Tweet Sentiment Analysis

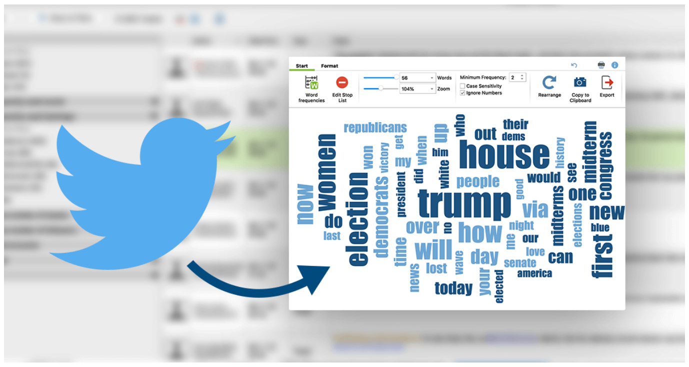
https://github.com/ryanherren/Tweet-Sentiment-Model

Team Members: Ryan Herren, Tanner Dunn, Mario Lloyd Galvao-Wilson, Jayant Shah

Abstract

The Tweet Sentiment Analysis Project is a Machine Learning Project that will evaluate single or multiple tweets and predict the sentiment (positive, negative, or neutral) based off of an opinion lexicon and a training set of tweets. Our model uses both Support Vector Machines and Decision Tree Classifiers to create predictions for the sentiment of each tweet, based off of quantity of negative and positive words and emojis in each tweet. The main outcome of this project is to be able to identify the sentiment behind a single tweet or a series of tweets. The series of tweets can be obtained from a user or from a timeline of tweets that a user follows.

1 Introduction



Understanding sentiment when reading tweets can sometimes be very difficult due to the lack of verbal inflection in typed/written language. Due to this, there are often misunderstandings in the true meanings and attitudes behind different texts. This project aims to build a Machine Learning Model that allows you to evaluate the sentiment behind tweets based off of the language, emojis, and other characters that they use. Tweets are obtained live from Twitter via the Twitter API and the Python package TweePy. Once handled, they are tokenized into words, emojis, mentions, and hashtags. Then, the tokenized tweet is run through the pre-trained model and evaluated using our pre-trained machine learning models to predict the sentiment (positive, negative, or neutral) of the tweet.

Data

We used data from the Kaggle Sentiment 140 dataset, which can be found here. This is a dataset of approximately 1.6 million tweets gathered the same way that we did- through the Twitter API. Diving into the dataset, it contained six main features about the set. The columns we used were the sentiment value (negative = 0, neutral = 2, and positive = 4), the tweet ID, text(the actual tweet itself), and the date of the tweet. The dataset also included flags and the user who tweeted it, but we had no use for those and disregarded them when loading in the dataset. After loading in the Kaggle140 data, we then reverse-applied our sentiment lexicon counting functions to the dataset. The first action was to tokenize the tweet (functions: tokenize, preprocess), which split the tweet into individual words, hashtags, mentions, and emojis. Second, we counted the number of positive and negative emojis and words from the tokenized tweet (functions: count_negative, count_positive, count_positive_emojis, count_negative_emojis, which provided four new columns for us to use when training our models and was the main quantitative data used in training. This was also the most time-intensive part of our code, as we iterated through positive-words.txt and negative-words.txt, which are the two files containing sorted sentiment words, which was a result from Bing Liu's research paper on sentiment words. For each word, these functions iterated through the sentiment word list and checked to see if it matched. For a sample of 15000 tweets, this process took approximately 20 seconds to occur, which, when applied to the full 1.6 million record dataset, took about a half an hour. This made loading and transforming the dataset every time we wanted the model to run not feasible. Instead, we settled for a random sample of 150000 tweets every time the script ran, which allowed for plenty of training data and variability in the training sets for the models. Lastly, we used similar functions to count the number of mentions and hashtags in each tweet, which was mostly ignored in training, but nice to have for experimentation.

