Frederick Lee
CS429: Spring 2021
Project 3: Naive Bayes

Section 1: Implementation

In analyzing my implementation of a Naive Bayes Classifier, I focus on how different approaches to tweet Tokenization change the accuracy of sentiment analysis with respect to the given testing and training data sets.

While the implementation of the core Bayes Classifier is contained entirely within the provided files, I pulled the Tokenizers and their related configuration options from the NLTK Tokenize python library module[1]. Note: for the *TweetTokenizer*, it was found that the *remove_len* parameter, which truncates the number of repeated characters in a given word to 3, had no measurable impact[2] on the accuracy of the testing and training data regardless of its activation. It has been left out of the analysis.

Tokenizer	Params
WhitespaceTokenizer[3]	N/A
WordPunctTokenizer[4]	N/A
TreebankWordTokenizer[5]	N/A
TweetTokenizer[6]	***preserve_case* (default: *True*)** *reduce_len* (default: *False*) ***strip_handles* (default: *False*)**

For this implementation, I implemented based of a shared Piazza article that described a loglikelihood representation[7]. The reason for implementing a log-likelihood model instead of a linear likelihood model was mainly due to floating-point stability, given that the span of log-probability representation (log(0) to log(1)) is much less susceptible to floating-point underflow, especially considering scenarios where multiple linear likelihoods near 0 are being multiplied together.

1 https://www.nltk.org/api/nltk.tokenize.html
2 That is to say, while likelihood calculations may have changed to some degree, the changes were not enough to change even 1 classification during evaluation on training and testing data.
3 https://www.nltk.org/_modules/nltk/tokenize/regexp.html#WhitespaceTokenizer
4 https://www.nltk.org/_modules/nltk/tokenize/regexp.html#WordPunctTokenizer
5 https://www.nltk.org/_modules/nltk/tokenize/treebank.html#TreebankWordTokenizer
6 https://www.nltk.org/_modules/nltk/tokenize/casual.html#TweetTokenizer
7 https://medium.datadriveninvestor.com/implementing-naive-bayes-for-sentiment-analysis-in-python-951fa8dcd928

Below is a table of testing and training accuracies for each relevant Tokenizer + Param combination

Tokenizer + Param	**Training Accuracy**	**Testing Accuracy**
WhitespaceTokenizer	99.59108%	98.16594%
WordPunctTokenizer	99.21137%	97.64192%
TreebankWordTokenizer	99.21137%	98.25328%
TweetTokenizer *-preserve_case: False* *-strip_handles: False*	99.698178%	98.77729%
TweetTokenizer *-preserve_case: False* *-strip_handles: True*	99.64950%	98.68996%
TweetTokenizer *-preserve_case: True* *-strip_handles: False*	99.73712%	98.34061%
TweetTokenizer *-preserve_case: True* *-strip_handles: True*	99.66897%	98.25328%

Based on this data, we can see that, while every Tokenization policy does extremely well on both the training and testing data sets, some minute performance differences on the testing data accuracies can be observed.

