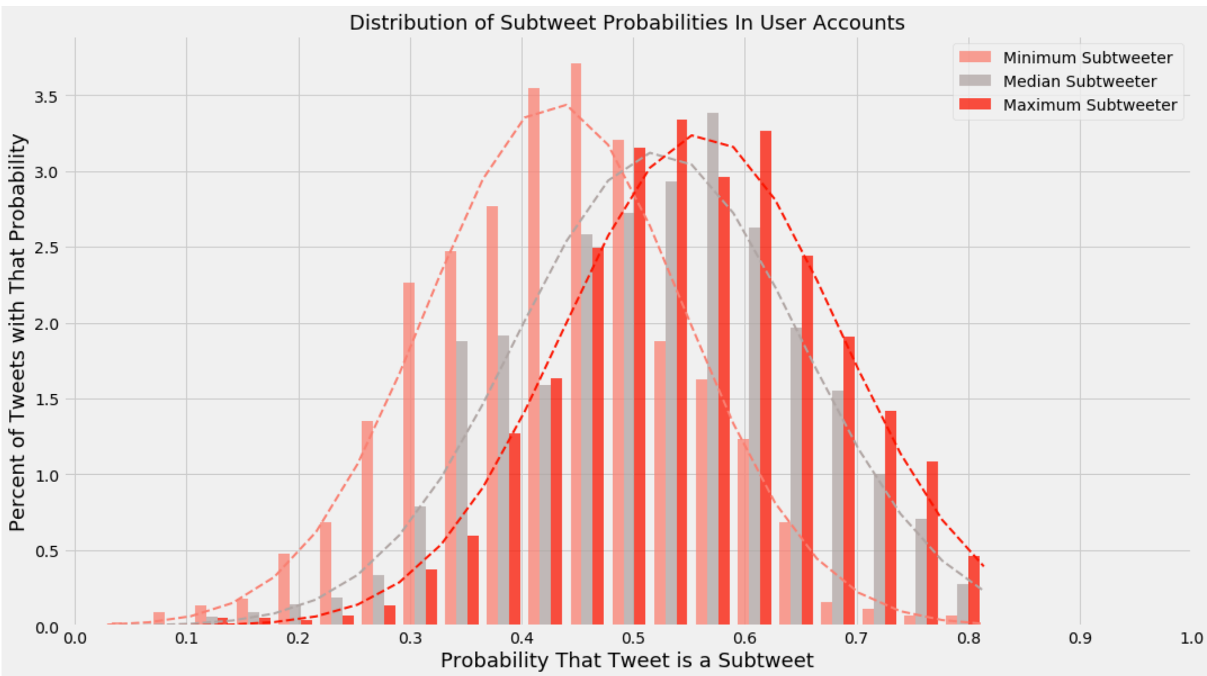

Figure 3.3.2. Distribution of Subtweet Probabilities for 3 Known Subtweeters
($\mu_{min} = 0.430, \sigma_{min} = 0.116$), ($\mu_{med} = 0.523, \sigma_{med} = 0.128$), ($\mu_{max} = 0.561, \sigma_{max} = 0.123$)

3.4 Most Informative Features

By iterating through the entire vocabulary of features on which the classifier was trained, and sorting those features by the logarithm of the estimated probability that the feature belongs in the subtweet class, we are able to see which features are most informative for classifying tweets as **predicted subtweets**.

Feature	Log Probability
.	-7.655317955479811
,	-8.022304498271094
GENERIC_URL	-8.048671040237405
”	-8.148437784201970
people	-8.427022837390009
?	-8.552068283434405
like	-8.672840985124703
don’t	-8.682709277848670
just	-8.744231174479117
i’m	-8.832921046724476
!	-8.863926249605123
it’s	-9.079518769840222
. GENERIC_URL	-9.100407613290404
:	-9.109587924777987
know	-9.183559311149267
...	-9.204228807382403
you’re	-9.210332638698153
twitter	-9.283961663904783
love	-9.328945234571764
*	-9.404169039956173
friends	-9.418393807734482

The only bigram in the top 20 most informative features is “. GENERIC_URL,” which can only occur when a URL follows the end of a sentence. In comparison, the rest of these terms are unigrams of which 8 are exclusively punctuating characters (period, comma, quotation, question, exclamation mark, colon, ellipsis, and asterisk). It comes as no surprise that subtweeters talk a lot about Twitter.