A downfall of the Kaggle140 dataset was that it did not contain any emoticons. Nowadays, many people use emojis to communicate, and Twitter is no exception. To alleviate this issue, we generated approximately 50 tweets that contained emojis and assigned sentiment to them. We also produced a list of 40 negative and positive emojis, 20 of each. This list contained some of the emojis most used in either positive or negative ways. Having that data allowed us to not only analyze the words, but also the emojis used in each tweet. This data was joined to the original Kaggle140 dataset and used in training.

Code Structure

We used Jupyter Notebook as our file type for the project, allowing for a modular code base with the ability to rerun and test code in segments. First, we imported all necessary packages. The main packages used in our project were Tweepy and Sklearn, along with all other normal Python packages used in a Data Science environment, like pandas and numpy.

Tweepy version 3.7.0 was the package that interacted directly with the Twitter API and allowed for tweets to be brought in to our model in real time. To use the API, we were required to set up a Twitter Developer account in accordance with an existing Twitter account. For this, we created a Twitter account with the username @DS301Bot. We then generated access tokens and API secrets for this account, allowing us to access the account's timeline and other things accessible through the Twitter API. The information is pulled from Twitter in JSON form, so there needed to be some tricky code work in order to read the tweets, reformat them from JSON, and turn them into Pandas DataFrames that allowed for easy manipulation and training of models.

In order to keep our access tokens a secret, a YAML configuration file was created. The YAML file contained the necessary access tokens for the API and was stored only on local machines, keeping the secrets out of the code and our Twitter Developer information safe. Each of us downloaded the YAML file onto our local machines, meaning that all we had to do to activate the API locally was to change the OS Environment Python Bot Config file path to match our own local directory where it was stored.

From there, live Twitter data was stored in DataFrames and we were ready to begin manipulation and training.

Expected Outcomes

The primary expected outcome of our project was to accurately and efficiently predict sentiment of tweets in a way that was friendly to a user. We wanted a clean, easy way for an end-user to select a Twitter user and a machine learning model and get a quick look into what the sentiment of the words might truly be. We trained three different models for this project. The first was a Support Vector Machine, predicting only between positive and negative sentiment. The second was a Decision Tree Classifier with only two possible predicted sentiment values - positive and negative. Lastly, we trained a Decision Tree Classifier with the option to predict neutral sentiment in addition to positive and negative.

2 Related Work

The main related work to this project lies in the use of the Opinion Lexicon. The Opinion Lexicon was produced from a University of Illinois-Chicago study, led by Bing Liu, that began in 2004. The goal of the study was to identify words that are used primarily in a non-neutral setting in terms of sentiment. From the study, they were able to make an Opinion Lexicon, consisting of over 6800 positive or negative words that dictate true sentiment when used in a written manner.

This work is in the area of sentiment analysis and opinion mining from social media, e.g., reviews, forum discussions, and blogs. In their KDD-2004 paper, they proposed the Feature-Based Opinion Mining model, which is now also called Aspect-Based Opinion Mining (as the term feature here can be confused with the term feature used in machine learning). The output of such opinion mining is a feature-based opinion summary or aspect-based opinion summary. The commonly known sentiment classification is a sub-task.

- Sentiment Analysis or Mining of Regular Opinions

  In this research, they aim to mine and to summarize online opinions in reviews, tweets, blogs, forum discussions, etc. Specifically, we mine features or aspects of entities (e.g., products) or topics on which people have expressed their opinions and determine whether the opinions are positive or negative.

  Abstraction of the problem: Feature-based opinion mining and summarization (aspect-based opinion mining and summarization) of multiple reviews (KDD-04 and WWW-05). Formal definitions can be found in their book Sentiment Analysis and Opinion Mining. They are based on several of their papers in 2004 and 2005. The abstraction provides a model of online opinions, describes what should be extracted from opinion sources (e.g., reviews, forums, and blogs) and how the results may be organized and presented to the user. The main mining tasks are:

  - mining entities and their features (or aspects) that have been commented on or evaluated by people,
  - determining whether the comment/opinion on each entity feature (or aspect) is positive, negative or neutral (aspect-based sentiment classification), and
  - summarizing the results.