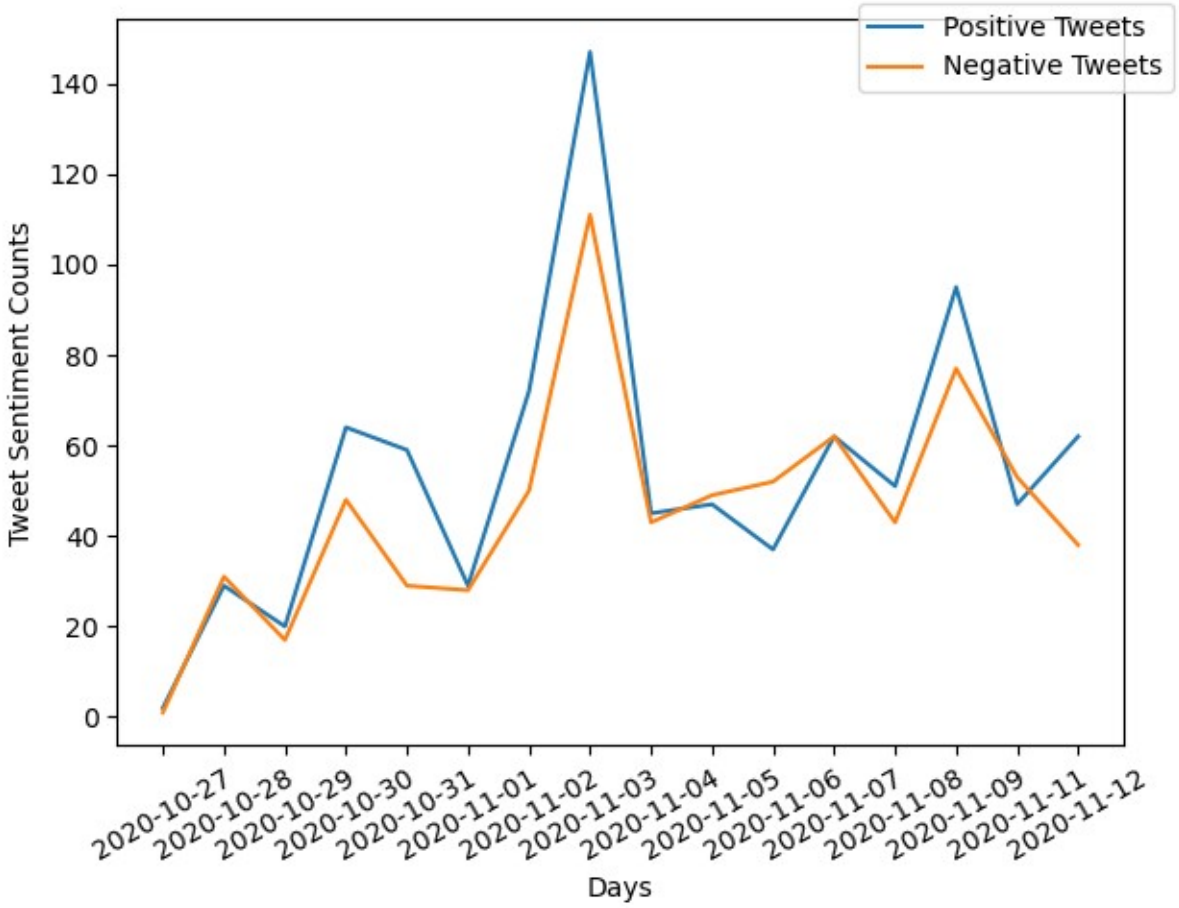
Firstly, it's surprising[8] to see that even the naive *Whitespace* tokenizer, which takes a tweet (represented as a single long string) and splits tokens based purely on whitespace, does quite well both on the training and testing data. This is mostly an indication that Naive Bayes can do surprisingly well in this scenario of binary labelling.

Given that there was no clear separation in performance among all of these Tokenization strategies, I decided to use the *TweetTokenizer* with params {*preserve_case: False, strip_handles: False*} for analyzing the election tweets for a couple of reasons. Firstly, stripping handles made little sense for election sentiment analysis, given that retweets of prominent political figures would probably have some effect on the ability of the Bayes classifier to predict sentiment e.g. @realdonaldtrump. Secondly, per the above tables: regardless of the other parameter, we can see testing accuracy improvement when we do not preserve case.

8 To me

Section 2: Election Twitter

Having decided to use the *TweetTokenizer* with params {*preserve_case: False*, *strip_handles: False*}, the per-day sentiment count predictions are shown in the plot below.



Here, we can see that, leading up to the election day (2020-11-03), there is a somewhat stable (but not monotonic) uptick in tweet volume within the given data set, *Tweets_election_trial.txt*. On the day of the election, we see a punctuated maximal value in both (predicted) positive and negative sentiment, with a sharp drop in user engagement on the day after.

Just a few days later, we can see a slight but erratic increase in election engagement, probably due to the fact that several swing states were still officially tallying up their vote counts, though I couldn't find any one event to which I could attribute this local engagement maximum.

Notably, throughout this timeline, there is nearly always a bias toward predicting positive sentiment, with November 6 being the lone (known[9]) day where there noticeably more negative sentiment tweets than positive. Twitter is typically known for being a somewhat left-leaning platform, so the bias toward positive sentiment, especially post-election, makes some sense.

9 Curiously, there were no tweets dated November 10 in *Tweets_election_trial.txt*