3.5 The Twitter Bot

After training and testing our classifier, we utilized it in creation of a Twitter bot which interacts with **predicted subtweets** in real time. By limiting the minimum **predicted subtweet** probability threshold to 75%, we prevent the Twitter bot from interacting with users too much. Figures 3.5.1 and 3.5.2 show (censored) examples of these interactions.

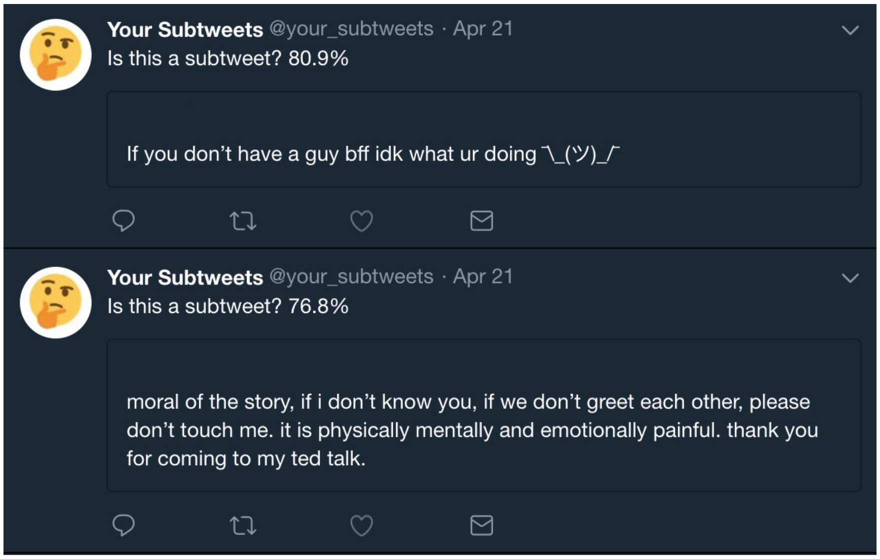

Figure 3.5.1. Example of the Twitter Bot Quoting Users' Tweets

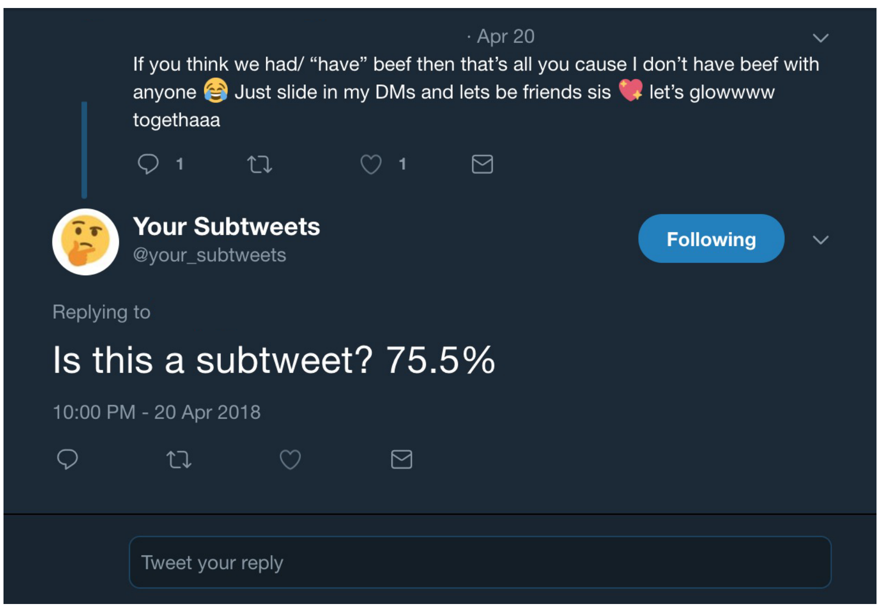

Figure 3.5.2. Example of the Twitter Bot Replying to Users' Tweets

3.6 Discussion

We obtained 12,169 **true subtweets** and 21,411 **true non-subtweets** to be used in our ground truth dataset. After confirming they were English and removing duplicates, there were 11,288 **true subtweets** and 19,289 **true non-subtweets** remaining. In order to preserve a balance between the classes, we limited the size of **true non-subtweets** to be the same as the set of **true subtweets**. Thus, our final ground truth dataset contains 22,576 tweets. Our classifier identifies non-subtweets with an F_1 **score** of 71.66% and subtweets with an F_1 **score** of 73.05%. Subtweets have garnered attention from news organizations (Madrigal, 2014), social scientists (Edwards and Harris, 2016), and governmental officials (Ohlheiser, 2017), but sentiment analysis of subtweets has been entirely unresearched. Utilizing the Twitter API, we acquired data for training and testing a Naive Bayes classifier, and developed a Twitter bot which actively calls-out subtweets in real time.