- Sentiment Analysis of Comparative Opinions

  A comparative sentence usually expresses an ordering relation between two sets of entities with respect to some shared features (or aspects). For example, the comparative sentence ”Canon's optics are better than those of Sony and Nikon” expresses the comparative relation: (better, optics, Canon, Sony, Nikon). Comparative sentences use different language constructs from typical opinion sentences (e.g., "Cannon's optic is great").

Abstraction of the problem: "which is better than which about what". Again, the formal definitions can be found in their book Sentiment Analysis and Opinion Mining. The main mining tasks are:

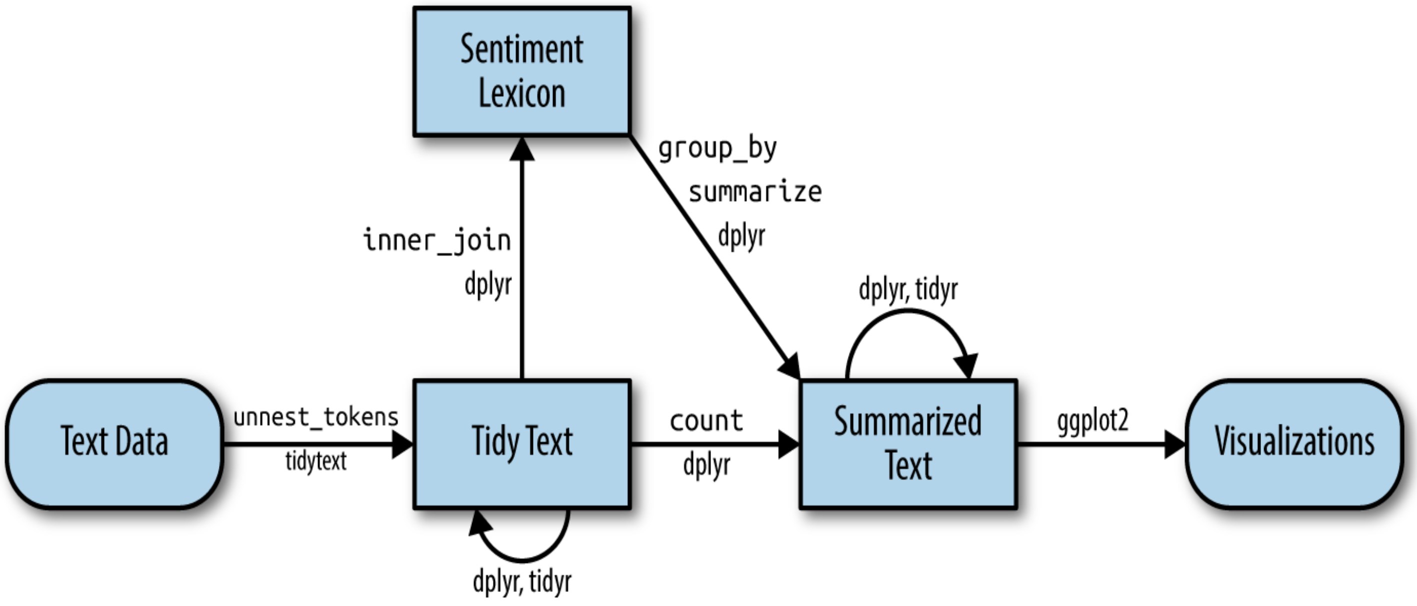
- identify comparative sentences from texts, e.g., reviews, forum or blog postings, and news articles.
- extract comparative relations from the identified comparative sentences.

From this, we decided upon using the foundation of Bing Liu's research as our main method in determining sentiment. We extracted their positive and negative words list -the opinion lexicon- and used it as our main quantitative method in determining sentiment.

3 Tweet Sentiment Analysis with SVMs and Decision Tree Classifiers

The main technique of semantic analysis that we used is the lexicon-based approach. This technique calculates the sentiment orientations of the whole document or set of sentence(s) from semantic orientation of lexicons. Semantic orientation can be positive, negative, or neutral. The dictionary of lexicons can be created manually as well as automatically generated. The WorldNet dictionary is used by many researchers. First of all, lexicons are found from the whole document and then WorldNet or any other online thesaurus can be used to discover the synonyms and antonyms to expand that dictionary.

Lexicon-based techniques use adjectives and adverbs to discover the semantic orientation of the text. For calculating any text orientation, adjective and adverb combinations are extracted with their sentiment orientation value. These can then be converted to a single score for the whole value.



Setting up

Before we can get started, you needed to make sure we have the following:

- Python 3 installed.
- The YAML Configuration file, containing DS301Bot's access tokens and API secrets.
- The DS301_Notebook.ipynb file that contains the code for the extraction, manipulation, training, and analyis of the data.

Authenticating and connecting to the Twitter API

To access the configuration file, you will need to point the OS Environment Variable 'python-bot-config' to your local version of the YAML file that has the access tokens:

```
consumer_key = cfg.get("TwitterAPI").get("consumer_key")
consumer_secret = cfg.get("TwitterAPI").get("consumer_secret")
access_token = cfg.get("TwitterAPI").get("access_token")
access_token_secret = cfg.get("TwitterAPI").get("access_token_secret")
```

Then, you will need to use Tweepy's OAuth handler to connect to the API with the correct tokens:

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth, wait_on_rate_limit=True)
```

Tweet Sentiment Model

- Data Collection

  The dataset is live tweets being brought into our ipynb file via the Twitter API. This means the data can be any series of tweets by a user or a series of tweets derived from a users timeline based off of who they follow. There is functionality with the version of Tweepy that we are using to load in Tweets that all contain a certain hashtag (e.g #DataScience) but we did not write the code to be able to handle those search requests.

- Data Pre-Processing

  The first step in processing of the data is the tokenizing methods. Tokenizing takes a tweet from a long string of words into a list of words, emojis, hashtags, and mentions that were contained in the original tweet. The method of tokenization we used in our notebook was through code that analyzed tweets based on custom regex statements. An example of how the tokenizing function works is as follows:

  Before tokenizing: "Wow, what an awesome day! #SoMuchFun"
  After tokenizing: "Wow", "what", "an", "awesome", "day", "#SoMuchFun"

- Feature Selection

  The features are as follows: user, mentions, emojis, URLs, hashtags, and words. The labels will be in correspondence with the training set used with Bing Liu's Opinion Lexicon (Hu and Liu, KDD-2004). Upon receiving this data, we then added the following columns: number of negative words, number of negative emojis, number of positive words, number of positive emojis, number of mentions, and number of hashtags.

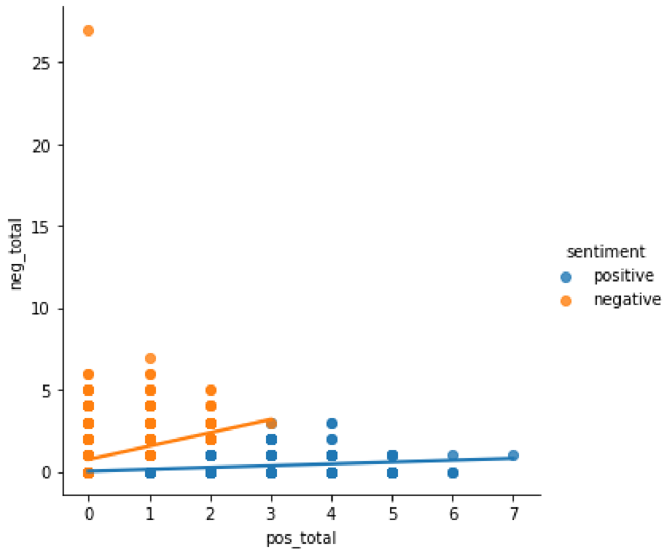
Training Models

To start, we identified three models that we wanted to train. We decided on one Support Vector Machine and two Decision Tree Classifiers, one with neutral value prediction, and one without.

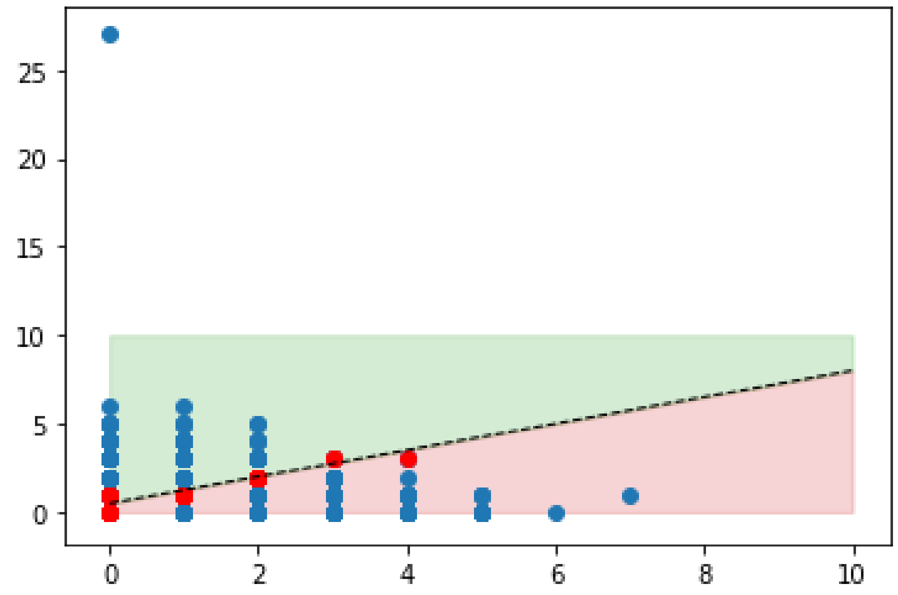
We first split our dataset into training and testing sets using Sklearn's train_test_split functionality. Once we did this, we vectorized our features into one array and our sentiment values (output) into another. We chose number of negative and positive words and emojis as our training features. From here, we began training our models.

Support Vector Machine

We first trained an SVM on the four aforementioned features and using the model to predict sentiments of either positive or negative. We aggregated the number of positive/negative emojis/words into two single statistics of neg_total and pos_total, which represent the total amounts of negative and positive tokens in each tweet. The graphic below shows the distribution of the data points, with rough decision lines for each sentiment value. The one outlier on the Y-axis is a datapoint that contains a tweet that repeats a swear word, one that is found in the negative-sentiment word list from the lexicon, 27 times.



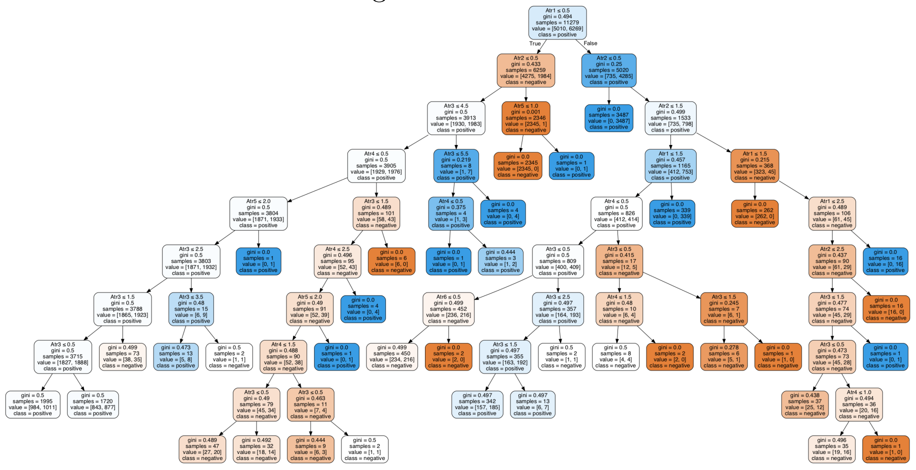