4
Conclusion

4.1 Summary of Project Achievements

We treated a new colloquial form of online harassment—"subtweeting"—as a text classification problem and used the probabilistic Naive Bayes classification algorithm to identify it. We judged the performance of the algorithm and went beyond identification to engage with subtweets, promoting publicity for content which is deliberately written to be unseen by the targeted party. We utilized Twitter's API to demonstrate a potential use for sentiment analysis on this kind of text.

All data acquired and all programs developed for this project have been made publicly available on GitHub (Noah Segal-Gould, 2018).

4.2 Future Work & Considerations

Resources played an important role in this project. With more time, more training data can be acquired. With funds, the Twitter API can be accessed through one of its paid plans to search for tweets from up to 30 days in the past. Because we utilized the free API, the maximum age of an alleged subtweet or non-subtweet when it was acquired could only be one week.

Our implementation of Naive Bayes exclusively classified using features from the unicode text contained within tweets, but other features related to the metadata contained within tweet objects and their replies will probably prove fruitful in producing a better subtweets classifier. We did not test other classification algorithms in favor of pursuing Naive Bayes singularly, but there exists no reason to **not** utilize others. Using topic modeling via methods such as non-negative matrix factorization and latent dirichlet allocation, we can analyze the topics about which most users are tweeting, and potentially group these users into subtweet-topic networked communities.

Finally, we treated this project as a binary classification problem between subtweets and non-subtweets. What would we find if we trained a classifier to distinguish ironic tweets, sarcastic tweets, mocking tweets, and subtweets? What are the features of the figurative language used in each of these examples? These questions motivate further research.

Bibliography

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

Stone, Biz. 2006. *Introducing the Twitter API*, available at `https://blog.twitter.com/official/en_us/a/2006/introducing-the-twitter-api.html`.

Twitter. 2018. *Twitter API Docs*, available at `https://developer.twitter.com/en/docs`.

______. 2016. *Twitter Terms of Service*, available at `https://twitter.com/en/tos`.

Roesslein, Joshua. 2009. *tweepy Documentation* **5**, available at `http://docs.tweepy.org/en/v3.5.0/`.

Twitter. 2018. *Tweet objects*, available at `https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object`.

Bird, Steven and Edward Loper. 2004. *NLTK: the natural language toolkit*, Proceedings of the ACL 2004 on Interactive poster and demonstration sessions, 31.

Zhang, Harry. 2004. *The optimality of naive Bayes*, AA **1**, no. 2, 3.

Borovicka, Tomas, Marcel Jirina Jr, Pavel Kordik, and Marcel Jirina. 2012. *Selecting representative data sets*.

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

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

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

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

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

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

Ohlheiser, Abby. 2017. *A running list of all the possible subtweets of President Trump from government Twitter accounts*, available at `https://goo.gl/hFh81R`.

Twitter. 2012. *Twitter turns six*, available at `https://blog.twitter.com/official/en_us/a/2012/twitter-turns-six.html`.

Dorsey, Jack. 2006. *inviting coworkers*, available at `https://twitter.com/jack/status/29`.

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

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

Allport, Gordon W. 1954. *The nature of prejudice.*

Ghosh, Aniruddha, Guofu Li, Tony Veale, Paolo Rosso, Ekaterina Shutova, John Barnden, and Antonio Reyes. 2015. *Semeval-2015 task 11: Sentiment analysis of figurative language in twitter*, Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), 470–478.

Van Hee, Cynthia, Els Lefever, and Vronique Host. 2018. *Semeval-2018 Task 3: Irony detection in English Tweets*, Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018).

Guilbeault, Douglas and Samuel Woolley. 2016. *How Twitter Bots Are Shaping the Election*, available at `https://goo.gl/ez155d`.

Bhatnagar, Ranjit. 2018. *Pentametron*, available at `https://twitter.com/pentametron`.