The following image shows the decision boundary generated by the model. Again, the outlier is ignored by the model in its decision boundaries, but we can generally assume that a tweet only containing swear words will be of negative sentiment. The Y-axis is negative count and the X-axis is positive count



We found that this model predicted sentiment correctly with accuracy of 77% on average over approximately 10 iterations. This model stuggled to predict the correct sentiment in tweets that had similar amounts of positive words and negative words. It was effectively able to identify tweets that were strongly negative or positive, but basically came down to guessing on tweets that were mostly neutral in sentiment.

Decision Tree Classifier - No Neutral

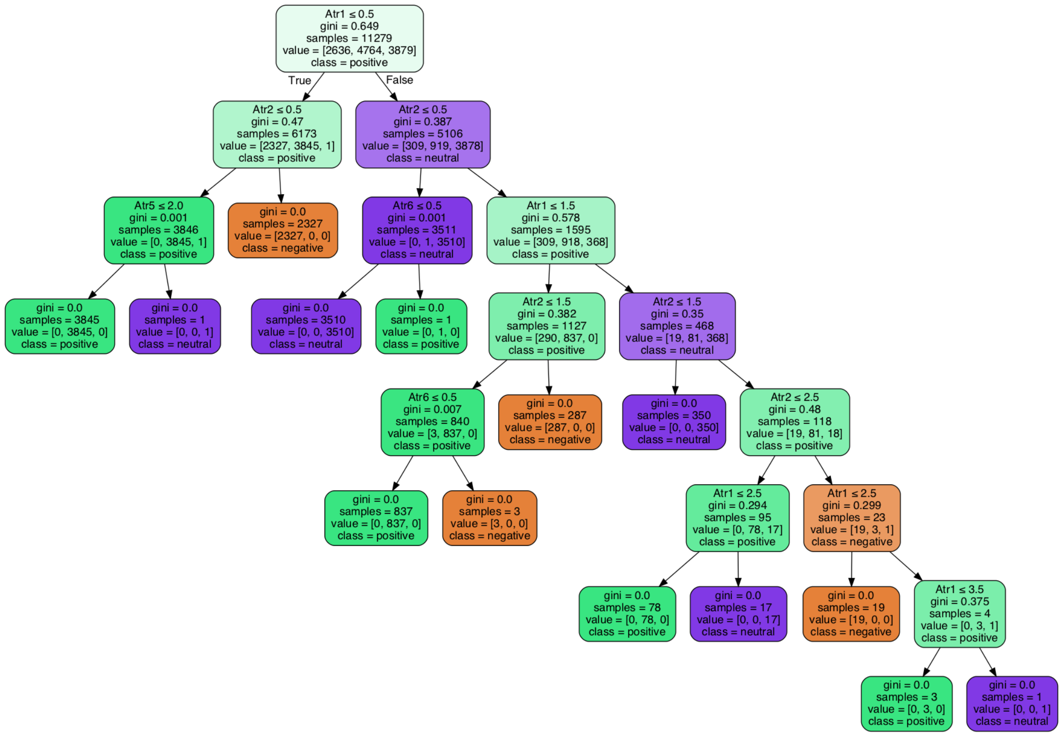
Following the conclusion of our SVM trials, we decided to test the efficiency of Decision Trees in comparison to the SVM. We resampled the full Sentiment140 dataset and then trained the Decision Tree. The resulting tree was as follows:



This tree ended up being fairly complex, but with relatively few paths that ended in confident results. In the image, negative results are denoted in orange, while positive results are denoted in blue. As the colors trend closer to white, the less confident the model was in being able to predict the sentiment of the tweet. On paths that were not strongly swayed in either sentiment direction, the model was basically making a 50/50 guess on the sentiment. Being such, the accuracy was only slightly better than the SVM, coming in at approximately 78.5% over 10 iterations. With the lack of increase in training accuracy, we realized that this could be a good space for another possible predicted sentiment category: neutral.

Decision Tree Classifier - Positive, Negative, Neutral

The training process for this Decision Tree Classifier model was the same as in the prior two models, but with one big difference. In this model, we allowed the Decision Tree to output a 'neutral' sentiment rating on tweets that it deemed both not negative and not positive. The resulting tree is shown below:



As you can see when comparing this tree to the first, the complexity was reduced significantly. This model was able to make decisions much easier due to the neutral category being introduced. In the tree above, negative decisions are denoted in orange, positive in green, and neutral in purple. You can see that this tree had a lot less issue classifying tweets, since the difficult decisions from the prior tree are now predicted into the neutral category. The neutral rating allowed for the model to only make difficult decisions between positive/neutral and neutral/negative. Anything that the model did not deem polarized enough in the negative or positive direction fell into the neutral category. As such, this model had over a 98% accuracy when predicting on the test split of the training dataset.

Results Overview

In the end, the neutral sentiment value ended up making the biggest impact on correct predictions. It allowed for our models to not have to guess on positive/negative sentiment for tweets that were generally neutral in sentiment. The majority of the error in our initial two models came from tweets that either had very few sentimentally-skewed words or had similar amounts of negative and positive words, and the neutral sentiment value alleviated those issues.

4 Conclusion

Future Plan

In the future we would like to do more testing and retrain our model with a larger variety of tweets with different themes and contexts to compare to. Something else that we think would help our model a lot is to train our model using a Twitter user's previous tweets to make a more accurate prediction by using their past language to predict the sentiment of their most recent tweets. However, the feasibility of this is questionable, since it would require each user to have their own training set to train off of. This would take a large amount of human interaction.

Additionally, we would like to be able to use an Opinion Lexicon with a range of weights applied to the words in the list. We hypothesize that this would make our models more accurate because negative words such as 'devastate' and 'catastrophe' would carry more weight than 'sick' or 'unreasonable' when scoring a tweet's sentiment. There is also a need to be able to recognize negation words within a tweet. For example, a user could say 'I am not happy' and our model might consider this a positive tweet because of the word happy. In this case, 'I am not happy' is obviously negative in sentiment. The drawback is that we would need to find a way to evaluate entire sentences/sections of language at a time to be able to accomplish this.

5 References

Bonzanini, Marco. "Mining Twitter Data with Python ." Marco Bonzanini, 19 July 2017, https://marcobonzanini.com/2015/03/02/mining-twitter-data-with-python-part-1/.

Liu, Bing. "Opinion Mining, Sentiment Analysis, and Opinion Spam Detection." Opinion Mining, Sentiment Analysis, Opinion Extraction, 15 May 2004, https://www.cs.uic.edu/ liub/FBS/sentiment-analysis.html.

Twitter. (n.d.). How to analyze the sentiment of your own tweets — docs — twitter developer platform. Twitter. Retrieved October 23, 2021, from https://developer.twitter.com/en/docs/tutorials/how-to-analyze-the-sentiment-of-your-own-tweets.

Twitter. (n.d.). Getting started with the Twitter API — docs — twitter developer platform. Twitter. Retrieved October 23, 2021, from https://developer.twitter.com/en/docs/twitter-api/getting-started/about-twitter-api.

Lexicon-based approach. Lexicon-Based Approach - an overview — ScienceDirect Topics. (n.d.). Retrieved December 11, 2021, from https://www.sciencedirect.com/topics/computer-science/lexicon-based-approach.