Medhat, Walaa, Ahmed Hassan, and Hoda Korashy. 2014. *Sentiment analysis algorithms and applications: A survey*, Ain Shams Engineering Journal **5**, no. 4, 1093–1113.

Kelly, Ryan. 2016. *PyEnchant: a spellchecking library for Python.*

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, and Vincent Dubourg. 2011. *Scikit-learn: Machine learning in Python*, Journal of machine learning research **12**, no. Oct, 2825–2830.

Twitter. 2018. *Rate Limiting*, available at `https://developer.twitter.com/en/docs/basics/rate-limiting`.

Noah Segal-Gould. 2018. *Senior-Project-Subtweets*, available at `https://github.com/segalgouldn/Senior-Project-Subtweets`.

Appendix A

Acquiring Ground Truth Data

A.1 subtweets_downloader.py

```
# coding: utf-8

# #### Script for downloading a ground truth subtweets dataset

# #### Import libraries for accessing the API and managing JSON data

# In[ ]:


import tweepy
import json


# #### Load the API credentials

# In[ ]:


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


# #### Authenticate the connection to the API using the credentials

# In[ ]:


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


# #### Connect to the API

# In[ ]:


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
```

```python

# In[ ]:


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

# In[ ]:


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

# In[ ]:


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

A.2 non_subtweets_downloader.py

```python
# coding: utf-8

# #### Script for downloading a ground truth non-subtweets dataset

# #### Import libraries for accessing the API and managing JSON data

# In[ ]:


import tweepy
import json

```

```python

# #### Load the API credentials

# In[ ]:


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


# #### Authenticate the connection to the API using the credentials

# In[ ]:


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


# #### Connect to the API

# In[ ]:


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

# In[ ]:


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

# In[ ]:


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

# In[ ]:


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

Appendix B

Training & Testing the Classifier

B.1 classifier_creator.py

```
# coding: utf-8

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

# #### In all tables, assume:
# * "GENERIC_URL" represents a single URL
# * "GENERIC_MENTION" represents a single mention of a username (e.g. "@noah")
# * "GENERIC_NAME" represents a single mention of an English first name

# #### Import libraries

# In[1]:


get_ipython().run_line_magic('matplotlib', 'inline')


# In[2]:


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

# In[3]:


urls_pattern = re.compile(r'(?i)\b((?:https?://|www\d{0,3}[.]|'
                          '[a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s('
                          ')<>]|\(([^\s()<>]+|(\([^\s()<>]+\))'
                          ')*\))+(?:\(([^\s()<>]+|(\([^\s()<>]'
                          '+\)))*\)|[^\s`!()\[\]{};:\'".,<>?'
                          '\xab\xbb\u201c\u201d\u2018\u2019]))')


# In[4]:


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


# In[5]:


names = open("../data/other_data/first_names.txt").read().split("\n")
names_pattern = re.compile(r'\b(?:{})\b'.format('|'.join(names)))


# #### Prepare English dictionary for language detection

# In[6]:


english_dict = enchant.Dict("en_US")


# #### Use NLTK's tokenizer instead of Scikit's

# In[7]:


tokenizer = nltk.casual.TweetTokenizer()


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

# In[8]:


pd.set_option("display.height", 1000)
pd.set_option("display.max_rows", 500)
pd.set_option("display.max_columns", 500)
pd.set_option("display.width", 1000)
pd.set_option("max_colwidth", 1000)


# In[9]:


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


# #### Load the two data files
# #### Only use tweets with at least 10% English words
```

```
# #### Also, make the mentions of usernames, names, and URLs generic

# In[10]:


def load_data(filename, threshold=0.1):
    data = [(urls_pattern.sub("GENERIC_URL",
             at_mentions_pattern.sub("GENERIC_MENTION",
             names_pattern.sub("GENERIC_NAME",
             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"]["lang"] == "en"
            and t["reply"]["lang"] == "en"
            and 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


# In[11]:


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


# In[12]:


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


# #### Remove tweets which are present in both datasets

# In[13]:


subtweets_data = [tweet for tweet in subtweets_data
                  if tweet not in non_subtweets_data]


# In[14]:


non_subtweets_data = [tweet for tweet in non_subtweets_data
                      if tweet not in subtweets_data]


# #### Show examples

# In[15]:


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

```python


# In[16]:


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


# #### Find the length of the smaller dataset

# In[17]:


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


# #### Cut both down to be the same length

# In[18]:


subtweets_data = sample(subtweets_data, smallest_length)


# In[19]:


non_subtweets_data = sample(non_subtweets_data, smallest_length)


# In[20]:


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


# #### Prepare data for training

# In[21]:


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


# In[22]:


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


# #### Combine them

# In[23]:


training_data = subtweets_data + non_subtweets_data


# #### Build the pipeline

# In[24]:


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

```python
    ("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

# In[25]:


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_accuracies = []
    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_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_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)

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


# In[26]:


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


# #### See the most informative features
# [How does "MultinomialNB.coef_" work?](https://stackoverflow.com/a/29915740/6147528)

# In[27]:


def most_informative_features(pipeline, n=15000):
    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)]})}


# In[28]:


most_informative_features_all = most_informative_features(sentiment_pipeline)


# In[29]:


most_informative_features_non_subtweet = most_informative_features_all["non-subtweet"]


```

```python
# In[30]:


most_informative_features_subtweet = most_informative_features_all["subtweet"]


# In[31]:


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(50)


# #### Define function for visualizing confusion matrices

# In[32]:


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

# In[33]:


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

# In[34]:


```

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


# #### Save the classifier for another time

# In[35]:


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


# #### Print tests for the classifier

# In[36]:


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(urls_pattern, "GENERIC_URL")
        user_df["Text"] = user_df["Text"].str.replace(at_mentions_pattern, "GENERIC_MENTION")
        user_df["Text"] = user_df["Text"].str.replace(names_pattern, "GENERIC_NAME")
        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

# In[37]:


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


# In[38]:


dataframes = process_tweets_for_testing(filenames)


# #### Show a random table

# In[39]:


chosen_username = choice(list(dataframes.keys()))
```

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


# #### Prepare statistics on tweets

# In[40]:


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


# In[41]:


test_df_stats = test_df.count()


# #### Total number of tweets per account:

# In[42]:


test_df_stats


# In[43]:


test_df_over = test_df[test_df >= 0.5]


# In[44]:


test_df_over_stats = test_df_over.count()


# #### Total number of classified subtweets (>= 50%) per account

# In[45]:


test_df_over_stats


# In[46]:


test_df_combined = pd.concat([test_df_over_stats, test_df_stats], axis=1)


# In[47]:


test_df_combined.columns = ["subtweets", "total_tweets"]


# In[48]:


test_df_combined["percent_subtweets"] = test_df_combined["subtweets"]/test_df_combined["total_tweets"]


# In[49]:


test_df_combined.sort_values(by="percent_subtweets", inplace=True)

```

```

# #### Overall stats

# In[50]:


test_df_combined


# #### Min subtweeter

# In[51]:


min_sub = (test_df_combined.loc[test_df_combined.percent_subtweets==test_df_combined.percent_subtweets.min()]
           .transpose())


# In[52]:


min_sub


# In[53]:


min_name = min_sub.columns[0]


# #### Median subtweeter

# In[54]:


med_sub = (test_df_combined.loc[test_df_combined.percent_subtweets==test_df_combined.percent_subtweets.median()]
           .transpose())


# In[55]:


med_sub


# In[56]:


med_name = med_sub.columns[0]


# #### Maximum subtweeter

# In[57]:


max_sub = (test_df_combined.loc[test_df_combined.percent_subtweets==test_df_combined.percent_subtweets.max()]
           .transpose())


# In[58]:


max_sub

```

```python

# In[59]:


max_name = max_sub.columns[0]


# #### Plot a histogram with three random users

# In[60]:


choices = [dataframes[min_name], dataframes[med_name], dataframes[max_name]]
scores = [df_dict["scores"][df_dict["scores"].columns[0]].tolist()
          for df_dict in choices]

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

n, bins, patches = ax.hist(scores,
                           bins="scott",
                           color=["#FA8072", "#B0A6A4", "#FC1501"],
                           density=True,
                           label=["Minimum Subtweeter", "Median Subtweeter", "Maximum Subtweeter"],
                           alpha=0.75)

stats = [df_dict["stats"][df_dict["stats"].columns[0]].tolist()
         for df_dict in choices]
for e in stats:
    print(e[1:3])
line_1 = scipy.stats.norm.pdf(bins, stats[0][1], stats[0][2])
ax.plot(bins, line_1, "--", color="#FA8072", linewidth=2)

line_2 = scipy.stats.norm.pdf(bins, stats[1][1], stats[1][2])
ax.plot(bins, line_2, "--", color="#B0A6A4", linewidth=2)

line_3 = scipy.stats.norm.pdf(bins, stats[2][1], stats[2][2])
ax.plot(bins, line_3, "--", color="#FC1501", 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

# In[61]:


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

# In[62]:


new_tests_df.describe()
```

```python


# #### Now plot

# In[63]:


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

n, bins, patches = ax.hist(new_tests_df["SubtweetProbability"].tolist(),
                           bins="scott",
                           color="#6E8B3D",
                           edgecolor="black",
                           density=True,
                           alpha=0.75)
print(new_tests_df_stats["SubtweetProbability"][1])
print(new_tests_df_stats["SubtweetProbability"][2])
line = scipy.stats.norm.pdf(bins, new_tests_df_stats["SubtweetProbability"][1],
                            new_tests_df_stats["SubtweetProbability"][2])

ax.plot(bins, line, "--", color="#6E8B3D", 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()


# #### Statisitics on training data

# #### Remove generic tokens for these statistics

# In[64]:


subtweets_data = [tweet[0] for tweet in subtweets_data]
non_subtweets_data = [tweet[0] for tweet in non_subtweets_data]


# #### Lengths

# In[65]:


length_data_subtweets = [len(tweet) for tweet in subtweets_data]
length_data_non_subtweets = [len(tweet) for tweet in non_subtweets_data]


# In[66]:


length_data_for_stats_subtweets = pd.DataFrame({"Length": length_data_subtweets,
                                                "Tweet": subtweets_data})
length_data_for_stats_non_subtweets = pd.DataFrame({"Length": length_data_non_subtweets,
                                                    "Tweet": non_subtweets_data})


# #### Tweet length statistics

# In[67]:


```

```
length_data_for_stats_subtweets.describe()


# In[68]:


length_data_for_stats_non_subtweets.describe()


# #### Punctuation

# In[69]:


punctuation_data_subtweets = [len(set(punctuation).intersection(set(tweet)))
                              for tweet in subtweets_data]
punctuation_data_non_subtweets = [len(set(punctuation).intersection(set(tweet)))
                                  for tweet in non_subtweets_data]


# In[70]:


punctuation_data_for_stats_subtweets = pd.DataFrame({"Punctuation": punctuation_data_subtweets,
                                                     "Tweet": subtweets_data})
punctuation_data_for_stats_non_subtweets = pd.DataFrame({"Punctuation": punctuation_data_non_subtweets,
                                                         "Tweet": non_subtweets_data})


# In[71]:


punctuation_data_for_stats_subtweets.describe()


# In[72]:


punctuation_data_for_stats_non_subtweets.describe()


# #### Stop words

# In[73]:


stop_words_data_subtweets = [len(set(stopwords.words("english")).intersection(set(tweet.lower())))
                             for tweet in subtweets_data]
stop_words_data_non_subtweets = [len(set(stopwords.words("english")).intersection(set(tweet.lower())))
                                 for tweet in non_subtweets_data]


# In[74]:


stop_words_data_for_stats_subtweets = pd.DataFrame({"Stop words": stop_words_data_subtweets,
                                                    "Tweet": subtweets_data})
stop_words_data_for_stats_non_subtweets = pd.DataFrame({"Stop words": stop_words_data_non_subtweets,
                                                        "Tweet": non_subtweets_data})


# #### Tweets stop words statistics

# In[75]:


stop_words_data_for_stats_subtweets.describe()

```

```

# In[76]:


stop_words_data_for_stats_non_subtweets.describe()


# #### Unique words

# In[77]:


unique_words_data_subtweets = [sum([english_dict.check(token)
                                    for token in set(tokenizer.tokenize(tweet))])
                               for tweet in subtweets_data]
unique_words_data_non_subtweets = [sum([english_dict.check(token)
                                        for token in set(tokenizer.tokenize(tweet))])
                                   for tweet in non_subtweets_data]


# In[78]:


unique_words_data_for_stats_subtweets = pd.DataFrame({"Unique words": unique_words_data_subtweets,
                                                      "Tweet": subtweets_data})
unique_words_data_for_stats_non_subtweets = pd.DataFrame({"Unique words": unique_words_data_non_subtweets,
                                                          "Tweet": non_subtweets_data})


# #### Tweets unique words statistics

# In[79]:


unique_words_data_for_stats_subtweets.describe()


# In[80]:


unique_words_data_for_stats_non_subtweets.describe()


# #### Plot them

# In[81]:


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

n, bins, patches = ax.hist([length_data_subtweets, length_data_non_subtweets],
                           bins="sturges",
                           edgecolor="black",
                           color=["#B4436C", "#3066BE"],
                           alpha=0.8)
ax.legend(["Subtweets", "Non-Subtweets"], loc="best")
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()


# In[82]:


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

```
ax = fig.add_subplot(111)

n, bins, patches = ax.hist([punctuation_data_subtweets, punctuation_data_non_subtweets],
                           bins="doane",
                           edgecolor="black",
                           color=["#B4436C", "#3066BE"],
                           alpha=0.8)
ax.set_xticks(np.arange(0, 13, step=1))
ax.legend(["Subtweets", "Non-Subtweets"], loc="best")
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()


# In[83]:


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

n, bins, patches = ax.hist([stop_words_data_subtweets, stop_words_data_non_subtweets],
                           bins="doane",
                           edgecolor="black",
                           color=["#B4436C", "#3066BE"],
                           alpha=0.8)

ax.legend(["Subtweets", "Non-Subtweets"], loc="best")
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()


# In[84]:


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

n, bins, patches = ax.hist([unique_words_data_subtweets, unique_words_data_non_subtweets],
                           bins="sturges",
                           edgecolor="black",
                           color=["#B4436C", "#3066BE"],
                           alpha=0.8)

ax.legend(["Subtweets", "Non-Subtweets"], loc="best")
ax.set_title("Training Dataset Distribution of Unique Words", fontsize=18)
ax.set_xlabel("Unique English Words in Tweet", fontsize=18)
ax.set_ylabel("Number of Tweets with That Number of Unique English Words", fontsize=18)

plt.show()
```

B.2 live_subtweets_classifier.py

```
# coding: utf-8

# ### Script for running a Twitter bot that interacts with subtweets

# #### Import some libraries

# In[1]:


```

```
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

# In[2]:


THRESHOLD = 0.75 # 75% positives and higher, only
DURATION = 60*15 # 60*60*24*7 # 1 week


# #### Set up regular expressions for genericizing extra features

# In[3]:


urls_pattern = re.compile(r'(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+'
                          '[.][a-z]{2,4}/)(?:[^\s()<>]|\(([^\s()<>]+|(\(['
                          '^\s()<>]+\)))*\))+(?:\(([^\s()<>]+|(\([^\s()<>'
                          ']+\)))*\)|[^\s`!()\[\]{};:\'".,<>?\xab\xbb\u201'
                          'c\u201d\u2018\u2019]))')


# In[4]:


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


# In[5]:


names = open("../data/other_data/first_names.txt").read().split("\n")
names_pattern = re.compile(r'\b(?:{})\b'.format('|'.join(names)))


# #### Load the classifier pipeline which was previously saved

# In[6]:


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


# #### Load the Twitter API credentials
```

```python

# In[7]:


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


# #### Connect to the API

# In[8]:


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

# In[9]:


subtweets_live_list = []
non_subtweets_live_list = []


# #### Use pyenchant to check if words are English

# In[10]:


english_dict = enchant.Dict("en_US")


# #### Use NLTK for tokenizing

# In[11]:


tokenizer = nltk.casual.TweetTokenizer()


# #### Create a custom StreamListener class for use with Tweepy

# In[12]:


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 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"]
            else:
                pass # Something else?
        elif "text" in status._json:
```

```
            text = status._json["text"]

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

        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_english = False
        if percent_english_words >= 0.1:
            is_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[:2],
                is_english, not retweeted, not in_reply_to]):
            try:
                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
            except:
                print("Unable to interact with tweet!")
        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

# In[13]:


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

# In[14]:


```

```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


# In[15]:


# %%time
# my_mutuals_mutuals = get_mutuals_and_mutuals_mutuals_ids()


# #### Load the IDs JSON

# In[16]:


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

```
# In[17]:


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


# #### Begin streaming

# In[18]:


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


# In[19]:


# %%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()
```